Game Leaderboard

Algorithm – CS0007

Dr. Hadji Tejuco

Members:

Junio Adrian - Lead programmer

Opulencia, Kaiser - Error handling, Documentation

Submission date: 1/30/25

**TABLE OF CONTENTS**

[**Introduction**](#_kggylamh4e58) **3**

[Purpose](#_u0t7mlsv2ndi) 3

[Scope](#_l5ftzd7hbc0i) 3

[**Project Overview**](#_uv1poo8rfouk) **3**

[Problem Statement](#_u5b6yec4s7i9) 3

[Key Features](#_9zxj3cvt8l3n) 3

[**Requirements Analysis**](#_rgsqb0aqw76z) **4**

[Functional Requirements](#_3x11lldikdp3) 4

[Non-Functional Requirements](#_vxv2u7g1wzbn) 4

[**System Design**](#_t70jczyp1bb2) **4**

[**Implementation**](#_4ld2869xvmb6) **5**

[Technologies Used](#_e5a7j3wdbtj1) 5

[**Testing**](#_d0z14parmpw) **5**

[case 1: Adding player name & score](#_c33601iys6sc) 5

[Case 2:display leaderboard](#_l4uxciou0x4g) 6

[Case 3: Updating player score](#_dko7xq48mnqd) 6

[Case 4: Add 11th player to replace a player in the leaderboard](#_ednwswuik68r) 7

[Case 5: Save into txt file](#_1cdym48zqlx8) 7

[Case 6: Adding player that already have a record](#_8i6a8q9y4pg) 8

[Case 7: Error handling (user input mismatch)](#_lhhtc59dqehx) 9

[**User Manual**](#_z54w2pn14nza) **9**

[Installation Guide](#_3ltym2qc0elf) 9

[Usage Instructions](#_ar97732776jz) 12

[Start up](#_9a8sdd4ujjb2) 12

[[1] Add Player Record](#_whgxu2ir3k5x) 12

[[2] Update Player Record](#_r4mz86m9bltb) 12

[[3] Display Leaderboard](#_r40bfaeprfxt) 12

[[4] Save Leaderboard to File](#_upfnet6aub5y) 13

[[5] Exit](#_j3hx2gajbtcj) 13

[Challenges and Solutions](#_getp5hw0ip2a) 13

[Future Enhancements](#_lf085zk3k0lr) 14

[Conclusion](#_uu4nblpn7272) 14

[References](#_25cc1vi33vw4) 14

[Appendices](#_ahn4n5xrwfdk) 15

[Appendix A: Additional diagrams or charts.](#_qo1d4lcwmsph) 15

[Appendix B: Complete Code](#_no3ck5fctfcy) 15

# Introduction

## Purpose

Videogames has been a long standing entertainment, from the creation of “*Tennis for two”* way back in 1958 to the modern indie developed games and company backed studio of triple A games. The purpose of this program is to recreate those types of score leaderboard tracking and to learn how to optimize the system.

**Objectives**

* Make a game leaderboard that tracks the player and their score
* Use Priority Queue to manage the top 10 scores
* Use Linked list to store all player scores
* Use Array to store the top 10 scores for quick display

## Scope

**Included:**

* Add/Update Player Scores
* Display top 10 player scores
* File saving for the top 10 player scores

**Excluded:**

* Load savefile to continue the leaderboard
* Simple game to go with the leaderboard
* GUI interface

# Project Overview

## Problem Statement

Technology evolved and videogames followed after but the core essence of games is to track scores between multiple players with the old Pac-Man in arcade machines or a single player game Sifu that tracks your score as you improve on clearing the different stages. How can the game leaderboard be improved upon?

## Key Features

The program accepts new player names and scores and tracks them together to be listed onto a top 10 leaderboard that is in descending order to show the top highest score at the top and the lowest at the bottom.

# Requirements Analysis

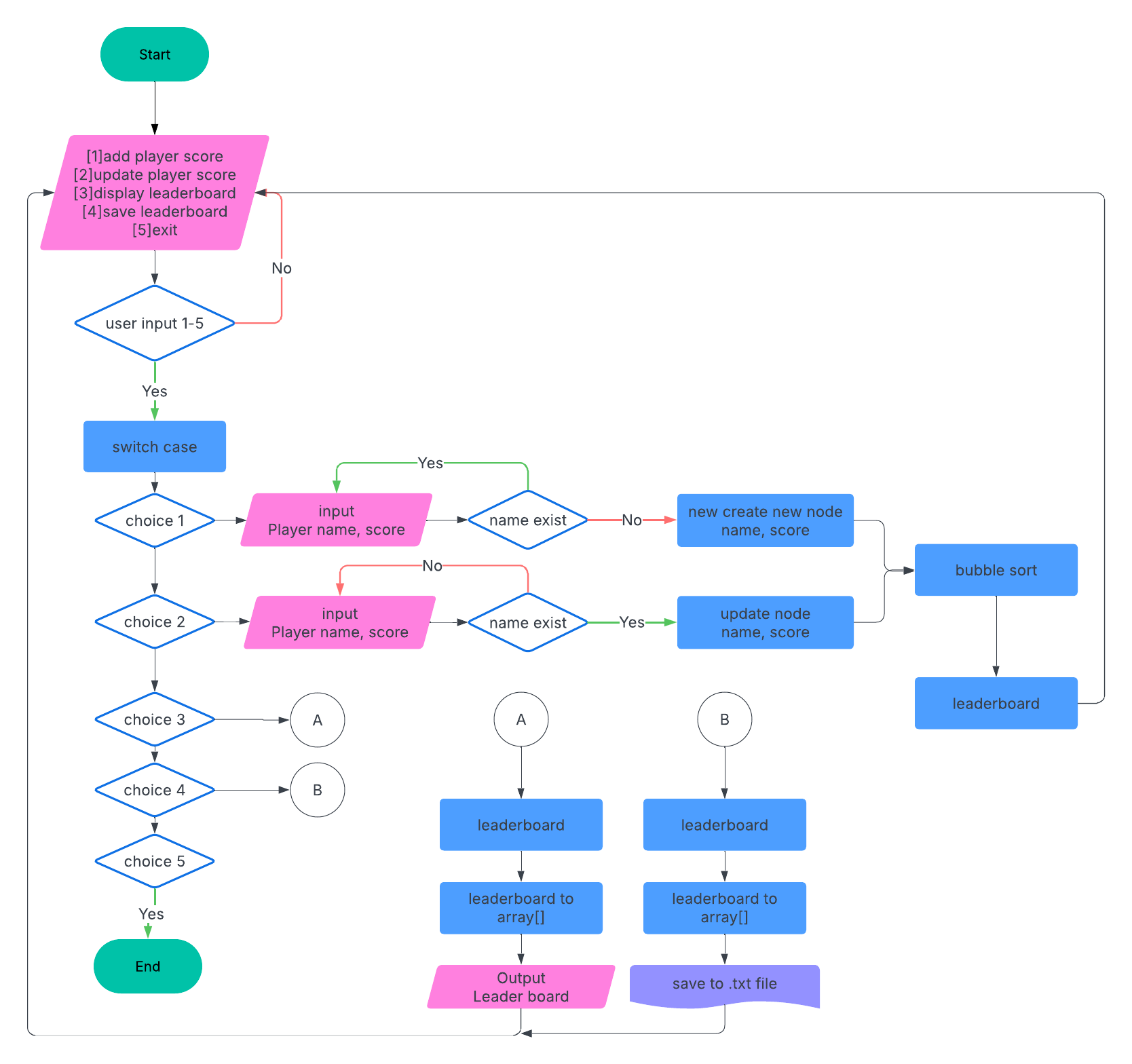
## Functional Requirements

* When adding or updating the player score the input of the score should only be numbers and not other characters
* Must be in c++
* Use UDF, no special algorithms

## Non-Functional Requirements

* Quick and responsive program that takes an average of 1.14seconds to compile
* User interface is clean and readable
* Only usable if user has c++ compiler to run the program
* Doesn’t have encryption to protect the data in the saved file

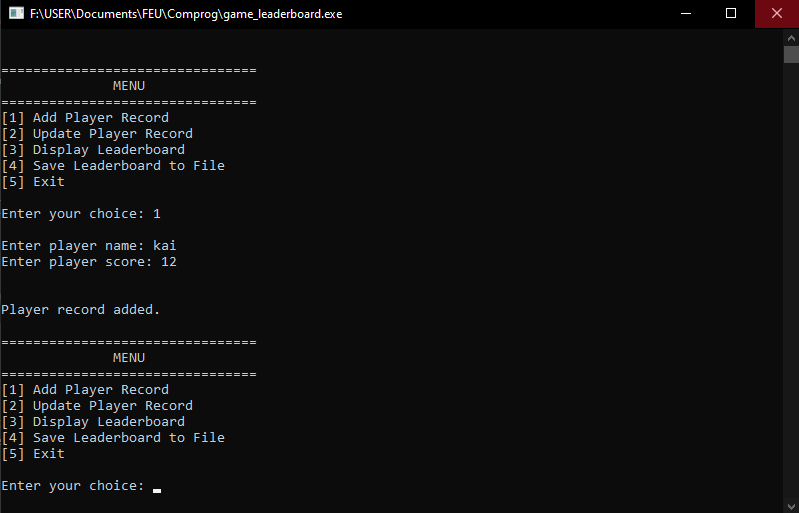
# System Design



# Implementation

## Technologies Used

* Language uses is C++
* Iostream for input output, fstream for file handling, and string for string manipulation
* DevC++, onlineGDB, Github



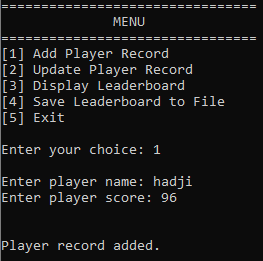
# Testing

## case 1: Adding player name & score

Enter choice: 1 add player record

Enter player name: hadji

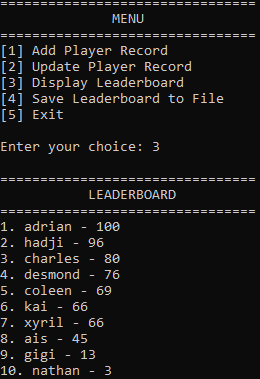
Enter player score: 96



The user when selecting the first option should be able to input the name of the player and the score

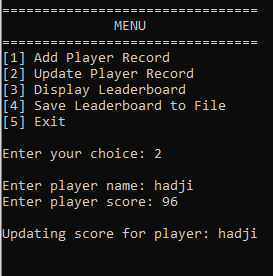
## Case 2:display leaderboard

Enter choice :3



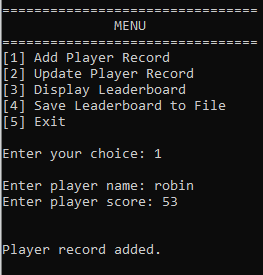
Simple output and should only be the list of the players and their score in the leaderboard in a descending

## Case 3: Updating player score

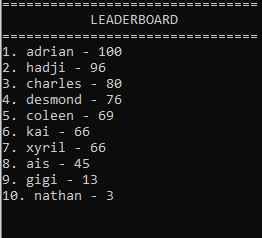
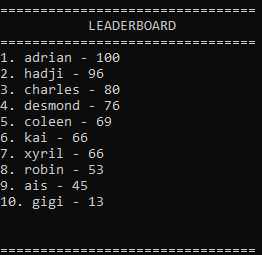


It asks the user to enter the player name and look for it in the leaderboard, if the player does not exist then prompting that it doesn't exist. If it exists then it will update the score to the new one.

## Case 4: Add 11th player to replace a player in the leaderboard



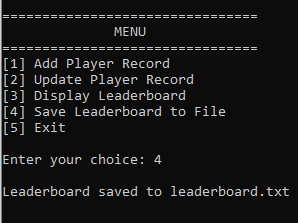
Before: After:

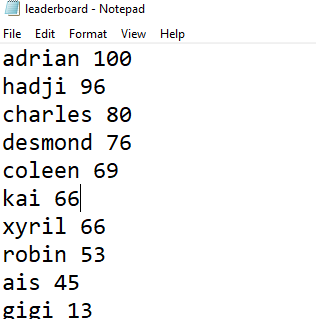
When a new player is added it the program will determine if the new player is in the top 10, if it is then the lowest player in the top 10 will be removed from the array.

## Case 5: Save into txt file

Enter choice: 4

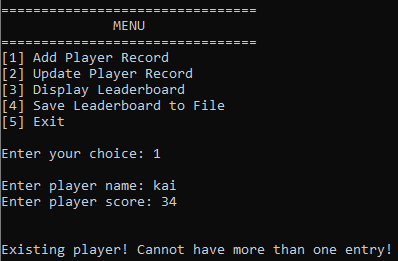


Text file should be in the same folder of the executable



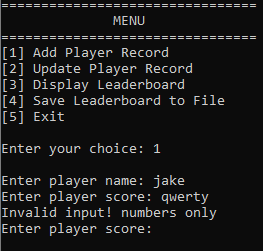
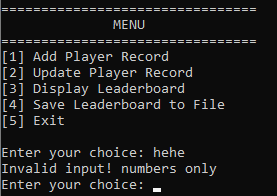
With file handling the stored array of top 10 players will be printed on to each line on to a text file

## Case 6: Adding player that already have a record



When adding a new player in the record if the new name is identical to the stored name then the program will not allow that player name to be used prompting a different name.

## Case 7: Error handling (user input mismatch)



Most of the error handling is mismatch of user input from the required input

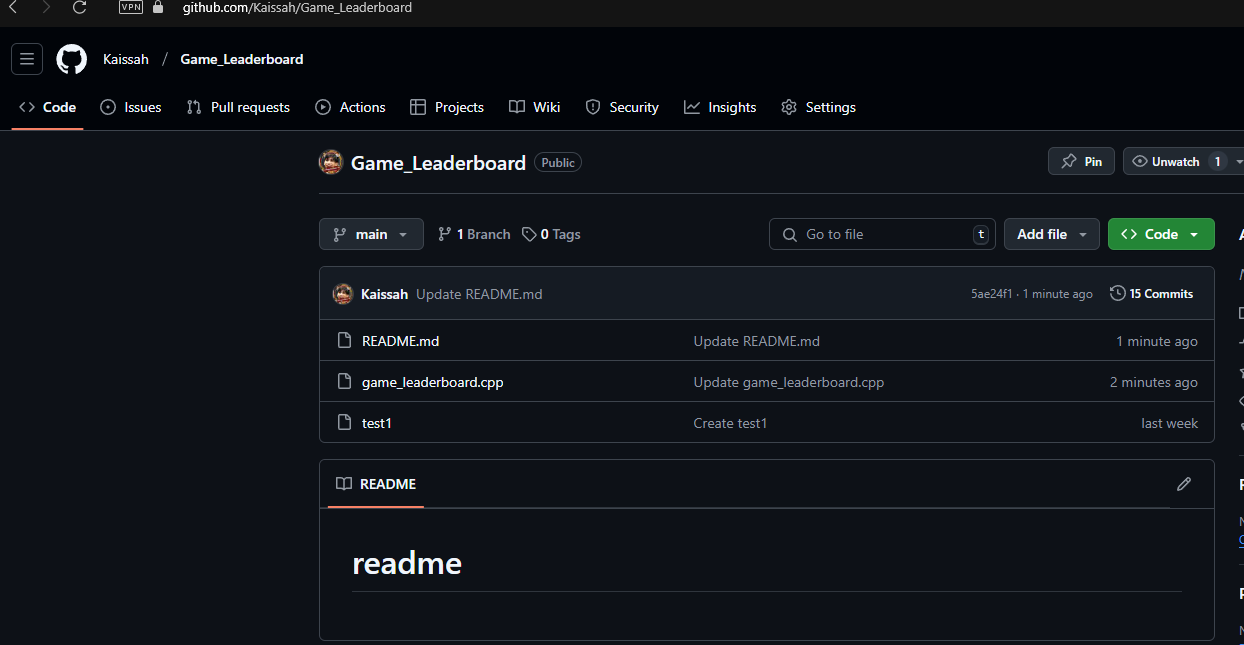
# User Manual

## Installation Guide

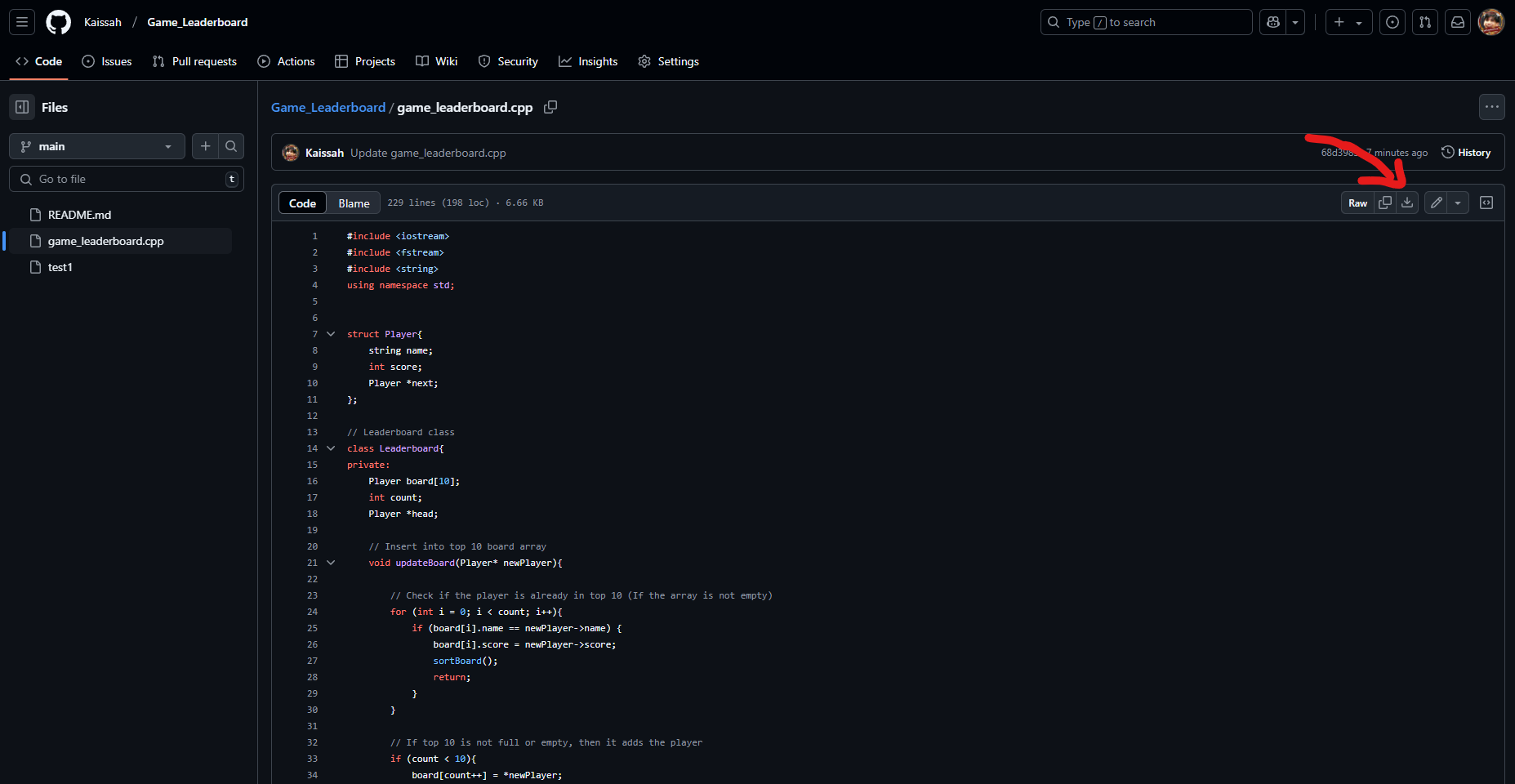
1. Install Dev C++ and follow their installation (<https://www.bloodshed.net>)



1. Go to <https://github.com/Kaissah/Game_Leaderboard>



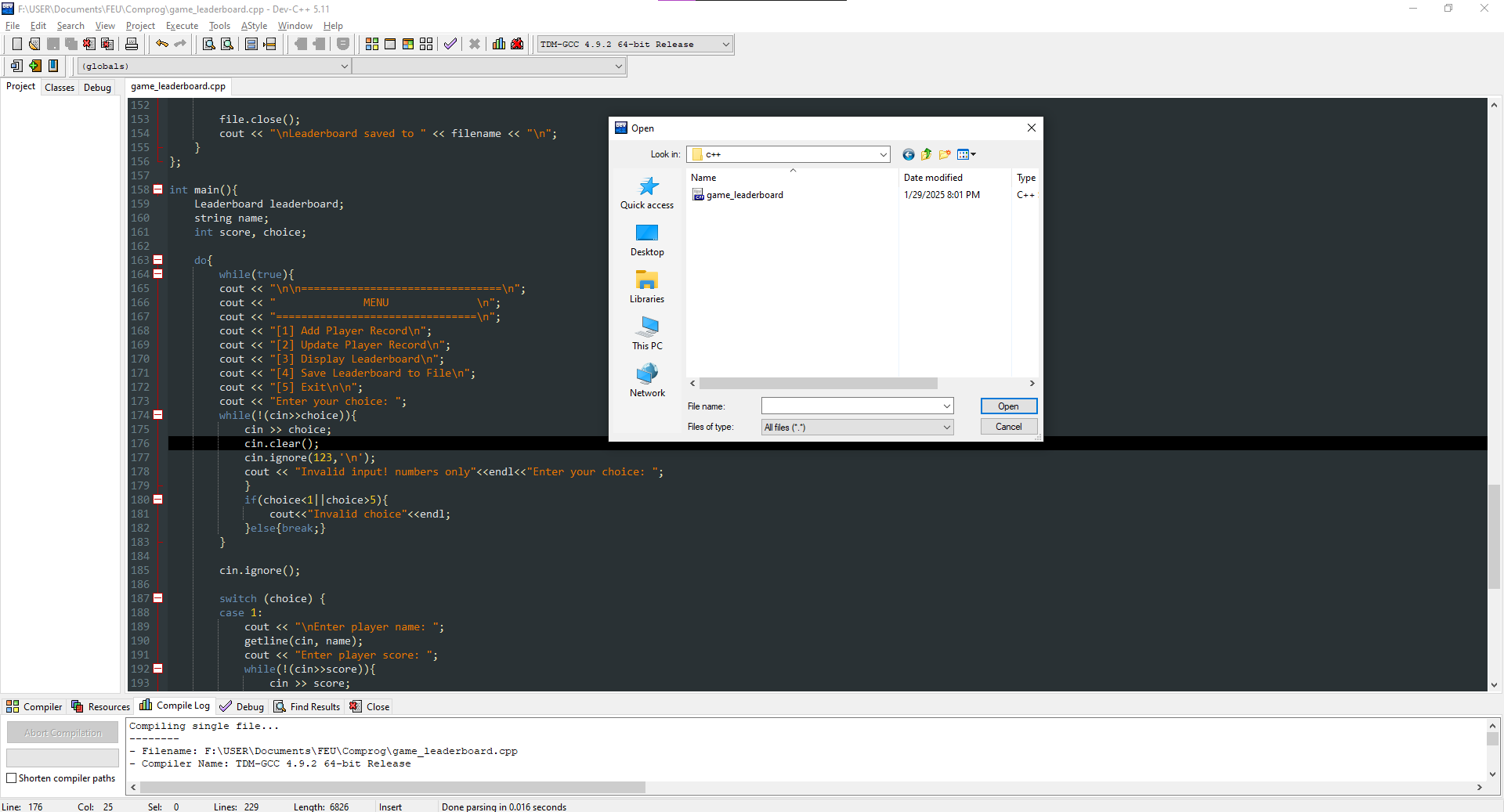
1. Select “game\_leaderboard.cpp” and download the files 



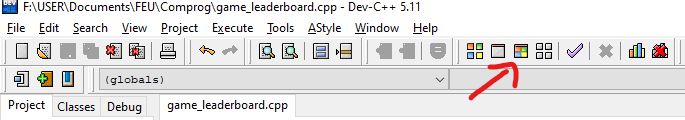
1. Select file location for the download



1. Open devC++ > open files > game\_leaderboard.cpp



1. Run and compile the program by pressing F11 or the  button. You should have a .exe file in the same file location



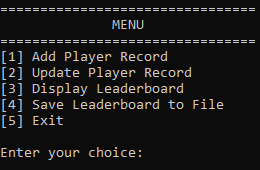


1. Application program is ready to use!

## Usage Instructions

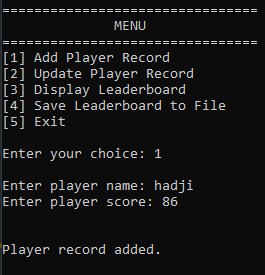
### Start up

When you open the .exe executable file you should be greeted with this user interface that will show you the options to choose from.



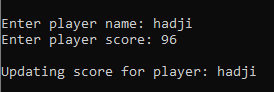
### [1] Add Player Record

When adding a new player score the program will ask for player name and player score, the input on the player name is not restricted unlike the player score that only strictly collects numerical inputs. If the user input the wrong type it will be prompted to retry the attempt on adding the score



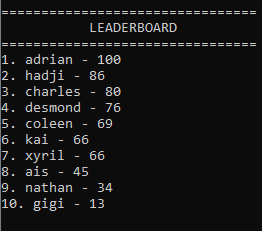
### [2] Update Player Record

When updating the player score the player should initially exist in the leaderboard if the name of the player is not found then that mean that the player does not exist thus need to go to the **[1] Add player Record** menu to add the player. If the player exist then the old score will be overwritten with the new score updating the user that the player has been updated



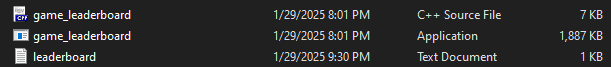
### [3] Display Leaderboard

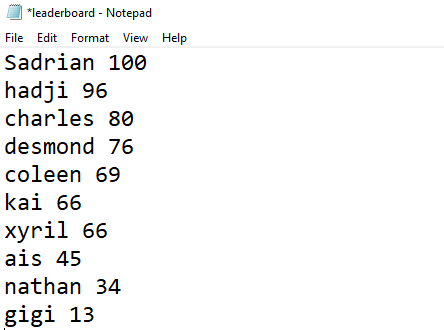
Upon selecting the 3rd option, display leaderboard the program will showcase the top 10 players in the leaderboard as shown the the figure below



### [4] Save Leaderboard to File

When this option is selected the program will create a text file named leaderboard.txt that has the latest leaderboard scores and it will be saved where the file location of the executable.





Contents of the file created

### [5] Exit

After you are done with the program you may select **[5] Exit** to close the program.



## Challenges and Solutions

There were several problems we encountered. The earliest one is traversing through the list to organize it in a descending order and to play where the new player will be placed in the list. Next was in error handling where the program would infinite loop if the user input the wrong data type in the input. Lastly there were several times that the notion of creating a mini game in the program like a rock-paper-scissors that will keep track of the players score and will store it once the win streak ends.

How the solution is implemented is that with the sorting of the scoreboard, the priority queue that is used is the bubble sort to organize each player node to its corresponding score. Second problem’s solution is to use a while loop that checks and repeats its prompt until the user input the desired input. Lastly it would take too long to create an additional minigame with the program due to time constraints but it would be wonderful to tackle the minigame creation as another project with the backbones of the leaderboard system of this project, it would be easily implemented onto the other project.

## Future Enhancements

Future improvements could be in the program that there should be a feature that save files could be loaded back into the system after restarting the application. The program should also have a Graphical User Interface for the streamline experience for the user. Lastly there could be a faster sorting algorithm that can be implemented, for the mean time the player count is small but when it grows bigger the old sorting algorithm won't be able to handle that much load.

## Conclusion

In the end the program runs as intended from organizing players with their scores to the creation of a save file. The program takes in player name and player score to be sorted into the top 10 players in the leaderboard with every addition of a new player an old player gets moved down or up the leaderboard. During the creation of the project the learning point is that the value of a sorting algorithm can streamline the process of the organizing of multiple players, and how linked lists should be used in the majority of the project for it dynamically changing size depending on the runtime of the user. Thinking outside of the box is a big help not just in solving problems but improving onto an idea, e.g. this leaderboard project will not show it full potential if there are no games or system attached to it, and it will make the leaderboard value not really needed.

## References

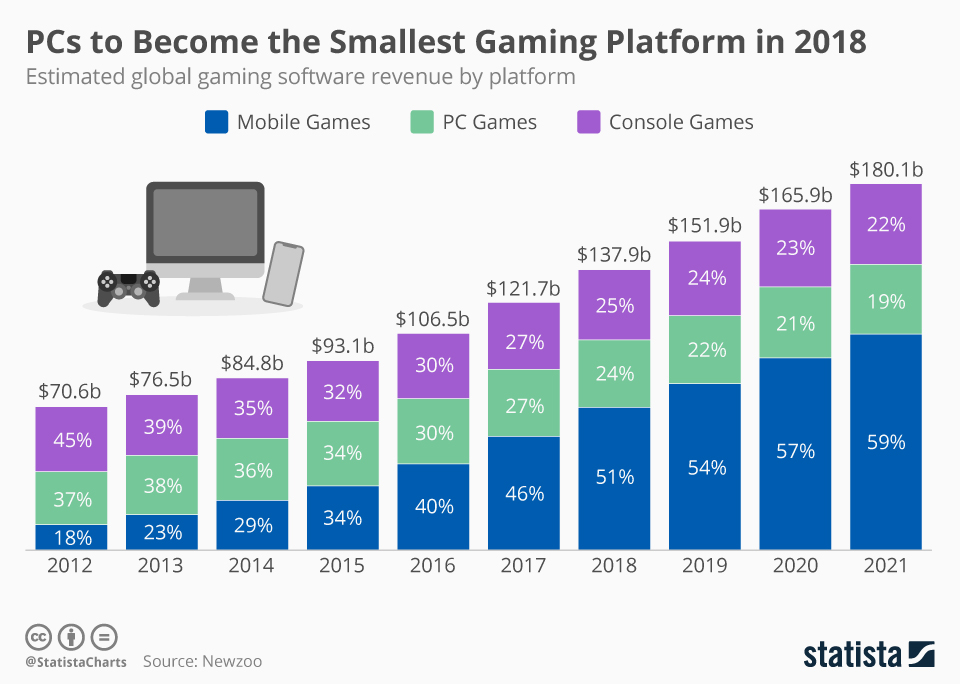
Tennis for Two (2025, January 24). In Wikipedia <https://en.wikipedia.org/wiki/Tennis_for_Two>

Research by Jörnmark, Axelsson, Ernkvist (2005) Wherever Hardware, There’ll be Games: The Evolution of Hardware and Shifting Industrial Leadership in the Gaming Industry <https://dl.digra.org/index.php/dl/article/download/205/205>

Richter, F. (2018). PCs to Become the Smallest Gaming Platform in 2018 https://www.statista.com/chart/13789/worldwide-video-game-revenue-forecast/

## Appendices

### Appendix A: Additional diagrams or charts.



### Appendix B: Complete Code

| #include <iostream>  #include <fstream>  #include <string>  using namespace std;  struct Player{  string name;  int score;  Player \*next;  };  // Leaderboard class  class Leaderboard{  private:  Player board[10];  int count;  Player \*head;  // Insert into top 10 board array  void updateBoard(Player\* newPlayer){  // Check if the player is already in top 10 (If the array is not empty)  for (int i = 0; i < count; i++){  if (board[i].name == newPlayer->name) {  board[i].score = newPlayer->score;  sortBoard();  return;  }  }  // If top 10 is not full or empty, then it adds the player  if (count < 10){  board[count++] = \*newPlayer;  sortBoard();  } else{  // Check if the new player's score is higher than the 10th player in the leaderboard (triggers if and only if the array is full)  if (newPlayer->score > board[count - 1].score) {  board[count - 1] = \*newPlayer;  sortBoard();  }  }  }  // Sort top 10 array by score (descending) (Temporary: This should be the priority queue part)  void sortBoard() {  for (int i = 0; i < count - 1; i++) {  for (int j = 0; j < count - i - 1; j++) {  if (board[j].score < board[j + 1].score ||  (board[j].score == board[j + 1].score && board[j].name > board[j + 1].name)) {  Player temp = board[j];  board[j] = board[j + 1];  board[j + 1] = temp;  }  }  }  }  public:  Leaderboard(){  head = nullptr;  count = 0;  }  // Add or update player  void addPlayer(string name, int score){  if (score < 0 || score > 100){  cout << "Invalid score. Enter a score between 0-100.\n"; // Error handling for invalid score.  return;  }  Player\* current = head;  while (current != nullptr){  if (current->name == name){  cout << "\n\nExisting player! Cannot have more than one entry!"; // Error handling for multiple entries  return;  }  current = current->next;  }  // If list is empty or no matching player is found; add new player to the linked list.  Player \*newPlayer = new Player;  newPlayer->name = name;  newPlayer->score = score;  newPlayer->next = head;  head = newPlayer;  updateBoard(newPlayer);  cout << "\n\nPlayer record added.";  }  void updatePlayer(string name, int score){  if (!head)  cout << "\n\nNo records found. Player list is empty.";  else{  if (score < 0 || score > 100){  cout << "Invalid score. Enter a score between 0-100.\n"; // Error handling for invalid score.  return;  }  // Search for the player in the linked list if the list is not empty.  bool found = 0;  Player \*current = head;  while (current != nullptr){  if (current->name == name){  found = 1;  cout << "\nUpdating score for player: " << name << "\n";  current->score = score;  updateBoard(current);  return;  }  current = current->next;  }  if (found == 0)  cout << "\n\nNo matching player found.";  }  }  // Display top 10 players using the array.  void displayBoard(){    if (count == 0)  cout << "\n\nLeaderboard is currently empty.";  else{  cout << "\n================================\n";  cout << " LEADERBOARD \n";  cout << "================================\n";  for (int i = 0; i < count; i++) {  cout << i + 1 << ". " << board[i].name << " - " << board[i].score << "\n";  }  }  }  // Save leaderboard to file  void saveToFile(const char\* filename){  ofstream file(filename);  if (!file) {  cout << "Error: File does not exist.\n"; // Error handling for files  return;  }  for (int i = 0; i < count; i++){  file << board[i].name << " " << board[i].score << "\n";  }  file.close();  cout << "\nLeaderboard saved to " << filename << "\n";  }  };  int main(){  Leaderboard leaderboard;  string name;  int score, choice;  do{  while(true){  cout << "\n\n================================\n";  cout << " MENU \n";  cout << "================================\n";  cout << "[1] Add Player Record\n";  cout << "[2] Update Player Record\n";  cout << "[3] Display Leaderboard\n";  cout << "[4] Save Leaderboard to File\n";  cout << "[5] Exit\n\n";  cout << "Enter your choice: ";  while(!(cin>>choice)){  cin >> choice;  cin.clear();  cin.ignore(123,'\n');  cout << "Invalid input! numbers only"<<endl<<"Enter your choice: ";  }  if(choice<1||choice>5){  cout<<"Invalid choice"<<endl;  }else{break;}  }    cin.ignore();  switch (choice) {  case 1:  cout << "\nEnter player name: ";  getline(cin, name);  cout << "Enter player score: ";  while(!(cin>>score)){  cin >> score;  cin.clear();  cin.ignore(123,'\n');  cout << "Invalid input! numbers only"<<endl<<"Enter player score: ";  }  cin.ignore();  leaderboard.addPlayer(name, score);  break;  case 2:  cout << "\nEnter player name: ";  getline(cin, name);  cout << "Enter player score: ";  while(!(cin>>score)){  cin >> score;  cin.clear();  cin.ignore(123,'\n');  cout << "Invalid input! numbers only"<<endl<<"Enter player score: ";  }  cin.ignore();  leaderboard.updatePlayer(name, score);  break;  case 3:  leaderboard.displayBoard();  break;  case 4:  leaderboard.saveToFile("leaderboard.txt");  break;  case 5:  cout << "\n\nThank you for using the program!";  break;  default:  cout << "Invalid choice. Please try again.\n";  }  } while (choice != 5);  return 0;  } |
| --- |