2024 Spring OOP Assignment Report

과제 번호 : 5
학번 : 20230024
이름 : 문요준
Povis ID : yojun313

명예서약 (Honor Code)
나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다.

1. 프로그램 개요

1-1. 개요

Goose Goose Duck 은 온라인 멀티플레이 게임으로, '어몽어스'와 파티 게임 '마피아'를 모티브로 한 사회적 추론 게임이다. 서로의 정체를 알고 있는 오리 진영과 서로의 정체를 모르는 거위 진영 간의 대결을 다루며, 각 진영은 승리 조건을 달성하기 위해 라운드마다 상대 진영의 구성원을 제거해야 한다. 이 프로그램은 해당 온라인 게임을 개인이 플레이할 수 있도록 간소화한 프로그램이다.

1-2. 프로그램 실행 방법 및 사용 방법

[실행 방법]

이 프로그램은 C++로 작성된 게임으로, 사용자는 텍스트 명령을 통해 게임을 조작한다. 프로그램 실행은 컴파일된 실행 파일을 통해 이루어지며, 컴파일 후 생성된 **.exe** 파일을 실행하면 된다. 게임이 시작되면 기본적으로 게임 설정 메뉴가 표시된다.

[사용 방법]

< 게임 설정 메뉴 >

게임 설정 메뉴는 다음과 같은 옵션들로 구성되어 있다:

1. 플레이어 추가 (메뉴 번호: 1):

- 이 옵션을 선택하면 새로운 플레이어를 게임에 추가할 수 있다.
- 플레이어의 이름을 입력하라는 메시지가 나타난다.
- 플레이어의 역할 번호를 입력하라는 메시지가 나타난다.
  - 역할 번호는 다음과 같다:
    1. 송골매 -> 0번
    2. 암살자 오리 -> 1번
    3. 오리 -> 2번
    4. 탐정 거위 -> 3번
    5. 장의사 거위 -> 4번
    6. 거위 -> 5번
    7. 도도새 -> 6번

2. 라운드당 오리 진영 살조 제한 횟수 설정 (메뉴 번호: 2):
   - 이 옵션을 선택하면 오리 진영의 살조 제한 횟수를 설정할 수 있다.
   - 제한 횟수를 입력하라는 메시지가 나타난다.
3. 게임 시작하기 (메뉴 번호: 3):
   - 이 옵션을 선택하면 게임이 시작된다.
   - 게임이 시작되면 각 라운드마다 플레이어들이 능력을 사용하고 투표를 진행하여 상대 진영의 구성원을 제거해 나간다.
   - 각 라운드가 진행된 후, 게임이 종료될 때까지 반복된다.

< 게임 진행 >

게임이 시작되면 다음과 같은 단계로 진행된다:

1. 능력 사용 단계:
   - 각 플레이어는 자신의 역할에 따라 특정 능력을 사용할 수 있다.
   - 능력 사용 여부를 묻는 메시지가 나타나며, 플레이어는 자신의 능력을 사용할지 여부를 결정할 수 있다.
2. 투표 단계:
   - 모든 플레이어가 투표를 통해 제거할 플레이어를 선택한다.
   - 각 플레이어는 투표를 할지 여부를 결정하고, 투표할 플레이어의 이름을 입력한다.
   - 송골매는 자동으로 무효표로 투표된다.
3. 투표 결과 처리:
   - 투표 결과에 따라 가장 많은 투표를 받은 플레이어가 제거된다.
   - 만약 무효표가 가장 많거나, 최다 득표자가 둘 이상인 경우 아무도 제거되지 않는다.

4. 게임 종료 조건 확인:
    - 각 라운드가 종료될 때마다 게임 종료 조건을 확인한다.
    - 특정 조건이 만족되면 게임이 종료되고 승리한 진영이 결정된다.

< 게임 종료 >

게임이 종료되면 최종 결과가 출력된다. 각 진영의 승리 조건에 따라 다음과 같은 메시지가 출력된다:

- 도도새가 승리했을 경우: "도도새의 승리입니다!"
- 거위 진영이 승리했을 경우: "거위의 승리입니다!"
- 오리 진영이 승리했을 경우: "오리의 승리입니다!"
- 송골매가 승리했을 경우: "송골매의 승리입니다!"

1-3. 프로그램 파일 설명

[소스 파일(cpp)]

1. main.cpp
    - 프로그램의 진입점이다.
    - GGD 객체를 생성하고 게임을 시작한다.
2. AssassinDuck.cpp
    - AssassinDuck 클래스의 구현 파일이다.
    - AssassinDuck 클래스의 생성자, 소멸자, 스킬 사용 함수 등을 정의한다.
3. Bird.cpp
    - Bird 클래스의 구현 파일이다.
    - Bird 클래스의 생성자, 소멸자, 멤버 함수들을 정의한다.
4. BirdList.cpp
    - BirdList 클래스의 구현 파일이다.
    - BirdList 클래스의 생성자, 소멸자, 멤버 함수들을 정의한다.
5. BirdNode.cpp
    - BirdNode 클래스의 구현 파일이다.
    - BirdNode 클래스의 생성자와 멤버 함수들을 정의한다.
6. DetectiveGoose.cpp
    - DetectiveGoose 클래스의 구현 파일이다.
    - DetectiveGoose 클래스의 생성자, 소멸자, 스킬 사용 함수 등을 정의한다.

7. DodoBird.cpp
    - DodoBird 클래스의 구현 파일이다.
    - DodoBird 클래스의 생성자, 소멸자, 스킬 사용 함수 등을 정의한다.
8. Duck.cpp
    - Duck 클래스의 구현 파일이다.
    - Duck 클래스의 생성자, 소멸자, 스킬 사용 함수 등을 정의한다.
9. Falcon.cpp
    - Falcon 클래스의 구현 파일이다.
    - Falcon 클래스의 생성자, 소멸자, 스킬 사용 함수 등을 정의한다.
10. GGD.cpp
    - GGD 클래스의 구현 파일이다.
    - GGD 클래스의 생성자, 소멸자, 게임 시작 함수, 라운드 진행 함수, 게임 종료 확인 함수 등을 정의한다.
11. Goose.cpp
    - Goose 클래스의 구현 파일이다.
    - Goose 클래스의 생성자, 소멸자, 스킬 사용 함수 등을 정의한다.
12. MorticianGoose.cpp
    - MorticianGoose 클래스의 구현 파일이다.
    - MorticianGoose 클래스의 생성자, 소멸자, 스킬 사용 함수 등을 정의한다.

[헤더 파일(H)]

1. AssassinDuck.h
    - AssassinDuck 클래스의 선언 파일이다.
    - AssassinDuck 클래스의 멤버 변수와 함수들을 선언한다.
2. Bird.h
    - Bird 클래스의 선언 파일이다.
    - Bird 클래스의 멤버 변수와 함수들을 선언한다.
3. BirdList.h
    - BirdList 클래스의 선언 파일이다.
    - BirdList 클래스의 멤버 변수와 함수들을 선언한다.
4. BirdNode.h
    - BirdNode 클래스의 선언 파일이다.
    - BirdNode 클래스의 멤버 변수와 함수들을 선언한다.
5. DetectiveGoose.h
    - DetectiveGoose 클래스의 선언 파일이다.

- DetectiveGoose 클래스의 멤버 변수와 함수들을 선언한다.

6. DodoBird.h
   - DodoBird 클래스의 선언 파일이다.
   - DodoBird 클래스의 멤버 변수와 함수들을 선언한다.
7. Duck.h
   - Duck 클래스의 선언 파일이다.
   - Duck 클래스의 멤버 변수와 함수들을 선언한다.
8. Falcon.h
   - Falcon 클래스의 선언 파일이다.
   - Falcon 클래스의 멤버 변수와 함수들을 선언한다.
9. GGD.h
   - GGD 클래스의 선언 파일이다.
   - GGD 클래스의 멤버 변수와 함수들을 선언한다.
10. Goose.h
    - Goose 클래스의 선언 파일이다.
    - Goose 클래스의 멤버 변수와 함수들을 선언한다.
11. MorticianGoose.h
    - MorticianGoose 클래스의 선언 파일이다.
    - MorticianGoose 클래스의 멤버 변수와 함수들을 선언한다.
12. Macro.h
    - 여러 상수와 매크로 정의를 포함한 파일이다.
    - 조류 역할 코드와 이름을 정의한다.

2. 프로그램의 구조 및 알고리즘

2-1. Class 설명

1. Class: Bird

개요: Bird 클래스는 모든 조류 역할의 기본 클래스이다. 이 클래스는 각 조류 객체의 공통적인 속성과 기능을 정의하며, 이를 기반으로 다양한 역할을 가진 조류 객체들이 상속받아 구현된다.

멤버 변수 (Member Variables)

1. AttackSomeone (bool)
   - false 로 초기화되는 논리형 변수다.

- 현재 라운드에서 이 조류가 누군가를 공격했는지 여부를 나타낸다.

2. isAlive (bool)
    - true 로 초기화되는 논리형 변수다.
    - 이 조류가 살아있는지 여부를 나타낸다.
3. VoteCount (int)
    - 0 으로 초기화되는 정수형 변수다.
    - 이 조류가 현재 라운드에서 받은 투표 수를 나타낸다.
4. kill_limit (static int)
    - 정적 변수로, 기본값은 1 이다.
    - 모든 Duck 객체가 사용할 수 있는 살인 제한 횟수를 나타낸다.
5. default_kill_limit (static int)
    - 정적 변수로, 기본값은 1 이다.
    - 살인 제한 횟수의 기본값을 나타낸다.
6. player_name (std::string)
    - 플레이어의 이름을 저장하는 문자열 변수다.

생성자 및 소멸자 (Constructor and Destructor)

1. Bird()
    - 기본 생성자로, 멤버 변수들을 초기화한다.
2. virtual ~Bird()
    - 가상 소멸자로, 파생 클래스에서 올바르게 소멸될 수 있도록 한다.

멤버 함수 (Member Functions)

1. static void Set_Kill_limit(int value)
    - kill_limit 변수를 설정하는 정적 함수다.
    - 모든 Duck 객체에 대한 살인 제한 횟수를 설정한다.
2. std::string GetPlayerName() const
    - 플레이어의 이름을 반환하는 함수다.
    - 멤버 변수 player_name 의 값을 반환한다.
3. bool GetisAlive()
    - 이 조류가 살아있는지 여부를 반환하는 함수다.
    - 멤버 변수 isAlive 의 값을 반환한다.
4. bool GetAttackSomeone()
    - 이 조류가 현재 라운드에서 누군가를 공격했는지 여부를 반환하는 함수다.
    - 멤버 변수 AttackSomeone 의 값을 반환한다.
5. void Killed()

- 이 조류를 죽음 상태로 설정하는 함수다.
- 멤버 변수 isAlive 를 false 로 설정한다.

6. void Add_VoteCount()
   - 이 조류가 받은 투표 수를 증가시키는 함수다.
   - 멤버 변수 VoteCount 를 1 증가시킨다.
7. void Reset_VoteCount()
   - 이 조류의 투표 수를 초기화하는 함수다.
   - 멤버 변수 VoteCount 를 0 으로 설정한다.
8. int GetVoteCount()
   - 이 조류가 현재 라운드에서 받은 투표 수를 반환하는 함수다.
   - 멤버 변수 VoteCount 의 값을 반환한다.
9. int GetDefaultKillLimit()
   - 기본 살인 제한 횟수를 반환하는 함수다.
   - 멤버 변수 default_kill_limit 의 값을 반환한다.
10. void Reset_AttackSomeone()
    - 이 조류가 현재 라운드에서 공격한 여부를 초기화하는 함수다.
    - 멤버 변수 AttackSomeone 을 false 로 설정한다.
11. int GetKillLimit()
    - 현재 살인 제한 횟수를 반환하는 함수다.
    - 멤버 변수 kill_limit 의 값을 반환한다.

순수 가상 함수 (Pure Virtual Functions)

1. *virtual void Skill(BirdList birdList) = 0**
   - 순수 가상 함수로, 파생 클래스에서 구현해야 한다.
   - 조류의 특별한 능력을 정의한다.
2. virtual bool AskSkill() = 0
   - 순수 가상 함수로, 파생 클래스에서 구현해야 한다.
   - 조류가 능력을 사용할지 여부를 묻는다.
3. virtual std::string GetJob() = 0
   - 순수 가상 함수로, 파생 클래스에서 구현해야 한다.
   - 조류의 역할을 문자열로 반환한다.

내부에서의 동작 (Internal Operations)

- 생성자와 소멸자:
  - 객체가 생성될 때 기본값으로 멤버 변수를 초기화한다.

    - 객체가 삭제될 때 가상 소멸자가 호출되어 파생 클래스의 소멸자도 올바르게 호출된다.
- Set_Kill_limit:
    - 정적 멤버 변수 kill_limit 의 값을 설정하여 모든 Duck 객체에 대해 전역적으로 적용된다.
- GetPlayerName:
    - 플레이어의 이름을 반환하여, 각 객체가 고유의 플레이어 이름을 갖도록 한다.
- GetisAlive, GetAttackSomeone:
    - 객체의 상태를 확인하는 함수로, 게임 진행 중에 각 조류의 상태를 추적할 수 있다.
- Killed:
    - 객체를 죽음 상태로 설정하여, 게임 진행 중에 해당 조류가 더 이상 활동하지 못하도록 한다.
- Add_VoteCount, Reset_VoteCount, GetVoteCount:
    - 투표 관련 함수를 통해 라운드마다 각 조류가 받은 투표 수를 관리한다.
- GetDefaultKillLimit, GetKillLimit:
    - 살인 제한 횟수를 관리하여, 게임 규칙에 따라 조류가 살인할 수 있는 횟수를 제어한다.
- Reset_AttackSomeone:
    - 라운드가 끝날 때마다 공격 여부를 초기화하여, 새로운 라운드에서 다시 사용할 수 있도록 한다.
- 순수 가상 함수:
    - 파생 클래스에서 반드시 구현해야 하며, 각 조류의 고유한 능력과 역할을 정의하는 데 사용된다.

2. Class: BirdList

개요: BirdList 클래스는 조류 객체들을 연결리스트 형태로 관리하는 클래스다. 각 조류 객체는 BirdNode 객체로 저장되며, 이 리스트를 통해 조류 객체들의 추가, 검색, 투표 처리, 라운드 초기화, 승리 조건 판단 등의 작업을 수행한다.

멤버 변수 (Member Variables)

1. head (BirdNode)*
    - 리스트의 첫 번째 노드를 가리키는 포인터다.
    - 초기값은 nullptr 이다.

2. tail (BirdNode)*
    - 리스트의 마지막 노드를 가리키는 포인터다.
    - 초기값은 nullptr 이다.
3. BirdCount (int)
    - 리스트에 있는 조류 객체의 수를 나타내는 정수형 변수다.
    - 초기값은 0 이다.

생성자 및 소멸자 (Constructor and Destructor)

1. BirdList()
    - 기본 생성자로, head, tail, BirdCount 를 초기화한다.
2. ~BirdList()
    - 소멸자로, 리스트에 있는 모든 노드를 삭제한다.

멤버 함수 (Member Functions)

1. BirdNode *GetHead() const**
    - 리스트의 head 포인터를 반환한다.
2. *BirdNode GetTail() const**
    - 리스트의 tail 포인터를 반환한다.
3. *void AddBirdNode(BirdNode node)**
    - 리스트에 새로운 노드를 추가한다.
    - 리스트가 비어 있으면 head 와 tail 모두를 새로운 노드로 설정한다.
    - 리스트가 비어 있지 않으면, tail 의 next 를 새로운 노드로 설정하고 tail 을 새로운 노드로 갱신한다.
    - BirdCount 를 증가시킨다.
4. *BirdNode FindBird(std::string playername)**
    - 주어진 플레이어 이름을 가진 노드를 찾는다.
    - 리스트의 첫 번째 노드부터 시작하여 각 노드의 player_name 과 비교하며, 일치하는 노드를 찾으면 해당 노드를 반환한다.
    - 일치하는 노드를 찾지 못하면 nullptr 을 반환한다.
5. int GetBirdCount()
    - 리스트에 있는 조류 객체의 수를 반환한다.
6. *Bird ConductVoting(int NeutralVote)**
    - 투표 결과를 처리하여 가장 많은 투표를 받은 조류 객체를 반환한다.
    - 두 번의 순회를 통해 가장 많은 투표 수와 최다 득표자를 확인한다.
    - 최다 득표자가 둘 이상이면 nullptr 을 반환한다.
    - 무효표가 최다 득표와 같으면 최다 득표자 수를 증가시킨다.

7. void RoundReset()
    - 라운드가 끝날 때 각 조류 객체의 투표 수와 공격 여부를 초기화한다.
    - 송골매(Falcon)의 경우, 라운드마다 공격 기회를 초기화한다.
    - 오리(Duck)의 경우, 살인 제한 횟수를 초기화한다.
8. char JudgeWin()
    - 게임의 승리 조건을 판단한다.
    - 리스트의 모든 조류 객체를 순회하며 각 진영의 생존자 수를 계산한다.
    - 거위 진영이 승리 조건을 만족하면 'G'를 반환한다.
    - 오리 진영이 승리 조건을 만족하면 'D'를 반환한다.
    - 송골매가 승리 조건을 만족하면 'F'를 반환한다.
    - 어떤 진영도 승리하지 못하면 'N'을 반환한다.

내부에서의 동작 (Internal Operations)

- 생성자와 소멸자:
    - BirdList 객체가 생성될 때 head, tail, BirdCount 를 초기화한다.
    - 객체가 소멸될 때 리스트에 있는 모든 노드를 삭제하여 메모리 누수를 방지한다.
- GetHead, GetTail:
    - 리스트의 첫 번째 노드와 마지막 노드를 가리키는 포인터를 반환한다.
- AddBirdNode:
    - 새로운 노드를 리스트에 추가하는 함수다.
    - 리스트가 비어 있을 경우 head 와 tail 을 새로운 노드로 설정하고, 비어 있지 않을 경우 tail 의 next 를 새로운 노드로 설정한 후 tail 을 갱신한다.
    - BirdCount 를 증가시켜 리스트의 조류 객체 수를 업데이트한다.
- FindBird:
    - 리스트를 순회하여 주어진 이름의 조류 객체를 찾는다.
    - 일치하는 객체를 찾으면 해당 노드를 반환하고, 찾지 못하면 nullptr 을 반환한다.
- GetBirdCount:
    - 리스트에 있는 조류 객체의 수를 반환한다.
    - 현재 BirdCount 값을 반환한다.
- ConductVoting:
    - 두 번의 순회를 통해 투표 결과를 처리한다.
    - 첫 번째 순회에서는 각 조류 객체의 투표 수를 확인하여 최다 득표 수를 찾는다.
    - 두 번째 순회에서는 최다 득표 수를 가진 조류 객체를 찾고, 최다 득표자가 둘 이상이면 nullptr 을 반환한다.
    - 무효표가 최다 득표 수와 같으면 최다 득표자 수를 증가시킨다.
- RoundReset:

    - 각 라운드가 끝날 때 각 조류 객체의 투표 수와 공격 여부를 초기화한다.
    - 송골매의 경우 라운드마다 공격 기회를 초기화하고, 오리의 경우 살인 제한 횟수를 초기화한다.
- JudgeWin:
    - 리스트의 모든 조류 객체를 순회하여 각 진영의 생존자 수를 계산한다.
    - 각 진영의 승리 조건을 확인하여, 거위 진영이 승리하면 'G', 오리 진영이 승리하면 'D', 송골매가 승리하면 'F'를 반환한다.
    - 어떤 진영도 승리하지 못하면 'N'을 반환한다.

3. Class: BirdNode

개요: BirdNode 클래스는 BirdList 클래스에서 사용되는 노드 클래스다. 각 노드는 하나의 조류 객체(Bird 객체)를 포함하며, 다음 노드를 가리키는 포인터를 가진다. 이 클래스는 연결리스트 구조에서 각 노드의 역할을 수행하며, 리스트의 각 요소를 연결한다.

멤버 변수 (Member Variables)

1. bird (Bird)*
    - 노드가 가리키는 Bird 객체에 대한 포인터다.
    - 각 노드는 하나의 조류 객체를 포함한다.
2. next (BirdNode)*
    - 다음 노드를 가리키는 포인터다.
    - 초기값은 nullptr 이며, 다음 노드가 없는 경우 nullptr 을 유지한다.

생성자 (Constructor)

1. BirdNode(Bird bird)*
    - 주어진 Bird 객체에 대한 포인터로 노드를 초기화하는 생성자다.
    - bird 멤버 변수를 주어진 Bird 객체로 설정하고, next 멤버 변수는 nullptr 로 초기화한다.

멤버 함수 (Member Functions)

1. Bird GetBird() const*
    - 노드가 가리키는 Bird 객체를 반환하는 함수다.
    - 멤버 변수 bird 의 값을 반환한다.
2. BirdNode GetNext() const*
    - 노드가 가리키는 다음 노드를 반환하는 함수다.

    - 멤버 변수 next 의 값을 반환한다.
3. void SetNext(BirdNode next)*
    - 노드가 가리키는 다음 노드를 설정하는 함수다.
    - 멤버 변수 next 를 주어진 BirdNode 포인터로 설정한다.

내부에서의 동작 (Internal Operations)

- *생성자 (BirdNode(Bird bird))**:
    - 주어진 Bird 객체에 대한 포인터로 노드를 초기화한다.
    - bird 멤버 변수를 주어진 Bird 객체로 설정하고, next 멤버 변수는 nullptr 로 초기화한다.
    - 이를 통해 새로운 BirdNode 객체가 생성될 때, 해당 노드는 특정 Bird 객체를 포함하고, 다음 노드를 아직 가리키지 않는 상태가 된다.
- GetBird:
    - 노드가 가리키는 Bird 객체를 반환하는 함수다.
    - 이 함수는 단순히 bird 멤버 변수를 반환한다.
    - 이를 통해 연결리스트의 특정 노드에 저장된 Bird 객체에 접근할 수 있다.
- GetNext:
    - 노드가 가리키는 다음 노드를 반환하는 함수다.
    - 이 함수는 단순히 next 멤버 변수를 반환한다.
    - 이를 통해 연결리스트의 다음 노드로 이동할 수 있다.
- SetNext:
    - 노드가 가리키는 다음 노드를 설정하는 함수다.
    - 이 함수는 next 멤버 변수를 주어진 BirdNode 포인터로 설정한다.
    - 이를 통해 현재 노드의 다음 노드를 지정할 수 있으며, 연결리스트의 구조를 형성하거나 수정할 수 있다.

예시를 통한 설명

예를 들어, 두 개의 BirdNode 객체를 생성하여 연결리스트를 구성한다고 가정하자:

1. 첫 번째 노드 생성:
    - BirdNode* node1 = new BirdNode(new Bird("Player1"))
    - node1 은 "Player1"이라는 이름을 가진 Bird 객체를 포함하는 노드다.
2. 두 번째 노드 생성:
    - BirdNode* node2 = new BirdNode(new Bird("Player2"))
    - node2 는 "Player2"라는 이름을 가진 Bird 객체를 포함하는 노드다.
3. 첫 번째 노드의 다음 노드 설정:
    - node1->SetNext(node2)

- 이제 node1 의 next 포인터는 node2 를 가리킨다.
- 이를 통해 node1 과 node2 가 연결리스트 형태로 연결된다.

이 예제에서 node1 은 BirdList 의 head 가 되고, node2 는 node1 의 다음 노드(next)가 된다. 이를 통해 연결리스트 구조가 형성된다.
BirdNode 클래스는 이러한 연결리스트 구조를 구성하는 기본 단위로, 각 노드는 하나의 조류 객체를 포함하고, 다음 노드를 가리키는 포인터를 통해 리스트를 연결한다.

4. Class: Duck

개요: Duck 클래스는 Bird 클래스를 상속받아 구현된 클래스다. 이 클래스는 오리(Duck) 역할을 수행하며, 특정 능력(살조)을 가지고 있다. Duck 클래스에는 추가적인 멤버 변수와 메서드가 있으며, 이를 통해 오리의 행동을 정의하고 관리한다.

멤버 변수 (Member Variables)

- Duck 클래스는 Bird 클래스를 상속받기 때문에 Bird 클래스의 모든 멤버 변수를 포함한다. 그러나 Duck 클래스 자체에는 추가적인 멤버 변수가 없다.

생성자 및 소멸자 (Constructor and Destructor)

1. Duck()
    - 기본 생성자로, 멤버 변수를 초기화한다.
    - player_name 변수를 빈 문자열로 초기화한다.
2. Duck(std::string player_name)
    - 주어진 플레이어 이름으로 Duck 객체를 초기화한다.
    - player_name 멤버 변수를 주어진 값으로 설정한다.
3. ~Duck()
    - 소멸자로, 특별한 동작은 없고 기본 소멸자를 사용한다.

멤버 함수 (Member Functions)

1. void Skill(BirdList birdList)*
    - 오리의 스킬을 실행하는 함수다.
    - 오리가 다른 조류를 살조하는 동작을 수행한다.
    - 먼저 AskSkill 함수를 호출하여 스킬을 사용할지 여부를 확인한다.
    - 스킬을 사용하기로 결정하면, 살조할 플레이어의 이름을 입력받는다.
    - 입력받은 이름을 통해 조류 리스트에서 해당 조류를 찾아 Killed 메서드를 호출하여 해당 조류를 죽음 상태로 설정한다.

- AttackSomeone 을 true 로 설정하고 kill_limit 을 1 감소시킨다.
- 살조 성공 메시지를 출력한다.

2. bool AskSkill()
   - 오리가 스킬을 사용할지 여부를 묻는 함수다.
   - player_name 과 함께 오리임을 알리는 메시지를 출력한다.
   - kill_limit 이 1 이상인지 확인하여 스킬 사용 여부를 결정한다.
   - 스킬을 사용할 수 있으면 사용자에게 스킬을 사용할지 묻고, 사용 여부에 따라 true 또는 false 를 반환한다.
   - kill_limit 이 1 미만이면 살인 제한 횟수에 도달했음을 알리는 메시지를 출력하고 false 를 반환한다.
3. std::string GetJob()
   - 오리의 역할을 반환하는 함수다.
   - 오리의 역할을 나타내는 문자열 "오리"를 반환한다.

내부에서의 동작 (Internal Operations)

- 생성자 (Duck()):
  - 기본 생성자로, player_name 변수를 빈 문자열로 초기화한다.
- 생성자 (Duck(std::string player_name)):
  - 주어진 플레이어 이름으로 player_name 변수를 초기화한다.
  - 이를 통해 특정 이름을 가진 오리 객체를 생성할 수 있다.
- 소멸자 (~Duck()):
  - 특별한 동작은 없으며, 기본 소멸자를 사용하여 객체가 삭제될 때 기본적으로 할당된 리소스를 해제한다.
- Skill:
  - 이 함수는 오리의 살조 능력을 실행한다.
  - 먼저 AskSkill 함수를 호출하여 스킬을 사용할 수 있는지 여부를 확인한다.
  - 사용자가 스킬 사용에 동의하면 살조할 플레이어의 이름을 입력받는다.
  - 입력받은 이름을 통해 조류 리스트에서 해당 조류를 찾아 Killed 메서드를 호출하여 해당 조류를 죽음 상태로 설정한다.
  - AttackSomeone 을 true 로 설정하고 kill_limit 을 1 감소시킨다.
  - 살조 성공 메시지를 출력한다.
  - 이를 통해 오리는 게임 내에서 다른 조류를 제거할 수 있다.
- AskSkill:
  - 이 함수는 오리가 스킬을 사용할지 여부를 결정한다.
  - player_name 과 함께 오리임을 알리는 메시지를 출력한다.
  - kill_limit 이 1 이상인지 확인하여 스킬 사용 여부를 결정한다.

- 스킬을 사용할 수 있으면 사용자에게 스킬을 사용할지 묻고, 사용 여부에 따라 true 또는 false 를 반환한다.
- kill_limit 이 1 미만이면 살인 제한 횟수에 도달했음을 알리는 메시지를 출력하고 false 를 반환한다.

- GetJob:
  - 이 함수는 오리의 역할을 반환한다.
  - 오리의 역할을 나타내는 문자열 "오리"를 반환한다.
  - 이를 통해 게임 내에서 각 조류 객체의 역할을 명확히 할 수 있다.

5. Class: AssassinDuck

개요: AssassinDuck 클래스는 Duck 클래스를 상속받아 구현된 클래스다. 이 클래스는 암살자 오리 역할을 수행하며, 특정 능력(암살)을 추가로 가지고 있다. AssassinDuck 클래스에는 추가적인 멤버 변수와 메서드가 있으며, 이를 통해 암살자 오리의 행동을 정의하고 관리한다.

멤버 변수 (Member Variables)

- AssassinChance (int)
  - 암살 기회를 나타내는 정수형 변수다.
  - 초기값은 0 이다.
  - 암살 시도 횟수를 관리한다.

생성자 및 소멸자 (Constructor and Destructor)

1. AssassinDuck()
   - 기본 생성자로, 멤버 변수를 초기화한다.
   - AssassinChance 를 0 으로 초기화한다.
2. AssassinDuck(std::string player_name)
   - 주어진 플레이어 이름으로 AssassinDuck 객체를 초기화한다.
   - player_name 멤버 변수를 주어진 값으로 설정하고, AssassinChance 를 0 으로 초기화한다.
3. ~AssassinDuck()
   - 소멸자로, 특별한 동작은 없고 기본 소멸자를 사용한다.

멤버 함수 (Member Functions)

1. void Skill(BirdList birdList)*
   - 암살자 오리의 스킬을 실행하는 함수다.

- 암살자 오리가 다른 조류를 암살하는 동작을 수행한다.
- 먼저 AskSkill 함수를 호출하여 암살을 시도할지 여부를 확인한다.
- 암살을 시도하기로 결정하면, 암살할 플레이어의 이름과 역할 번호를 입력받는다.
- 입력받은 이름과 역할 번호를 통해 조류 리스트에서 해당 조류의 역할을 확인하고, 일치하면 Killed 메서드를 호출하여 해당 조류를 죽음 상태로 설정한다.
- AssassinChance 를 1 증가시키고 AttackSomeone 을 true 로 설정한다.
- 암살 성공 메시지를 출력한다.
- 만약 역할이 일치하지 않으면, 암살자 오리 자신을 죽음 상태로 설정하고 자살 메시지를 출력한다.
- 이후, 남은 살인 기회를 확인하여 추가적인 살인을 수행할 수 있도록 한다.

2. bool AskSkill()
    - 암살자 오리가 암살을 시도할지 여부를 묻는 함수다.
    - player_name 과 함께 암살자 오리임을 알리는 메시지를 출력한다.
    - AssassinChance 가 2 미만인지 확인하여 암살 시도 여부를 결정한다.
    - 암살을 시도할 수 있으면 사용자에게 암살을 시도할지 묻고, 사용 여부에 따라 true 또는 false 를 반환한다.
    - AssassinChance 가 2 이상이면 암살 제한 횟수에 도달했음을 알리는 메시지를 출력하고 false 를 반환한다.
3. std::string GetJob()
    - 암살자 오리의 역할을 반환하는 함수다.
    - 암살자 오리의 역할을 나타내는 문자열 "암살자 오리"를 반환한다.

내부에서의 동작 (Internal Operations)

- 생성자 (AssassinDuck()):
    - 기본 생성자로, AssassinChance 를 0 으로 초기화한다.
    - player_name 변수를 빈 문자열로 초기화한다.
- 생성자 (AssassinDuck(std::string player_name)):
    - 주어진 플레이어 이름으로 player_name 변수를 초기화한다.
    - AssassinChance 를 0 으로 초기화한다.
    - 이를 통해 특정 이름을 가진 암살자 오리 객체를 생성할 수 있다.
- 소멸자 (~AssassinDuck()):
    - 특별한 동작은 없으며, 기본 소멸자를 사용하여 객체가 삭제될 때 기본적으로 할당된 리소스를 해제한다.
- Skill:
    - 이 함수는 암살자 오리의 암살 능력을 실행한다.
    - 먼저 AskSkill 함수를 호출하여 암살을 시도할 수 있는지 여부를 확인한다.

    - 사용자가 암살 시도에 동의하면 암살할 플레이어의 이름과 역할 번호를 입력받는다.
    - 입력받은 이름과 역할 번호를 통해 조류 리스트에서 해당 조류의 역할을 확인하고, 일치하면 Killed 메서드를 호출하여 해당 조류를 죽음 상태로 설정한다.
    - AssassinChance 를 1 증가시키고 AttackSomeone 을 true 로 설정한다.
    - 암살 성공 메시지를 출력한다.
    - 만약 역할이 일치하지 않으면, 암살자 오리 자신을 죽음 상태로 설정하고 자살 메시지를 출력한다.
    - 이후, 남은 살인 기회를 확인하여 추가적인 살인을 수행할 수 있도록 한다.
- AskSkill:
    - 이 함수는 암살자 오리가 암살을 시도할지 여부를 결정한다.
    - player_name 과 함께 암살자 오리임을 알리는 메시지를 출력한다.
    - AssassinChance 가 2 미만인지 확인하여 암살 시도 여부를 결정한다.
    - 암살을 시도할 수 있으면 사용자에게 암살을 시도할지 묻고, 사용 여부에 따라 true 또는 false 를 반환한다.
    - AssassinChance 가 2 이상이면 암살 제한 횟수에 도달했음을 알리는 메시지를 출력하고 false 를 반환한다.
- GetJob:
    - 이 함수는 암살자 오리의 역할을 반환한다.
    - 암살자 오리의 역할을 나타내는 문자열 "암살자 오리"를 반환한다.
    - 이를 통해 게임 내에서 각 조류 객체의 역할을 명확히 할 수 있다.

6. Class: Goose

개요: Goose 클래스는 Bird 클래스를 상속받아 구현된 클래스다. 이 클래스는 거위 역할을 수행하며, 특정 능력이 없는 기본적인 조류 역할을 가진다. Goose 클래스는 Bird 클래스의 멤버 변수와 메서드를 상속받아 사용하며, 추가적인 멤버 변수나 메서드는 없다.

멤버 변수 (Member Variables)

- Goose 클래스는 Bird 클래스를 상속받기 때문에 Bird 클래스의 모든 멤버 변수를 포함한다. 그러나 Goose 클래스 자체에는 추가적인 멤버 변수가 없다.

생성자 및 소멸자 (Constructor and Destructor)

1. Goose()
    - 기본 생성자로, player_name 을 빈 문자열로 초기화한다.

2. Goose(std::string player_name)
    - 주어진 플레이어 이름으로 Goose 객체를 초기화한다.
    - player_name 멤버 변수를 주어진 값으로 설정한다.
3. ~Goose()
    - 소멸자로, 특별한 동작은 없고 기본 소멸자를 사용한다.

멤버 함수 (Member Functions)

1. *void Skill(BirdList birdList)**
    - 거위의 스킬을 실행하는 함수다.
    - 거위는 특별한 능력이 없으므로, 단순히 능력이 없음을 알리는 메시지를 출력한다.
2. bool AskSkill()
    - 거위가 스킬을 사용할지 여부를 묻는 함수다.
    - 거위는 특별한 능력이 없으므로, 항상 true 를 반환한다.
3. std::string GetJob()
    - 거위의 역할을 반환하는 함수다.
    - 거위의 역할을 나타내는 문자열 "거위"를 반환한다.

내부에서의 동작 (Internal Operations)

- 생성자 (Goose()):
    - 기본 생성자로, player_name 변수를 빈 문자열로 초기화한다.
    - 이를 통해 기본 값으로 초기화된 거위 객체를 생성할 수 있다.
- 생성자 (Goose(std::string player_name)):
    - 주어진 플레이어 이름으로 player_name 변수를 초기화한다.
    - 이를 통해 특정 이름을 가진 거위 객체를 생성할 수 있다.
- 소멸자 (~Goose()):
    - 특별한 동작은 없으며, 기본 소멸자를 사용하여 객체가 삭제될 때 기본적으로 할당된 리소스를 해제한다.
- Skill:
    - 이 함수는 거위의 스킬을 실행한다.
    - 거위는 특별한 능력이 없으므로, 단순히 능력이 없음을 알리는 메시지를 출력한다.
    - 이를 통해 거위는 특별한 행동을 수행하지 않음을 명확히 한다.
- AskSkill:

    - 이 함수는 거위가 스킬을 사용할지 여부를 결정한다.
    - 거위는 특별한 능력이 없으므로, 항상 true 를 반환한다.
    - 이를 통해 거위는 특별한 조건 없이 게임에 참여할 수 있다.
- GetJob:
    - 이 함수는 거위의 역할을 반환한다.
    - 거위의 역할을 나타내는 문자열 "거위"를 반환한다.
    - 이를 통해 게임 내에서 각 조류 객체의 역할을 명확히 할 수 있다.

7. Class: DetectiveGoose

개요: DetectiveGoose 클래스는 Goose 클래스를 상속받아 구현된 클래스다. 이 클래스는 탐정 거위 역할을 수행하며, 특정 능력(조사)을 가지고 있다. DetectiveGoose 클래스에는 추가적인 멤버 변수는 없지만, 추가적인 메서드가 있으며, 이를 통해 탐정 거위의 행동을 정의하고 관리한다.

멤버 변수 (Member Variables)

- DetectiveGoose 클래스는 Goose 클래스를 상속받기 때문에 Goose 클래스와 Bird 클래스의 모든 멤버 변수를 포함한다. 그러나 DetectiveGoose 클래스 자체에는 추가적인 멤버 변수가 없다.

생성자 및 소멸자 (Constructor and Destructor)

1. DetectiveGoose()
    - 기본 생성자로, player_name 을 빈 문자열로 초기화한다.
2. DetectiveGoose(std::string player_name)
    - 주어진 플레이어 이름으로 DetectiveGoose 객체를 초기화한다.
    - player_name 멤버 변수를 주어진 값으로 설정한다.
3. ~DetectiveGoose()
    - 소멸자로, 특별한 동작은 없고 기본 소멸자를 사용한다.

멤버 함수 (Member Functions)

1. void Skill(BirdList birdList)*
    - 탐정 거위의 스킬을 실행하는 함수다.
    - 탐정 거위가 다른 플레이어가 현재 라운드에서 공격을 했는지 여부를 조사한다.
    - 먼저 AskSkill 함수를 호출하여 스킬을 사용할지 여부를 확인한다.
    - 스킬을 사용하기로 결정하면, 조사할 플레이어의 이름을 입력받는다.

- 입력받은 이름을 통해 조류 리스트에서 해당 조류를 찾아 GetAttackSomeone 메서드를 호출하여 해당 조류가 현재 라운드에서 공격을 했는지 여부를 확인한다.
- 공격했으면 공격했다는 메시지를, 공격하지 않았으면 공격하지 않았다는 메시지를 출력한다.

2. bool AskSkill()
   - 탐정 거위가 스킬을 사용할지 여부를 묻는 함수다.
   - player_name 과 함께 탐정 거위임을 알리는 메시지를 출력한다.
   - 사용자에게 스킬을 사용할지 묻고, 사용 여부에 따라 true 또는 false 를 반환한다.
3. std::string GetJob()
   - 탐정 거위의 역할을 반환하는 함수다.
   - 탐정 거위의 역할을 나타내는 문자열 "탐정 거위"를 반환한다.

내부에서의 동작 (Internal Operations)

- 생성자 (DetectiveGoose()):
  - 기본 생성자로, player_name 변수를 빈 문자열로 초기화한다.
  - 이를 통해 기본 값으로 초기화된 탐정 거위 객체를 생성할 수 있다.
- 생성자 (DetectiveGoose(std::string player_name)):
  - 주어진 플레이어 이름으로 player_name 변수를 초기화한다.
  - 이를 통해 특정 이름을 가진 탐정 거위 객체를 생성할 수 있다.
- 소멸자 (~DetectiveGoose()):
  - 특별한 동작은 없으며, 기본 소멸자를 사용하여 객체가 삭제될 때 기본적으로 할당된 리소스를 해제한다.
- Skill:
  - 이 함수는 탐정 거위의 스킬을 실행한다.
  - 먼저 AskSkill 함수를 호출하여 스킬을 사용할 수 있는지 여부를 확인한다.
  - 사용자가 스킬 사용에 동의하면 조사할 플레이어의 이름을 입력받는다.
  - 입력받은 이름을 통해 조류 리스트에서 해당 조류를 찾아 GetAttackSomeone 메서드를 호출하여 해당 조류가 현재 라운드에서 공격을 했는지 여부를 확인한다.
  - 공격했으면 공격했다는 메시지를, 공격하지 않았으면 공격하지 않았다는 메시지를 출력한다.
  - 이를 통해 탐정 거위는 다른 플레이어의 행동을 조사할 수 있다.
- AskSkill:
  - 이 함수는 탐정 거위가 스킬을 사용할지 여부를 결정한다.
  - player_name 과 함께 탐정 거위임을 알리는 메시지를 출력한다.
  - 사용자에게 스킬을 사용할지 묻고, 사용 여부에 따라 true 또는 false 를 반환한다.
  - 이를 통해 탐정 거위는 조사 능력을 사용할지 여부를 결정할 수 있다.

- GetJob:
    - 이 함수는 탐정 거위의 역할을 반환한다.
    - 탐정 거위의 역할을 나타내는 문자열 "탐정 거위"를 반환한다.
    - 이를 통해 게임 내에서 각 조류 객체의 역할을 명확히 할 수 있다.

8. Class: MorticianGoose

개요: MorticianGoose 클래스는 Goose 클래스를 상속받아 구현된 클래스다. 이 클래스는 장의사 거위 역할을 수행하며, 특정 능력(죽은 플레이어의 역할 조사)을 가지고 있다. MorticianGoose 클래스에는 추가적인 멤버 변수는 없지만, 추가적인 메서드가 있으며, 이를 통해 장의사 거위의 행동을 정의하고 관리한다.

멤버 변수 (Member Variables)

- MorticianGoose 클래스는 Goose 클래스를 상속받기 때문에 Goose 클래스와 Bird 클래스의 모든 멤버 변수를 포함한다. 그러나 MorticianGoose 클래스 자체에는 추가적인 멤버 변수가 없다.

생성자 및 소멸자 (Constructor and Destructor)

1. MorticianGoose()
    - 기본 생성자로, player_name 을 빈 문자열로 초기화한다.
2. MorticianGoose(std::string player_name)
    - 주어진 플레이어 이름으로 MorticianGoose 객체를 초기화한다.
    - player_name 멤버 변수를 주어진 값으로 설정한다.
3. ~MorticianGoose()
    - 소멸자로, 특별한 동작은 없고 기본 소멸자를 사용한다.

멤버 함수 (Member Functions)

1. void Skill(BirdList birdList)*
    - 장의사 거위의 스킬을 실행하는 함수다.
    - 장의사 거위가 죽은 플레이어의 역할을 조사한다.
    - 먼저 AskSkill 함수를 호출하여 스킬을 사용할지 여부를 확인한다.
    - 스킬을 사용하기로 결정하면, 먼저 리스트에서 죽은 플레이어가 있는지 확인한다.
    - 죽은 플레이어가 없으면, 염습 가능 대상이 없음을 알리는 메시지를 출력한다.
    - 죽은 플레이어가 있으면, 염습하고 싶은 플레이어의 이름을 입력받는다.

    - 입력받은 이름을 통해 조류 리스트에서 해당 조류를 찾아 GetJob 메서드를 호출하여 해당 조류의 역할을 확인하고, 이를 출력한다.
2. bool AskSkill()
    - 장의사 거위가 스킬을 사용할지 여부를 묻는 함수다.
    - player_name 과 함께 장의사 거위임을 알리는 메시지를 출력한다.
    - 사용자에게 스킬을 사용할지 묻고, 사용 여부에 따라 true 또는 false 를 반환한다.
3. std::string GetJob()
    - 장의사 거위의 역할을 반환하는 함수다.
    - 장의사 거위의 역할을 나타내는 문자열 "장의사 거위"를 반환한다.

내부에서의 동작 (Internal Operations)

- 생성자 (MorticianGoose()):
    - 기본 생성자로, player_name 변수를 빈 문자열로 초기화한다.
    - 이를 통해 기본 값으로 초기화된 장의사 거위 객체를 생성할 수 있다.
- 생성자 (MorticianGoose(std::string player_name)):
    - 주어진 플레이어 이름으로 player_name 변수를 초기화한다.
    - 이를 통해 특정 이름을 가진 장의사 거위 객체를 생성할 수 있다.
- 소멸자 (~MorticianGoose()):
    - 특별한 동작은 없으며, 기본 소멸자를 사용하여 객체가 삭제될 때 기본적으로 할당된 리소스를 해제한다.
- Skill:
    - 이 함수는 장의사 거위의 스킬을 실행한다.
    - 먼저 AskSkill 함수를 호출하여 스킬을 사용할 수 있는지 여부를 확인한다.
    - 사용자가 스킬 사용에 동의하면 먼저 리스트에서 죽은 플레이어가 있는지 확인한다.
    - 죽은 플레이어가 없으면 염습 가능 대상이 없음을 알리는 메시지를 출력한다.
    - 죽은 플레이어가 있으면 염습하고 싶은 플레이어의 이름을 입력받는다.
    - 입력받은 이름을 통해 조류 리스트에서 해당 조류를 찾아 GetJob 메서드를 호출하여 해당 조류의 역할을 확인하고, 이를 출력한다.
    - 이를 통해 장의사 거위는 죽은 플레이어의 역할을 조사할 수 있다.
- AskSkill:
    - 이 함수는 장의사 거위가 스킬을 사용할지 여부를 결정한다.
    - player_name 과 함께 장의사 거위임을 알리는 메시지를 출력한다.
    - 사용자에게 스킬을 사용할지 묻고, 사용 여부에 따라 true 또는 false 를 반환한다.
    - 이를 통해 장의사 거위는 염습 능력을 사용할지 여부를 결정할 수 있다.
- GetJob:

- 이 함수는 장의사 거위의 역할을 반환한다.
- 장의사 거위의 역할을 나타내는 문자열 "장의사 거위"를 반환한다.
- 이를 통해 게임 내에서 각 조류 객체의 역할을 명확히 할 수 있다.

9. Class: Falcon

개요: Falcon 클래스는 Bird 클래스를 상속받아 구현된 클래스다. 이 클래스는 송골매 역할을 수행하며, 특정 능력(살해)을 가지고 있다. Falcon 클래스에는 추가적인 멤버 변수가 있으며, 이를 통해 송골매의 행동을 정의하고 관리한다.

멤버 변수 (Member Variables)

- KillChance_PerRound (int)
  - 라운드마다 살해할 수 있는 기회를 나타내는 정수형 변수다.
  - 기본값은 1 로 설정되어 있으며, 라운드가 끝날 때마다 초기화된다.

생성자 및 소멸자 (Constructor and Destructor)

1. Falcon()
   - 기본 생성자로, KillChance_PerRound 를 1 로 초기화한다.
   - player_name 변수를 빈 문자열로 초기화한다.
2. Falcon(std::string player_name)
   - 주어진 플레이어 이름으로 Falcon 객체를 초기화한다.
   - player_name 멤버 변수를 주어진 값으로 설정하고, KillChance_PerRound 를 1 로 초기화한다.
3. ~Falcon()
   - 소멸자로, 특별한 동작은 없고 기본 소멸자를 사용한다.

멤버 함수 (Member Functions)

1. *void Skill(BirdList birdList)**
   - 송골매의 스킬을 실행하는 함수다.
   - 송골매가 다른 조류를 살해하는 동작을 수행한다.
   - 먼저 AskSkill 함수를 호출하여 스킬을 사용할지 여부를 확인한다.
   - 스킬을 사용하기로 결정하면, 살해할 플레이어의 이름을 입력받는다.
   - 입력받은 이름을 통해 조류 리스트에서 해당 조류를 찾아 Killed 메서드를 호출하여 해당 조류를 죽음 상태로 설정한다.
   - AttackSomeone 을 true 로 설정하고 KillChance_PerRound 을 1 감소시킨다.

- 살해 성공 메시지를 출력한다.

2. bool AskSkill()
    - 송골매가 스킬을 사용할지 여부를 묻는 함수다.
    - player_name 과 함께 송골매임을 알리는 메시지를 출력한다.
    - KillChance_PerRound 이 1 이상인지 확인하여 스킬 사용 여부를 결정한다.
    - 스킬을 사용할 수 있으면 사용자에게 스킬을 사용할지 묻고, 사용 여부에 따라 true 또는 false 를 반환한다.
    - KillChance_PerRound 이 1 미만이면 살해 제한 횟수에 도달했음을 알리는 메시지를 출력하고 false 를 반환한다.
3. std::string GetJob()
    - 송골매의 역할을 반환하는 함수다.
    - 송골매의 역할을 나타내는 문자열 "송골매"를 반환한다.
4. void Reset_KillChance_PerRound()
    - 라운드가 끝날 때 송골매의 살해 기회를 초기화하는 함수다.
    - KillChance_PerRound 를 1 로 설정한다.

내부에서의 동작 (Internal Operations)

- 생성자 (Falcon()):
    - 기본 생성자로, KillChance_PerRound 를 1 로 초기화한다.
    - player_name 변수를 빈 문자열로 초기화한다.
    - 이를 통해 기본 값으로 초기화된 송골매 객체를 생성할 수 있다.
- 생성자 (Falcon(std::string player_name)):
    - 주어진 플레이어 이름으로 player_name 변수를 초기화한다.
    - KillChance_PerRound 를 1 로 초기화한다.
    - 이를 통해 특정 이름을 가진 송골매 객체를 생성할 수 있다.
- 소멸자 (~Falcon()):
    - 특별한 동작은 없으며, 기본 소멸자를 사용하여 객체가 삭제될 때 기본적으로 할당된 리소스를 해제한다.
- Skill:
    - 이 함수는 송골매의 살해 능력을 실행한다.
    - 먼저 AskSkill 함수를 호출하여 스킬을 사용할 수 있는지 여부를 확인한다.
    - 사용자가 스킬 사용에 동의하면 살해할 플레이어의 이름을 입력받는다.
    - 입력받은 이름을 통해 조류 리스트에서 해당 조류를 찾아 Killed 메서드를 호출하여 해당 조류를 죽음 상태로 설정한다.
    - AttackSomeone 을 true 로 설정하고 KillChance_PerRound 을 1 감소시킨다.
    - 살해 성공 메시지를 출력한다.

    - 이를 통해 송골매는 게임 내에서 다른 조류를 제거할 수 있다.
- AskSkill:
    - 이 함수는 송골매가 스킬을 사용할지 여부를 결정한다.
    - player_name 과 함께 송골매임을 알리는 메시지를 출력한다.
    - KillChance_PerRound 이 1 이상인지 확인하여 스킬 사용 여부를 결정한다.
    - 스킬을 사용할 수 있으면 사용자에게 스킬을 사용할지 묻고, 사용 여부에 따라 true 또는 false 를 반환한다.
    - KillChance_PerRound 이 1 미만이면 살해 제한 횟수에 도달했음을 알리는 메시지를 출력하고 false 를 반환한다.
    - 이를 통해 송골매는 살해 능력을 사용할지 여부를 결정할 수 있다.
- GetJob:
    - 이 함수는 송골매의 역할을 반환한다.
    - 송골매의 역할을 나타내는 문자열 "송골매"를 반환한다.
    - 이를 통해 게임 내에서 각 조류 객체의 역할을 명확히 할 수 있다.
- Reset_KillChance_PerRound:
    - 이 함수는 라운드가 끝날 때 송골매의 살해 기회를 초기화한다.
    - KillChance_PerRound 를 1 로 설정하여 새로운 라운드에서 다시 사용할 수 있도록 한다.

10. Class: DodoBird

개요: DodoBird 클래스는 Bird 클래스를 상속받아 구현된 클래스다. 이 클래스는 도도새 역할을 수행하며, 게임에서 도도새의 특별한 승리 조건을 가지고 있다. DodoBird 클래스에는 추가적인 멤버 변수는 없지만, 추가적인 메서드가 있으며, 이를 통해 도도새의 행동을 정의하고 관리한다.

멤버 변수 (Member Variables)

- DodoBird 클래스는 Bird 클래스를 상속받기 때문에 Bird 클래스의 모든 멤버 변수를 포함한다. 그러나 DodoBird 클래스 자체에는 추가적인 멤버 변수가 없다.

생성자 및 소멸자 (Constructor and Destructor)

1. DodoBird()
    - 기본 생성자로, player_name 을 빈 문자열로 초기화한다.
2. DodoBird(std::string player_name)
    - 주어진 플레이어 이름으로 DodoBird 객체를 초기화한다.
    - player_name 멤버 변수를 주어진 값으로 설정한다.

3. ~DodoBird()
    - 소멸자로, 특별한 동작은 없고 기본 소멸자를 사용한다.

멤버 함수 (Member Functions)

1. void Skill(BirdList birdList)*
    - 도도새의 스킬을 실행하는 함수다.
    - 도도새는 특별한 능력이 없으므로, 단순히 능력이 없음을 알리는 메시지를 출력한다.
2. bool AskSkill()
    - 도도새가 스킬을 사용할지 여부를 묻는 함수다.
    - 도도새는 특별한 능력이 없으므로, 항상 true 를 반환한다.
3. std::string GetJob()
    - 도도새의 역할을 반환하는 함수다.
    - 도도새의 역할을 나타내는 문자열 "도도새"를 반환한다.

내부에서의 동작 (Internal Operations)

- 생성자 (DodoBird()):
    - 기본 생성자로, player_name 변수를 빈 문자열로 초기화한다.
    - 이를 통해 기본 값으로 초기화된 도도새 객체를 생성할 수 있다.
- 생성자 (DodoBird(std::string player_name)):
    - 주어진 플레이어 이름으로 player_name 변수를 초기화한다.
    - 이를 통해 특정 이름을 가진 도도새 객체를 생성할 수 있다.
- 소멸자 (~DodoBird()):
    - 특별한 동작은 없으며, 기본 소멸자를 사용하여 객체가 삭제될 때 기본적으로 할당된 리소스를 해제한다.
- Skill:
    - 이 함수는 도도새의 스킬을 실행한다.
    - 도도새는 특별한 능력이 없으므로, 단순히 능력이 없음을 알리는 메시지를 출력한다.
    - 이를 통해 도도새는 특별한 행동을 수행하지 않음을 명확히 한다.
- AskSkill:
    - 이 함수는 도도새가 스킬을 사용할지 여부를 결정한다.
    - 도도새는 특별한 능력이 없으므로, 항상 true 를 반환한다.
    - 이를 통해 도도새는 특별한 조건 없이 게임에 참여할 수 있다.
- GetJob:
    - 이 함수는 도도새의 역할을 반환한다.

- 도도새의 역할을 나타내는 문자열 "도도새"를 반환한다.
- 이를 통해 게임 내에서 각 조류 객체의 역할을 명확히 할 수 있다.

11. Class: GGD

개요: GGD 클래스는 게임의 전체 흐름을 관리하는 클래스다. 이 클래스는 게임의 설정, 라운드 진행, 게임 종료 확인 등을 수행하며, 플레이어를 추가하고 게임의 승리 조건을 판단하는 기능을 포함한다. GGD 클래스에는 멤버 변수와 메서드가 있으며, 이를 통해 게임의 전체 로직을 관리하고 실행한다.

멤버 변수 (Member Variables)

1. bird_list (BirdList)*
    - 게임에 참여하는 조류 객체들을 관리하는 연결 리스트다.
    - BirdList 클래스의 포인터다.
2. dodoWin (bool)
    - 도도새가 승리했는지 여부를 나타내는 논리형 변수다.
    - 초기값은 false 다.

생성자 및 소멸자 (Constructor and Destructor)

1. GGD()
    - 기본 생성자로, bird_list 를 초기화하고, dodoWin 을 false 로 설정한다.
2. ~GGD()
    - 소멸자로, 특별한 동작은 없고 기본 소멸자를 사용한다.

멤버 함수 (Member Functions)

1. void GameStart()
    - 게임 설정 메뉴를 표시하고, 플레이어 추가, 오리 진영의 살인 제한 횟수 설정, 게임 시작 등을 처리하는 함수다.
    - 게임 시작 전에 사용자가 게임 설정을 완료할 수 있도록 한다.
2. void RoundProgress()
    - 라운드 진행을 관리하는 함수다.
    - 각 라운드에서 조류 객체의 스킬 사용, 투표, 투표 결과 처리 등을 수행한다.
    - 라운드가 종료될 때마다 게임 종료 조건을 확인하고, 게임이 끝나면 결과를 출력한다.
3. bool IsGameOver()

- 게임 종료 조건을 확인하는 함수다.
- bird_list 의 JudgeWin 메서드를 호출하여 게임이 종료되었는지 확인하고, dodoWin 변수를 확인하여 도도새의 승리 여부를 판단한다.
- 게임이 종료되면 true, 그렇지 않으면 false 를 반환한다.

4. void PrintGameResult()
   - 게임 결과를 출력하는 함수다.
   - bird_list 의 JudgeWin 메서드를 호출하여 승리한 진영을 확인하고, dodoWin 변수를 확인하여 도도새의 승리 여부를 판단한다.
   - 각 승리 조건에 따라 적절한 메시지를 출력한다.
5. void AddPlayer()
   - 게임에 플레이어를 추가하는 함수다.
   - 사용자가 입력한 이름과 역할 번호에 따라 적절한 조류 객체를 생성하고, 이를 bird_list 에 추가한다.

내부에서의 동작 (Internal Operations)

- 생성자 (GGD()):
  - 기본 생성자로, bird_list 를 초기화하고, dodoWin 을 false 로 설정한다.
  - bird_list 는 새로운 BirdList 객체로 초기화된다.
- 소멸자 (~GGD()):
  - 특별한 동작은 없으며, 기본 소멸자를 사용하여 객체가 삭제될 때 기본적으로 할당된 리소스를 해제한다.
- GameStart:
  - 이 함수는 게임 설정 메뉴를 표시하고, 사용자가 게임을 설정할 수 있도록 한다.
  - 게임 설정 메뉴에서 플레이어 추가, 오리 진영의 살인 제한 횟수 설정, 게임 시작 등을 처리한다.
  - 게임 시작이 선택되면 게임을 시작한다.
- RoundProgress:
  - 이 함수는 게임의 각 라운드를 진행한다.
  - 각 라운드에서 조류 객체의 스킬 사용을 처리하고, 사용자가 투표를 수행할 수 있도록 한다.
  - 투표 결과를 처리하여 최다 득표자를 결정하고, 해당 조류 객체를 죽음 상태로 설정한다.
  - 라운드가 종료될 때마다 게임 종료 조건을 확인하고, 게임이 끝나면 결과를 출력한다.
- IsGameOver:
  - 이 함수는 게임이 종료되었는지 여부를 확인한다.

- bird_list 의 JudgeWin 메서드를 호출하여 게임의 승리 조건을 확인하고, dodoWin 변수를 확인하여 도도새의 승리 여부를 판단한다.
- 게임이 종료되면 true, 그렇지 않으면 false 를 반환한다.

- PrintGameResult:
  - 이 함수는 게임 결과를 출력한다.
  - bird_list 의 JudgeWin 메서드를 호출하여 승리한 진영을 확인하고, dodoWin 변수를 확인하여 도도새의 승리 여부를 판단한다.
  - 각 승리 조건에 따라 적절한 메시지를 출력한다.
- AddPlayer:
  - 이 함수는 게임에 플레이어를 추가한다.
  - 사용자가 입력한 이름과 역할 번호에 따라 적절한 조류 객체를 생성하고, 이를 bird_list 에 추가한다.
  - 조류 객체는 역할 번호에 따라 생성되며, 생성된 객체는 bird_list 에 추가된다.

2-2. 전체적 알고리즘 설명 - main().cpp

개요: main() 함수는 GGD 객체를 생성하고 게임을 설정하고, 라운드를 진행하며, 게임 종료 조건을 확인하는 역할을 한다. GGD 클래스는 게임의 전체 흐름을 관리하며, 플레이어 추가, 라운드 진행, 게임 종료 확인 등의 주요 기능을 제공한다. 프로그램은 게임 설정, 라운드 진행, 게임 종료 확인의 단계를 거쳐 게임을 진행하며, 각 단계에서 사용자 입력과 게임 로직을 처리한다.

1. **게임 설정**
   - **GGD** 객체 생성
   - **GameStart()** 호출하여 게임 설정 메뉴 표시
   - 플레이어 추가, 오리 진영의 살인 제한 횟수 설정
   - 게임 시작 선택
2. **라운드 진행**
   - **RoundProgress()** 호출하여 각 라운드를 진행
   - 각 조류 객체의 스킬 사용
   - 투표 진행 및 결과 처리
   - 라운드 종료 시 게임 종료 조건 확인
3. **게임 종료 확인 및 결과 출력**
   - **IsGameOver()** 호출하여 게임 종료 여부 확인
   - 게임이 종료되면 **PrintGameResult()** 호출하여 결과 출력

3. 토론 및 개선

3-1. 배우거나 깨달은 내용

1. **상속과 다형성:**
    - Bird 클래스를 상속받아 Goose, Duck, Falcon, DodoBird, DetectiveGoose, MorticianGoose, AssassinDuck 등의 클래스를 정의함으로써 상속과 다형성의 개념을 이해할 수 있었다.
    - 순수 가상 함수(추상 메서드)를 사용하여 각 파생 클래스에서 구현해야 하는 메서드를 정의하고, 이를 통해 다형성을 구현하는 방법을 배울 수 있었다.
2. **캡슐화:**
    - 각 클래스의 멤버 변수를 private 또는 protected 로 선언하고, 이를 접근하기 위한 public 메서드를 제공함으로써 캡슐화의 중요성을 배울 수 있었다.
    - 객체 내부의 상태를 보호하고, 외부에서는 객체의 메서드를 통해서만 상태를 변경하도록 설계하는 방법을 이해했다.

3-2. 개선 방향

1. **AI 플레이어 추가:**
    - 사용자가 혼자서도 게임을 즐길 수 있도록 AI 플레이어를 추가한다.
    - AI 의 행동을 설계하고, 이를 통해 더 다양한 게임 플레이를 제공한다.