표지

프로젝트명:마피아 게임
팀명:[다 해줬잖아]
팀원:[김민준 5881222], [박준우 5460378], [박지현 5881344], [한지원 5881620]
제출일:2025년 6월 8일

목차

1. 프로젝트 개요

1.1 주제 및 목적

- **마피아 게임** 콘솔 버전 구현
- 주요 메커니즘(역할 배정, 밤·낮 사이클, 투표 시스템, 기억상실자) 재현을 통한 OOP 학습

1.2 마피아 게임 개요 및 개발 동기

- 마피아 게임은 파티·심리 전략 게임으로, 마피아와 시민 간 추리·토론이 핵심
- 이러한 복합 로직을 코드로 재현하며 설계·디버깅 능력 강화 목표

1.3 목표

1. 역할별 행동(마피아, 시민, 의사, 탐정, 기억상실자) 완전 구현
2. 낮 투표 시스템: 투표 집계, 동표 처리, 결과 발표
3. 승패 판정: 마피아·시민 동수 및 마피아 전멸 조건 정확 검출
4. 확장성 확보: 새로운 역할·규칙 추가 용이 구조 설계

2. 요구사항 분석

2.1 기능적 요구사항

- 플레이어 역할 무작위 배정
- 밤낮 페이즈 진행 및 각 역할별 행동
- 낮 투표 시스템(수집·집계·동표 처리)
- 승패 판정 및 게임 종료
- ASCII 아트 기반 UI 출력

2.2 비기능적 요구사항

- 언어: C++17
- 플랫폼: 콘솔 애플리케이션
- 설계 원칙: SOLID, 객체지향, RAII

- 품질: 유지보수성, 확장성, 메모리 안정성

3. 시스템 설계

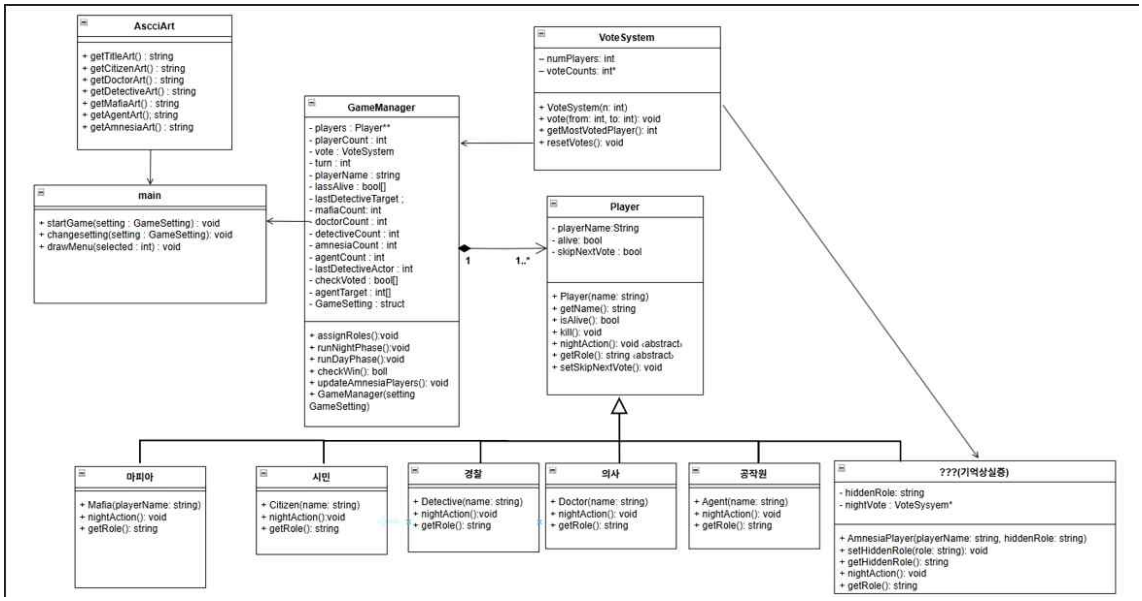
3.1 개발 스케줄

단계	시작일	종료일
프로젝트 구상	03/20	04/01
프로젝트 상세 설계	04/01	04/15
코딩	04/15	05/15
융합 & 디버깅	05/01	06/01
최종 마무리	06/01	Finale

3.2 전체 구조(아키텍처)

main.cpp → GameManager → Role 클래스들 → VoteSystem → 결과 출력

시스템 설계(그대로 복붙하시면 됩니다.)



1. Main 화면

Class: main		
변수/함수명	역할	비고
startGame(setting : GameSetting) : void	게임을 시작하는 함수	게임매니저에서 세팅 정보 참조
changesetting(setting : GameSetting): void	세팅을 변경하는 함수	참조한세팅정보를 변경하는 함수
drawMenu(selected : int) : void	아스키코드를 가져오는 함수	아스키클래스 참조

2. AscciArt

Class: AscciArt		
변수/함수명	역할	비고
getTitleArt() : string	메인 UI를 그린다.	

getTitleArt() : string	메인 UI를 그린다.	
getCitizenArt() : string	시민 그림을 그린다.	
getDoctorArt() : string	의사 그림을 그린다.	
getDetectiveArt() : string	경찰 그림을 그린다.	
getMafiaArt() : string	마피아 그림을 그린다.	
getAgentArt(); string	공작원 그림을 그린다.	
getAmnesiaArt() : string	기억상실증 그림을 그린다.	

3. GameManager

Class: GameManager		
변수/함수명	역할	비고
players : Player*	플레이어 정보를 저장한다.	
playerCount : int	플레이어 수를 저장한다.	
vote : VoteSystem	투표시스템을 불러와 투표한다.	votesystem
turn : int	현재 턴을 센다.	
playerName : string	플레이어 이름을 저장한다.	
lastAlive : bool[]	밤을 지난 뒤 생존 했는지 확인한다.	경찰, 공작원이 죽었을때 다음날 아침 능력이 출력되는 것을 방지한다.
lastDetectiveTarget int ;	밤에 조사할 플레이어를 저장한다.	경찰의 능력
mafiaCount: int	마피아 수를 저장한다.	
doctorCount : int	의사 수를 저장한다.	

detectiveCount : int	경찰 수를 저장한다.	
amnesiaCount : int	기억상실증 수를 저장한다.	
agentCount : int	공작원 수를 저장한다.	
lastDetectiveActor : int	경찰의 조사정보를 저장한다.	
checkVoted : bool[]	투표를 했는지 여부를 저장한다.	
agentTarget : int[]	공작원이 조사할 사람의 번호를 저장한다.	
GameSetting : struct	게임 인원수를 저장한다. 세팅시 기억상실증 기능을 작동시킬 수 있다.	defalt로 플레이어7명, 마피아2명, 각직업 2명이 되도록 저장되어있다.
assignRoles():void	플레이어의 역할을 부여한다.	
runNightPhase():void	밤을 실행한다.	
runDayPhase():void	낮을 실행한다.	
checkWin(): boll	승리팀을 판별한다.	
updateAmnesiaPlayers(): void	기억상실플레이어의 실제역할을 섞는다.	
GameManager(setting GameSetting)	생성자를 통해서 플레이어의 정보를 입력받는다.	

4. VoteSystem

Class: Vote System		
변수/함수명	역할	비고
numPlayers: int	플레이어수를 저장한다.	
voteCounts: int*	투표 받은 플레이어마다 1표를 추가한다.	

VoteSystem(n: int)	.투표 시스템 생성자 플레이어 수를 입력받는다.	
vote(from: int, to: int): void	누가 누구에게 투표를 했는지 저장한다.	
getMostVotedPlayer(): int	가장 많은 득표를한 플레이어 번호를 반환한다.	
resetVotes(): void	투표함을 초기화한다.	

5. Player

Class: Player		
변수/함수명	역할	비고
playerName:String	플레이어의 이름을 저장한다.	
alive: bool	생존 여부를 저장한다.	
skipNextVote : bool	투표 스킵여부를 저장한다.	
Player(name: string)	플레이어 생성자로 플레이어 정보를 저장한다.	
getName(): string	플레이어이름을 반환하는 함수	
isAlive(): bool	자신이 살았는지 체크한다.	
kill(): void	사람을 죽이는 기능이다.	
nightAction(): void ‹abstract›	밤 행동을 출력하는 역할의 인터페이스	
getRole(): string ‹abstract›	자신의 직업을 출력하는 역할의 인터페이스	
setSkipNextVote(): void	플레이어 정보를 수정해 투표를 스킵하는 상태로만든다.	공작원의 능력

6. 마피아, 시민, 경찰, 의사, 공작원 공통

Class: 마피아, 시민, 경찰, 의사, 공작원 공통		
변수/함수명	역할	비고
생성자(name: string)	이름을 저장한다.	
nightAction(): void	직업별 밤에 행동할 대상을 지정한다.	시민은 투표를 진행하지만 아무런 영향을 끼치지 않는다.
getRole(): string	자신의 직업을 반환한다.	Amnesia 플레이어는 ???라는 직업명을 반환 받는다.

7. Amnesia

Class: Amnesia(???)		
변수/함수명	역할	비고
hiddenRole: string	숨겨진 실제역할을 저장한다.	
nightVote : VoteSysyem*	숨겨진 직업의 투표를 진행한다.	시민은 투표를 진행하지만 아무런 영향을 끼치지 않는다.
AmnesiaPlayer(playerName: string, hiddenRole: string)	.투표 시스템 생성자 플레이어 수를 입력받는다.	
setHiddenRole(role: string): void	누가 누구에게 투표를 했는지 저장한다.	
getHiddenRole(): string	숨겨진 실제직업을 출력한다.	개발자 디버깅 출력용도
nightAction(): void	숨겨진 실제직업에 맞는 행동을 한다.	
getRole(): string	직업을 반환한다.	Amnesia는 ???를 반환한다.

개발하면서 아쉬운 점

첫번째로 현재 기억상실증 클래스의 nightAction()이 각 시민 직업을 상속받지 않고, 따로 기억상실증 직업 처럼 구현되어있다. 문제점은 기억상실증이 “다른 직업의 역할”을 밤에 실행한다는 컨셉과는 맞지 않는 상속 구현이다. 이렇게 코드가 짜여진 이유는 각 직업의 능력을 직업 클래스에 구현한 것이 아닌, “GameManager 클래스”에서 플레이어별 직업을 인식하고, 그 직업의 능력을 사용하도록 구현해 주었기 때문이다.

이 클래스다이어그램은 SOLID원칙 중 단일 책임의 원칙을 GameManaeger가 위반하며, 추가로 개방 폐쇄의 원칙도 위반해 코드의 확장이 어려워 유지보수가 어려움을 시사한다. 실제로 중간발표 이후 공작원이라는 직업을 추가하려할 때 처음 경찰, 의사를 구현할 때 보다 많은 시간이 걸리기도 하였다.

만약, 시간이 좀 더 있는 프로젝트 였다면, GameManager 클래스에서 구현된 직업행동들을 nightAction들로 옮기는 방향으로 개발하여 SOLID원칙을 지키며 유지보수성을 높이는 개발을 하였을 것이라는 아쉬움이 남는다.

4. 구현

4.1 주요 클래스/함수 설명

GameManager

- initializePlayers(): 역할별 Player 객체 생성 및 리스트 저장
- runGameLoop(): 밤·낮 페이즈 순환 제어
- nightPhase(): 각 Player::nightAction()호출
- dayPhase(): VoteSystem통한 투표 수집 및 집계
- resolveVotes(): 최다 득표자 제거 로직
- checkWinCondition(): 승패 판정

VoteSystem

- **생성자:**VoteSystem(int n)— 플레이어 수 설정, voteCounts초기화
- resetVotes(): 집계 초기화
- collectVotes(const vector<Player*>&): 플레이어별 vote(int from,int to)호출
- vote(int from,int to): 유효성 검사 후 집계 증가
- getMostVotedPlayer(): 최다 득표자 인덱스 반환(동표 시 무작위 선택)

Role 클래스

- **Player (추상):**virtual void nightAction(vector<Player*>&)
- **Agent:** nightAction()에서 지목한 agentTarget() 호출
- **Mafia:**동료 협의 후 희생자 선택
- **Doctor:**nightAction()에서 치료 대상 선택 및 heal()호출
- **Detective:**nightAction()에서 조사 대상 선택 및 investigate()호출
- **AmnesiaPlayer:**nightAction()에서 recoverMemory()수행

AsciiArt

- drawTitleArt(), getCitizenArt(), getDoctorArt(), getDetectiveArt(), getMafiaArt(), getAgentArt(), getAmnesiaArt() : 게임 UI용 ASCII 아트 출력

4.2 특이사항 및 도입 알고리즘

- **역할 배정:**Fisher-Yates 셔플 (std::shuffle)

4.3 코드 스냅샷 및 설명

```
// VoteSystem::getMostVotedPlayer()
int maxVotes = *max_element(voteCounts.begin(), voteCounts.end());
vector<int> ties;
for(int i=0;i<numPlayers;i++) if(voteCounts[i]==maxVotes) ties.push_back(i);
return ties.size()>1 ? ties[rand()%ties.size()] : ties[0];
```

- max_element로 최댓값 탐색 후 동표자 처리 로직 구현

5. 테스트 및 결과

5.1 테스트 방법

- **직접 플레이: 34**회 반복, 다양한 역할 조합 시나리오
- **유닛 테스트:**VoteSystem, checkWinCondition()등 핵심 메서드 검증

5.2 테스트 결과

- **정상 진행:**98% 성공
- **동표 처리:**100% 정확
- **역할 누락:**0건 발견

5.3 버그 및 해결 방법

버그	원인	수정 내용
치료 중복 처리	의사와 마피아 대상 동일 시 치료 우선 실패	if (healTarget==killTarget) skip kill적용
기억상실자 적용 누락	recoverMemory()호출 위치 오류	호출 위치를 nightPhase()상단으로 이동

6. 회고 및 개선점

6.1 느낀 점/배운 점

- 명확한 클래스 책임 분리의 중요성 체감

6.2 개선 아이디어

7. 결론

- 본 프로젝트를 통해 마피아 게임의 핵심 로직을 C++로 완전 구현
- 모듈화된 설계로 확장성·유지보수성 확보 확인
- 향후 GUI·멀티플레이 기능 추가 예정

8. 참고자료

- C++17 표준 문서
- StackOverflow: Fisher-Yates shuffle 구현 가이드
- GitHub 프로젝트 리포지터리 링크

9. 부록

- 전체 소스코드: GitHub 링크
- UML 다이어그램 원본 파일
- 테스트 로그 및 스크린샷