# 1. 문제 정의

이 게임은 두 명의 플레이어가 번갈아가며 진행되며, 각 차례에 플레이어는 Enter 키를 입력하여 0부터 2까지의 범위에서 무작위로 3개의 숫자를 생성한다. 이 3개의 숫자가 모두 동일할 경우, 해당 플레이어가 승리하며, 게임은 승자가 나올 때까지 계속된다.

주요 문제
- 두 명의 플레이어가 게임에 참여하여 번갈아 가며 진행할 수 있도록 해야 한다.
- 각 차례마다 0~2 범위에서 무작위로 3개의 숫자를 생성해야 한다.
- 3개의 숫자가 모두 동일할 경우, 그 플레이어가 승리함을 출력해야 한다.

# 2. 문제 해결 방법

## 1. 클래스 구조 설계

### 1-1. Player 클래스
각 플레이어의 이름을 저장하고 관리하는 역할을 수행하는 클래스이며, 생성자에서 플레이어 이름을 매개변수로 받아 저장한다. 이 클래스는 간단한 데이터 저장 클래스이므로, 추가적인 기능은 필요하지 않다.

### 1-2. GamblingGame 클래스
게임의 로직과 진행을 처리한다. 두 명의 Player 객체를 포함하는 배열을 사용하여, 플레이어의 이름을 저장한다. 게임의 주요 진행 로직을 포함하여, 플레이어의 차례, 랜덤 숫자 생성, 승리 조건 체크 등을 처리한다.

## 2. 게임 로직 구현

### 2-1. 게임 시작 및 플레이어 입력
게임이 시작될 때 두 명의 플레이어 이름을 cin을 사용하여 입력받고, 이를 통해 각 Player 객체를 초기화한다.

### 2-2. 랜덤 숫자 생성
srand(time(0))을 사용하여 프로그램 시작 시 랜덤 시드를 설정한다. 이 설정은 프로그램이 매번 실행될 때마다 다른 무작위 숫자를 생성하도록 한다.
rand() % 3을 사용하여 0, 1, 2의 값을 무작위로 생성한다. 세 개의 숫자를 생성하기 위해 반복문을 사용하고, 숫자들은 플레이어의 차례에 따라 각기 다르게 생성됩니다.

### 2-3. 게임 진행 및 차례 전환
while (true) 루프를 사용하여 게임을 계속 진행한다. 게임은 무한 루프에서 플레이어의 차례를 번갈아가며 진행되며, 3개의 숫자가 모두 동일한 경우라는 조건에 도달하면 루프를 종

료한다. currentPlayer 변수를 사용하여 현재 차례의 플레이어를 결정하고, 각 플레이어의 차례가 끝나면 해당 변수를 업데이트하여 다음 플레이어의 차례로 전환한다.

### 3. 승리 조건 체크

3개의 랜덤 숫자를 생성한 후, 이를 배열에 저장한다. 숫자가 모두 동일한지를 비교하기 위해 간단한 조건문을 사용하여, 배열의 첫 번째, 두 번째, 세 번째 요소를 비교하여 모든 요소가 동일한 경우 승리 메시지를 출력하고 게임을 종료한다. 만약 숫자가 동일하지 않다면, "아쉽군요!"라는 메시지를 출력하고 다음 플레이어로 전환한다.

### 4. 메모리 관리

메모리 누수를 방지하기위해 GamblingGame 클래스의 소멸자를 정의하여 동적으로 생성한 Player 객체의 메모리를 해제한다. 프로그램이 종료될 때 모든 동적 할당된 메모리를 정리하여, 효율적인 메모리 관리를 보장한다.

## 3. 아이디어 평가

### 1. 클래스 구조 설계 평가

1-1. Player 클래스
플레이어의 이름만을 저장하는 클래스로 간단하게 설계되었지만, 불필요한 기능이나 복잡성이 없어 간결하고 직관적이며 효율적으로 설계되었다.

결과: 플레이어의 이름을 저장하고 이를 참조하는 과정에서 별다른 문제없이 정상적으로 작동한다. 추가적인 기능이 필요하지 않은 단순한 구조로 설계되었다.

1-2. GamblingGame 클래스
이 클래스는 게임의 전체 흐름을 관리하고, 두 명의 Player 객체를 관리하는데 적합하게 설계되었다. 게임 진행, 차례 전환, 승리 조건 체크 등 게임의 주요 기능들이 모두 이 클래스에서 처리된다.

결과: 게임의 각 단계가 명확하게 정의되어 있으며, 플레이어의 차례를 번갈아 처리하는 로직도 매끄럽게 작동한다. 특히 클래스 간 상호작용이 잘 이루어지며, 두 명의 플레이어 간의 게임이 문제없이 진행된다.

### 2. 게임 로직 구현 평가

2-1. 게임 시작 및 플레이어 입력
두 명의 플레이어 이름을 입력받고 이를 통해 Player 객체를 초기화하는 부분은 매우 간단

하게 구현되었고, 이름을 입력받는 과정에서 오류나 문제는 없었다.

결과: 두 명의 플레이어가 게임에 참여할 수 있도록 초기 설정이 제대로 이루어졌으며, cin을 통해 플레이어의 이름을 입력받는 과정이 매끄럽게 동작한다.

2-2. 랜덤 숫자 생성
srand(time(0)) 을 사용하여 프로그램 실행 시마다 다른 숫자가 나오도록 하였으며, 이를 통해 게임의 랜덤성을 보장했고, rand() % 3 함수를 통해 0~2 사이의 숫자를 무작위로 생성하는 부분이 잘 구현되었다.

결과: 랜덤 숫자 생성 로직은 의도대로 작동하며, 매번 다른 결과가 나와 게임의 흥미를 유발한다. 숫자가 0~2로 제한되었기 때문에 세 개의 숫자가 동일해질 확률도 적절하게 유지되었다.

2-3. 게임 진행 및 차례 전환
while (true)를 사용하여 게임이 지속적으로 진행되며, 승리 조건을 만족하지 못할 경우 다음 플레이어로 차례가 전환되는 과정도 잘 구현되었다. currentPlayer 변수를 통해 각 플레이어의 차례가 번갈아 진행되도록 했으며, 이 과정에서 별다른 오류가 발생하지 않았다.

결과: 플레이어의 차례 전환이 정확하게 이루어지며, 게임 진행 중에도 입력이나 랜덤 숫자 생성에서 문제가 발생하지 않았다.

## 3. 승리 조건 체크 평가

3개의 랜덤 숫자를 비교하여 모두 동일한지 확인하는 부분은 간단한 조건문을 사용하여 구현되었다. 숫자가 모두 동일할 경우 해당 플레이어가 승리하도록 처리하며, 조건을 만족하지 못할 경우 다음 차례로 넘어간다.

결과: 승리 조건이 명확하게 정의되어 있으며, 세 개의 숫자가 동일할 경우 게임이 즉시 종료되고 승리 메시지가 출력된다. 승리 조건 체크 과정에서 오류가 발생하지 않았으며, 조건이 충족되지 않으면 자동으로 차례가 넘어가서 게임이 계속 진행된다.

## 4. 메모리 관리 평가

GamblingGame 클래스의 소멸자를 정의하여 동적으로 할당된 Player 객체의 메모리를 해제하는 부분은 중요하게 처리하였다. 이는 메모리 누수를 방지하고, 프로그램 종료 시 메모리를 적절하게 정리하도록 도와준다.

결과: 소멸자를 통해 메모리 해제가 정확히 이루어졌으며, 프로그램 종료 후에도 메모리 누수 현상이 발생하지 않았다. 동적으로 할당된 객체가 올바르게 해제되어 효율적인 메모리

관리가 이루어진다.

# 4. 문제를 해결한 키 아이디어 또는 알고리즘 설명

이 갬블링 게임에서 핵심적으로 문제를 해결한 아이디어와 알고리즘은 게임 로직의 흐름을 간단하고 명확하게 설계하는 것에 초점을 맞추었고, 각각의 주요 알고리즘 요소는 다음과 같은 방식으로 구현되었다.

## 1. 클래스 기반 설계

게임의 각 기능을 분리하기 위해 객체 지향 프로그래밍의 핵심 개념인 클래스를 사용하였다. Player 클래스와 GamblingGame 클래스는 각각의 역할에 맞추어 잘 나누어졌다.

### 1-1. Player 클래스

각 플레이어의 이름을 저장하고 관리하는 단순한 역할을 수행하였다. 이는 게임의 각 차례에서 플레이어를 식별하고 표시하는 데 중요한 요소였다.

### 1-2. GamblingGame 클래스

게임의 흐름과 로직을 모두 관리하는 중심 역할을 수행하였고, 플레이어의 차례를 관리하고, 숫자 생성 및 비교, 게임의 승리 조건 등을 모두 이 클래스에서 처리했다. 두 클래스 간 상호작용을 통해 게임이 순차적으로 진행되도록 설계되었다.

## 2. 랜덤 숫자 생성 알고리즘

### 2-1. 무작위성 보장

게임의 핵심 요소 중 하나는 각 차례에서 0, 1, 2 중 무작위로 3개의 숫자를 생성하는 부분이다. 이를 위해 rand() 함수와 srand(time(0))을 사용하여 실행할 때마다 새로운 난수 값이 나오도록 설정하였다.

### 2-2. 난수 시드 설정

srand(time(0))을 사용하여 프로그램이 실행될 때마다 난수 생성기의 시드를 설정함으로써, 매번 다른 무작위 숫자가 생성되도록 했다. 이를 통해 게임이 반복적으로 실행될 때도 매번 다른 결과가 나와 무작위성을 유지할 수 있다.

### 2-3. 숫자 범위 제한

rand() % 3을 통해 생성된 난수가 0, 1, 2 범위 내에 있도록 제한했다. 이렇게 작은 범위를 설정함으로써 세 개의 숫자가 동일할 확률을 어느 정도 높여, 게임이 지나치게 길어지지 않도록 설계되었고, 이 범위는 승리 조건을 적절히 달성할 수 있도록 충분히 작으면서도 게임의 흥미를 유지할 수 있도록 도움을 주었다.

## 3. 게임 진행 로직 및 차례 전환 알고리즘

3-1. 무한 루프와 조건 체크
while (true) 루프를 사용하여 게임이 종료될 때까지 계속해서 진행되도록 설계하였다. 게임은 두 명의 플레이어가 번갈아 가면서 숫자를 생성하며, currentPlayer 변수를 통해 현재 차례인 플레이어를 추적하며, 이 변수를 통해 차례가 끝나면 다음 플레이어로 전환합니다.

3-2. 차례 전환 로직
currentPlayer 변수를 번갈아가며 0과 1로 변경함으로써 두 플레이어가 번갈아 차례를 진행하도록 설계되었다. 이를 통해 게임이 자연스럽게 순환되며, 플레이어 간의 차례가 명확하게 구분된다.

## 4. 승리 조건 체크 알고리즘

4-1. 숫자 일치 여부 확인
각 차례에서 생성된 3개의 숫자가 모두 동일한지 확인하는 부분은 간단한 if 조건문을 사용하여 처리되었다.

4-2. 배열 비교
세 개의 숫자를 배열에 저장한 후, 배열의 첫 번째, 두 번째, 세 번째 요소를 비교하여 동일한 경우 승리 메시지를 출력하고 게임을 종료하도록 설계하였다. 이 비교는 단순한 연산이기 때문에 시간 복잡도 측면에서도 효율적이다.

4-3. 게임 종료
승리 조건을 만족하면 break 문을 사용하여 while 루프를 탈출하고, 해당 플레이어가 승리하였음을 알리는 메시지를 출력하고, 게임은 종료된다.

## 5. 메모리 관리

5-1. 동적 메모리 할당 관리
게임의 설계에서 Player 객체는 동적으로 생성되었으므로, 프로그램이 종료될 때 이를 적절하게 해제해야 한다. 이를 위해 GamblingGame 클래스의 소멸자를 정의하여, 프로그램 종료 시 동적으로 할당된 메모리를 모두 해제하여 메모리 누수가 발생하지 않도록 하였다.

5-2. 메모리 해제
동적으로 생성된 객체를 적절하게 삭제하여 메모리 누수를 방지함으로써, 프로그램이 종료될 때까지 시스템 리소스를 낭비하지 않도록 처리하였다.

## 코드 실행 흐름

이 문제의 코드는 두 명의 플레이어가 참가하는 간단한 갬블링 게임을 구현한 프로그램이다. 각 플레이어는 순차적으로 무작위로 생성된 숫자를 확인하고, 세 개의 숫자가 모두 같으면 해당 플레이어가 승리하는 방식으로 작동한다.

## 1. main() 함수 실행

프로그램이 시작되면 `main()` 함수가 실행된다. main() 함수에서 GamblingGame 객체 game이 생성되고, game.play()를 호출하여 게임을 시작한다.

```
#include "Gambling.h"

int main() {
    GamblingGame game;
    game.play();
    return 0;
}
```

## 2. 게임 초기화 (GamblingGame 생성자)

게임 시작 메시지: "***** 갬블링 게임을 시작합니다. *****" 메시지가 출력된다.
두 명의 플레이어 이름을 차례로 입력받아, 각 플레이어의 이름을 Player 클래스 객체로 저장한다.

```
GamblingGame::GamblingGame() {
    string name1, name2;
    cout << "***** 갬블링 게임을 시작합니다. *****" << endl;
    cout << "첫 번째 선수 이름 >> ";
    cin >> name1;
    cout << "두 번째 선수 이름 >> ";
    cin >> name2;
    players[0] = new Player(name1);
    players[1] = new Player(name2);
}
```

## 3. 게임 진행 (play 함수)

```
void GamblingGame::play() {
    srand(time(0));
    int currentPlayer = 0;

    while (true) {
```

```
        Player* player = players[currentPlayer];
        cout << player->name << ":<Enter>" << endl;
        cin.ignore();
        cin.get();
```

srand(time(0));를 사용해 난수를 생성할 준비를 한다. 이로 인해 매번 실행할 때마다 다른 난수가 생성된다. currentPlayer는 0으로 초기화되어 첫 번째 플레이어부터 게임을 시작한다. player->name << ":<Enter>"로 현재 차례인 플레이어의 이름을 출력하고, 플레이어가 <Enter>키를 눌러야 게임이 계속된다.

```
        int numbers[3];
        for (int i = 0; i < 3; i++) {
            numbers[i] = rand() % 3;
        }
```

세 개의 무작위 숫자가 생성된다. numbers[i] = rand() % 3;는 0, 1, 2 중 하나의 숫자를 각 배열 요소에 할당한다.

```
        cout << "\t\t";
        for (int i = 0; i < 3; i++) {
            cout << numbers[i] << "\t";
        }
```

생성된 세 개의 숫자를 화면에 출력한다.

```
        if (numbers[0] == numbers[1] && numbers[1] == numbers[2]) {
            cout << player->name << "님 승리!!" << endl;
            break;
        }
        else {
            cout << "아쉽군요!" << endl;
        }

        currentPlayer = (currentPlayer + 1) % 2;
    }
}
```

세 개의 숫자가 모두 같다면 if (numbers[0] == numbers[1] && numbers[1] == numbers[2]) 에서 참이 되어 해당 플레이어가 승리하고 "님 승리!!" 메시지를 출력한다. 세 개의 숫자가 다르다면 "아쉽군요!"라는 메시지가 출력된다.
currentPlayer = (currentPlayer + 1) % 2;를 통해 플레이어 차례를 바꿉니다. 0에서 1로, 1에서 0으로 반복된다.

## 4. 게임 종료 (소멸자)

```
GamblingGame::~GamblingGame() {
    delete players[0];
    delete players[1];
}
```

게임이 종료되면 소멸자가 호출되어 각 플레이어 객체가 메모리에서 해제된다.

전체 실행 흐름 요약

1. 게임 시작: GamblingGame 객체가 생성되고, 두 명의 플레이어 이름을 입력받습니다.
2. 난수 생성 및 출력: 각 플레이어는 차례대로 엔터를 눌러 세 개의 무작위 숫자를 확인합니다.
3. 승리 여부 판단: 숫자가 모두 같으면 해당 플레이어가 승리하고 게임이 종료됩니다. 그렇지 않으면 차례가 넘어가며 계속해서 게임이 진행됩니다.
4. 게임 종료: 한 명의 승자가 나오면 프로그램이 종료되고, 플레이어 객체가 메모리에서 해제됩니다.