

C++ 프로그래밍및실습

사설 게임 기록분석

진척보고서#3

제출일자: 2024/12/15

제출자명: 배재일

제출자학번: 215561

1. 프로젝트 목표 (16 pt)

1) 배경 및 필요성 (14 pt)

리그오브레전드(LoL)는 나와 비슷한 나이의 사람들이 즐겨 플레이하는 인기 게임으로, 솔로 플레이어에게는 랭크전에서 즐거운 경험을 제공한다. 하지만 많은 플레이어들에게 친구들끼리 두팀으로 나누어 플레이하는 사설 게임이 진정한 리그오브레전드의 재미라고 여겨진다. 이러한 사설 게임에서는 팀원과 함께 전략을 구상하며 문제점을 피드백하며 정말 재미있는 경험을 공유할 수 있다. 이러한 사설 게임의 가치에도 불구하고, 리그오브레전드를 운영하는 Riot Games는 2021년 이후로 사설 경기의 전적을 제공하지 않고 있다.

게임 통계에 접근할 수 있는 것은 또 다른 즐거움을 더해 준다고 생각한다. 이는 플레이어가 자신의 성과를 확인하고 개선할 부분을 깨닫고, 친구들과 게임에 대해 여러가지 관점으로 이야기할 수 있게 해준다. 이러한 데이터가 없다면 플레이어들은 위와 같은 즐거움을 놓치게 된다. 이러한 문제를 해결하기 위해 이러한 프로그램을 구상했다.

2) 프로젝트 목표

본 프로젝트의 목표는 각 경기의 모든 플레이어의 전적을 기록하고, 이를 통해 개인과 팀 단위의 지표를 분석할 수 있는 프로그램을 목표로 한다.

3) 차별점

리그 오브 레전드에서 게임 전적을 기록하는 대부분의 기존 사이트는 공식 랭크 게임의 통계와 순위에만 초점을 맞추고 있다. 이러한 플랫폼은 공식 랭크 게임과 모든 플레이어가 포함된 공식 랭킹에 대한 정보를 제공하지만, 친구나 가까운 지인과 함께 플레이하는 사설 게임의 전적을 알려주지 않고 있다.

이 프로그램은 친구들 또는 지인과의 사설 게임의 전적을 기록하고 친구/지인 그룹에서 나의 위치를 파악할 수 있는 점에서 차별점이 있다.

2. 기능 계획

1) 기능 1: 경기 데이터 입력

- 설명: 각 경기의 모든 플레이어의 전적을 입력 받는 기능

(1) 세부 기능 1: 플레이어 정보 수집

- 설명: 각 경기마다 각 플레이어의 이름, 역할, 챔피언, 킬, 데스, 어시스트 수 등을 입력 받는다

2) 기능 2 : 지표 분석

- 설명: 입력된 데이터에 대한 기본 통계 분석

(1) 세부 기능 1: 개인의 평균 지표 분석

- 설명: 모든 경기의 데이터에서 평균 킬, 데스, 어시스트 등 관련 데이터를 계산하여 출력한다.

(2) 세부 기능 2: 각각의 챔피언 별 지표 분석

- 설명: 입력된 경기 결과를 바탕으로 챔피언별 지표를 계산하고 출력한다.

3) 기능 3: 기록 조회

- 설명: 특정 플레이어의 경기 기록을 조회하는 기능

(1) 세부 기능 1: 플레이어 전적 조회

설명: 사용자가 입력한 플레이어의 이름을 통해 해당 플레이어의 모든 경기 기록을 출력한다.

(2) 세부 기능 2: 특정 경기 분석

설명: 특정 경기에 참여한 모든 플레이어의 지표를 출력한다.

3.

1) 기능 구현

(1) 데이터 입력 기능

- 입출력 : match ID = 매치 이름 name = 이름, role = 포지션, champion = 챔피언, kll= 킬, death= 데스, assist = 어시스트 을 입력
- 설명 : 한 번에 10명의 플레이어에 대한 데이터를 입력하며 또한 다음 데이터를 연속으로 적을지 선택한다
- 적용된 배운 내용:

반복문 (do-while): 10명의 데이터를 입력할 수 있도록 do-while문을 사용했다

조건문 (if): 사용자가 매치를 계속 추가할지 말지를 결정하는 부분에서 if 조건문을 사용하여 'y'일 경우 계속 입력받고, 'n'일 경우 종료된다

- 코드 스크린샷

```
int main() {
    StatsTracker tracker;
    char moreData;

    do {
        // 한 줄로 모든 데이터 입력받기
        string inputLine;
        cout << "Enter match ID (e.g., match1, match2, etc.): ";
        string matchID;
        cin >> matchID;
        cin.ignore();

        // 10명의 데이터 입력
        for (int i = 1; i <= 10; ++i) {
            string playerName, role, champion;
            int kills, deaths, assists;

            cout << "Enter data for Player " << i << " (format: Name Role Champion Kills Deaths Assists): ";
            getline(cin, inputLine);

            stringstream ss(inputLine);
            ss >> playerName >> role >> champion >> kills >> deaths >> assists;

            // 매치 데이터 추가
            tracker.addMatch(Match(playerName, role, champion, kills, deaths, assists));
        }

        // 새 매치를 더 입력할지 여부 묻기
        cout << "Do you want to enter another match? (y/n): ";
        cin >> moreData;
        cin.ignore();

    } while (moreData == 'y');

    return 0;
}
```

(2) 개인의 평균 분당 데미지 분석

- 입출력 :

playerName: 평균 분당 데미지를 계산할 특정 플레이어의 이름, matches: 이전에 입력된 경기 데이터들

출력: 해당 플레이어의 평균 분당 데미지

- 설명 : 플레이어의 데미지 총합을 playtime 총합으로 나누어서 평균 분당 데미지를 나타낸다

- 적용된 배운 내용: for 반복문을 사용하여 matches 벡터에 저장된 모든 경기 데이터를 순차적으로 확인하게 하고 if 조건문을 사용하여 특정 플레이어의 데미지와 플레이시간을 호출하여 계산하였다

```
// 특정 플레이어의 평균 분당 데미지 계산
double calculateAverageDamagePerMinute(const string& playerName) {
    double totalDamage = 0;
    int totalPlaytime = 0; // 총 플레이 시간
    int matchCount = 0;

    for (const auto& match : matches) {
        if (match.playerName == playerName) {
            totalDamage += match.damage;
            totalPlaytime += match.playtime; // 모든 경기의 총 플레이 시간을 더함
            matchCount++;
        }
    }

    if (matchCount == 0) return 0.0; // 해당 플레이어의 경기 기록이 없으면 0 반환
    return totalPlaytime == 0 ? 0.0 : static_cast<double>(totalDamage) / totalPlaytime; // 분당 데미지 계산
```

(3) 해당 플레이어의 평균 KDA(kill/death/assist) 분석

- 입출력: playerName: 평균 KDA를 계산할 특정 플레이어의 이름, matches : 각 경기의 데이터를 저장하고 있으며, 각 경기 데이터에는 해당플레이어의 킬(kills), 데스(deaths), 어시스트(assists)가 포함

해당 플레이어의 평균 KDA가 반환됨. KDA는 (킬 + 어시스트) / 데스로 계산됨,

- 설명: 해당 플레이어의 모든 경기에서 KDA를 계산하고 이를 누적하여 평균값을 구합니다 KDA를 계산할 때, deaths가 0인 경우를 처리하여 0으로 나누는 오류를 방지하고, 모든 경기의 KDA를 합산한 후 평균 KDA를 계산함

-적용된 배운 내용: for 반복문을 사용하여 matches 벡터에 저장된 모든 경기 데이터를 순차적으로 확인하게 하고 if 조건문을 사용하여 각각의 상황에 따른 KDA 계산을 하였다

```
// 특정 플레이어의 평균 KDA 계산
double calculateAverageKDA(const string& playerName) {
    double totalKDA = 0.0;
    int matchCount = 0;

    for (const auto& match : matches) {
        if (match.playerName == playerName) {
            double kda = match.deaths == 0 ? match.kills + match.assists
            : static_cast<double>(match.kills + match.assists) / match.deaths;
            totalKDA += kda;
            matchCount++;
        }
    }

    if (matchCount == 0) return 0.0; // 플레이어 기록이 없으면 0 반환
    return totalKDA / matchCount;
}
```

(4) 챔피언별 승률 계산

- 입출력:

championName: 승률을 계산할 챔피언의 이름, matches :각 경기 데이터를 저장하고 있으며, 각 경기에는 해당 챔피언(champion)과 해당 경기에서의 승패(win) 정보가 포함됨,

해당 챔피언의 **승률(%)**이 반환됨.

- 설명: 각 경기에서 해당 챔피언이 승리했는지 확인하고, 승리한 경기는 totalWins를 증가시키고, 전체 경기는 totalMatches를 증가시킨후 totalMatches가 0이 아닌 경우 $(totalWins / totalMatches) * 100$ 을 계산하여 승률을 반환한다.

만약 해당 챔피언이 한 경기도 출전하지 않았으면 승률은 0.0%로 반환한다

- 적용된 배운 내용: for 반복문을 사용하여 matches 벡터에 저장된 모든 경기 데이터를 순차적으로 확인하고 if 조건문을 사용하여 각 경기에서 해당 챔피언 이름과 일치하는 경기만을 찾았다

```

// 챔피언별 승률 계산
double calculateWinRateByChampion(const string& championName) {
    int totalMatches = 0;
    int totalWins = 0;

    for (const auto& match : matches) {
        if (match.champion == championName) {
            totalMatches++;
            if (match.win) {
                totalWins++;
            }
        }
    }

    if (totalMatches == 0) return 0.0; // 해당 챔피언의 경기 기록이 없을 때
    return static_cast<double>(totalWins) / totalMatches * 100.0; // 승률(%)
}

```

(5) 플레이어의 전적조회

- 입출력:

playerName: 통계를 확인할 플레이어 이름.

matches: 경기 데이터를 저장한 벡터 totalKills, totalDeaths, totalAssists: 총 킬, 데스, 어시스트 totalDamage: 총 데미지 totalPlaytime: 총 경기 시간을 저장한다.

matchCount: 플레이어가 참여한 경기 수를 저장

모든 경기에서 입력된 플레이어 이름을 찾아 통계를 모아서 킬, 데스, 어시스트, 데미지를 합치고, 참여한 경기 수를 세서 평균을 계산하여 결과로 평균 데미지와 평균 KDA를 반환한다. 만약 해당 챔피언이 한 경기도 출전하지 않았으면 승률은 0.0%로 반환한다

- 적용된 배운 내용: for 반복문을 사용하여 matches 벡터에 저장된 모든 경기 데이터를 순차적으로 확인하고 if 조건문을 사용하여 특정 플레이어의 데이터를 찾아내었다.

```

void displayPlayerStats(const string& playerName) const {
    int totalKills = 0, totalDeaths = 0, totalAssists = 0;
    int totalDamage = 0, totalPlaytime = 0;
    int matchCount = 0;

    for (const auto& match : matches) {
        auto processTeam = [&](const vector<PlayerStats>& team) {
            for (const auto& player : team) {
                if (player.playerName == playerName) {
                    totalKills += player.kills;
                    totalDeaths += player.deaths;
                    totalAssists += player.assists;
                    totalDamage += player.damage;
                    totalPlaytime += match.playtime;
                    matchCount++;
                }
            }
        };

        processTeam(match.blueTeam);
        processTeam(match.redTeam);
    }

    if (matchCount == 0) {
        cout << "No stats found for player: " << playerName << "\n";
        return;
    }

    double avgDamagePerMinute = totalPlaytime > 0 ? static_cast<double>(totalDamage) / totalPlaytime : 0.0;
    double avgKDA = totalDeaths == 0 ? totalKills + totalAssists : static_cast<double>(totalKills + totalAssists) / totalDeaths;

    cout << "\n[Player Statistics: " << playerName << "]\n";
    cout << "Matches Played: " << matchCount << "\n";
    cout << "Average Damage per Minute: " << avgDamagePerMinute << "\n";
    cout << "Average KDA: " << avgKDA << "\n";
}

```

(6) 챔피언별 승률조회

- 입출력:

championName: 승률을 계산할 챔피언의 이름. matches: 여러 경기 데이터를 저장하는 벡터

모든 경기를 확인하며 특정 챔피언이 출전했는지 찾아내어 출전한 경기 중 승리한 경기 수와 총 경기 수를 계산한다 승률(%)을 계산하여 출력합니다.출전 기록이 없으면 "데이터 없음" 메시지를 출력.

- 적용된 배운 내용: for 반복문을 사용하여 matches 벡터에 저장된 모든 경기 데이터를 순차적으로 확인하고 if 조건문을 사용하여 특정 챔피언의 데이터를 찾아내었다.


```

void displayChampionWinRate(const string& championName) const {
    int totalGames = 0, totalWins = 0;

    for (const auto& match : matches) {
        auto processTeam = [&](const vector<PlayerStats>& team, bool isBlueWin) {
            for (const auto& player : team) {
                if (player.champion == championName) {
                    totalGames++;
                    if (isBlueWin) totalWins++;
                }
            }
        };

        processTeam(match.blueTeam, match.blueTeamWin);
        processTeam(match.redTeam, !match.blueTeamWin);
    }

    if (totalGames == 0) {
        cout << "No stats found for champion: " << championName << "\n";
        return;
    }

    double winRate = static_cast<double>(totalWins) / totalGames * 100.0;
    cout << "\n[Champion Statistics: " << championName << "]\n";
    cout << "Win Rate: " << winRate << "%\n";
}
};

```

2) 테스트 결과

(1) 데이터 입력 기능

- 설명: 한 번에 10명의 플레이어에 대한 데이터를 입력하며 또한 다음 데이터를 연속으로 적을지 선택하는 기능
- 테스트 결과 스크린샷

```

Enter match ID (e.g., match1, match2, etc.): match3
Enter data for Player 1 (format: Name Role Champion Kills Deaths Assists): Alice Mid Ahri 10 2 8
Enter data for Player 2 (format: Name Role Champion Kills Deaths Assists): Bob Top Garen 5 4 3
Enter data for Player 3 (format: Name Role Champion Kills Deaths Assists): Charlie Jungle LeeSin 7 3 6
Enter data for Player 4 (format: Name Role Champion Kills Deaths Assists): Diana ADC Caitlyn 8 6 9
Enter data for Player 5 (format: Name Role Champion Kills Deaths Assists): Ethan Support Thresh 2 8 20
Enter data for Player 6 (format: Name Role Champion Kills Deaths Assists): Fiona Mid Syndra 9 4 5
Enter data for Player 7 (format: Name Role Champion Kills Deaths Assists): George Top Darius 6 7 2
Enter data for Player 8 (format: Name Role Champion Kills Deaths Assists): Helen Jungle Evelyn 8 5 6
Enter data for Player 9 (format: Name Role Champion Kills Deaths Assists): Ian ADC Jhin 10 3 7
Enter data for Player 10 (format: Name Role Champion Kills Deaths Assists): Jane Support Nami 3 4 18
Do you want to enter another match? (y/n):

```

(2) 플레이어의 전적조회

- 설명: 특정 플레이어의 kda, dpm등을 계산하여 보여줌

```
Enter player name: Alice
[Player Statistics: Alice]
Matches Played: 2
Average Damage per Minute: 800
Average KDA: 6.2
```

(3) 챔피언별 승률 조회

- 특정 챔피언의 승률을 조회한다

```
Choose an option: 3
Enter champion name: Ahri
[Champion Statistics: Ahri]
Win Rate: 50%
```

4. 계획 대비 변경 사항

1) 기능2

- 이전 ~12/1
- 이후 ~12/15
- 사유: 지표를 호출하는 여러가지 함수를 만들었지만 이것을 정확하게 테스트 하기 위해서는 기능3을 구현하는 과정에서 한번에 하는 편이 용이하다고 판단함

2) 지표분석

- 이전 원래는 **각각의 지표(예: 평균 KDA, 평균 분당 데미지)**를 계산하기 위해 별도의 함수를 만들어서 사용할 계획이었음.

calculateAverageDamagePerMinute() calculateAverageKDA()

와 같은 함수들을 각각 구현하고 호출할 계획이었음.

- 이후 지표를 보여주는 함수(displayPlayerStats) 안에서 각 지표를 바로 계산하고 처리하는 방식으로 변경함.

- 사유: 별도의 함수를 만들면 각 함수가 for 반복문을 사용하여 matches를 순회해야 함. 이 경우 여러 번 반복문이 실행되므로 비효율적일거라고 생각했으며 하나의 함수함수 작업하는 것이 더 직관적일거라고 생각하였음

3. 프로젝트 일정 (참고: 간트차트)

업무		11/3	11/10	11/17	11/24	12/1	12/8	12/15	12/22
제안선 작성		완료							
기능1	세부기능1		완료						
기능2	세부기능1				완료				
	세부기능2					완료			
기능3	세부기능1						부분 완료		
	세부기능2							----->	