애플리케이션 기획

윈도우 프로그래밍 – 2019182002 경태환

게임명 : 1945 모작

맵 구성 설명 : PPT 추가 예정

* High-Level

(클라이언트)------------------------------------------------------------------------------------

통신 관련 쓰레드 : 서버와의 TCP/IP 통신을 담당, 각 Tick 마다 필요한 데이터 송수신

렌더링 쓰레드 : 서버에서 수신한 데이터를 바탕으로 렌더링하는 쓰레드

싱글톤 매니저 : 각 레벨별로 다양한 싱글톤 매니저를 통해 클라이언트 로직 담당

(서버)---------------------------------------------------------------------------------------------

1. 클라이언트 수신 쓰레드 2개 : 각각의 클라이언트에 대해 TCP 소켓 관리

* 각각 받은 데이터를 뮤텍스로 보호하고 있는 별도의 큐 or 버퍼에 저장한 뒤 이를 동기화 쓰레드가 접근할 수 있도록 함 (데이터 수신 쓰레드는 데이터 수신 작업에만 집중하도록 구성)

2. 게임 상태 동기화 쓰레드 1개 : 모든 클라이언트에 대해 주기적으로 게임 상태를 동기화

* 주기적으로 Tick 루프문을 통해 통신 쓰레드에서 수집한 데이터(QUEUE에 저장되어있음) 를 GameState 객체에 반영

3. 충돌 처리 쓰레드 1개

* 주기적으로 Tick 루프문 내에서 Queue에 저장되어있는 데이터를 통해 충돌처리 여부를 판단하고 게임 State 객체에 반영함

4. 클라이언트 전송 쓰레드 1개

* 업데이트 된 게임 상태를 전송 큐에서 읽어 클라이언트로 전송(LateTick)

클라이언트 – 서버 프로그램 로직 예시

1. 클라이언트에서 받은 입력은 클라이언트의 입력 Queue에 저장(여러 프레임에 걸쳐서 입력이 들어갈 수 있음)
2. 입력을 받아둔 Queue의 내부 정보를 순차적으로 서버에 전송
3. 서버는 클라이언트 1,2 에서 정보를 받아서 Mutex로 보호되고 있는 각각의 클라이언트에 맞는 Queue에 저장(Queue는 2개 존재) (과도한 Blocking으로 프레임 드랍이 과하게 발생한다면 각각 따로 Queue를 관리하거나 다른 알고리즘 이용 예정)
4. 서버의 동기화 쓰레드는 Queue에서 데이터를 읽어와 GameState.Tick() 루프 내에서 1차로 동기화
5. 서버의 충돌 처리 쓰레드는 Queue에서 데이터를 읽어와 충돌 발생 여부를 확인하고 충돌 후 상황까지 처리하여 GameState에 반영함(Tick)
6. 최종적으로 데이터 송신을 위해 LateTick에서 클라이언트 전송 쓰레드가 처리된 데이터를 클라이언트로 송신함
7. 클라이언트의 Tick에서 받은 데이터로 객체 상태 바꿔주고 렌더 준비

* Low-Level

1. 애플리케이션 프로토콜
   1. 기본 패킷 구조
      1. 클라이언트 -> 서버 패킷(InputPacket)
         1. 패킷 유형 (enum 타입으로 Packet 종류 지정(우선 인풋 데이터 1개)
         2. 패킷 길이
         3. 실제로 입력한 데이터(Client의 InputData 구조체)
         * InputData 구조체 :
           + int playerId; // 플레이어 ID
           + float moveX; // X축 이동
           + float moveY; // Y축 이동
           + bool shoot; // 발사 여부
      2. 서버 -> 클라이언트 패킷 (GameStatePacket)
         1. 패킷 유형 (ex)Game\_State\_UPDATE)
         2. 패킷 길의
         3. 실제 데이터(GameState구조체)
         * GameState 구조체 :
           + struct GameState
           + { std::vector<PlayerState> players; // 플레이어 상태
           + std::vector<MonsterState> monsters; // 몬스터 상태
           + std::vector<PlayerBulletState> PlayerBullets; // 플레이어 총알
           + std::vector<MonsterBulletState> PlayerBullets; // 몬스터 총알
           + int score; // 현재 점수 };
2. 함수 및 기능(추가 예정)

서버)---------------------------------------------------------------------------------------

* + 1. void InitializeServer(int port) -> 소켓 생성 및 바인딩 코드
    2. GameState.Tick() -> 서버의 Tick에서 입력 데이터로 각 플레이어의 상태정보 업데이트
       1. ReciveClientData(int clientId) -> 서버에서 클라이언트로부터 전송받을 패킷을 수신해 각각 맞는 Queue에 저장
       2. Update\_Object() -> 각 객체들을 업데이트 함
       3. Check\_Collision() -> 충돌 처리를 담당하는 쓰레드로 충돌처리
    3. GameState.LateTick() -> 서버의 LateTick() 주로 처리된 데이터를 클라이언트로 송신함
       1. SendGameStateToClients() -> 클라이언트로 데이터 송신

위에 것들은 대충 구조 이미지 만들면서 바꿀 예정 => 서버도 클라이언트와 같은 레벨로 관리하고 레벨이 변경될 때 그에 맞는 소켓으로 변경 할 예정

Ex) 클라, 서버에서 메뉴 레벨일 경우 플레이어 선택에 관한 패킷을 보낼 것이고 선택 완료 정보가 넘어가면 둘다 게임 플레이로 넘어감. 이후에는 위에 소켓으로 바꾸어 데이터를 보낼 예정

클라이언트)-------------------------------------------------------------------------------

1. MainGame.Tick()
   1. 여기에 이제 구조적으로 추가해줘 -> 우선 결론적으로 Tick에서 데이터 송 수신이 이루어질꺼야. 서버에서 충돌처리까지 체크해서 넘어오니까 LateTick에서 받아온 플레이어, 몬스터, 기타등등에 대한 정보를 반영하는 구조로 가고싶어

================================

팀원 별 역할분담

개발환경

개발일정