**네트워크 게임 프로그래밍 추진계획서**

2018182005 김도성

2018182018 오세현

2018182017 송주석

1. 애플리케이션 기획

2018182018 오세현 컴퓨터 그래픽스 텀 프로젝트 제출물 (장애물 피하기)

[ 게임 진행 ]

3D 객체가 맵을 진행하면서 다가오는 객체들과의 충돌을 피하며, 점수쌓는 게임만들기. 시간이 지날수록 피하기 어려워지며, 난이도가 올라간다. 난이도는 총 4단계로 구성되어있다. 1단계에서는 파란색 객체, 2단계에서는 노란색, 3단계에서는 주황색, 4단계에서는 빨간색 객체가 점점 빨라지는 속도로 다가온다. 이 객체들을 피해서 최대한 오래 살아남고 점수를 높이 쌓는 것이 이 게임의 목표이다. 네트워크게임프로그래밍 강의에서는 이 게임을 활용해, 두 명의 플레이어가 경쟁하는 게임으로 진행된다.

[ 기존 게임 구조 소개 ]

- 각 객체 생성 / 렌더링

- 객체를 움직이는 타이머

- 객체의 충돌

- 플레이어 키입력

[ 추가 사항 ]

- 점수 랭킹 표시

- 멀티플레잉 기능

- 적 플레이어 렌더링

1. High-Level Design

텍스트, 도표, 기술 도면, 평면도이(가) 표시된 사진

자동 생성된 설명

1. Low-Level Design

패킷 구조체 정의 {

POSXYZ{ float posX, float posY, float posZ }

CSKeyPacket{ BYTE type, int keytype },

CSInitialPacket{ BYTE type, char nameInitial[NAMELEN] },

CSRestartPacket{ BYTE type, bool start },

SCCharacterPacket{BYTE type, bool isCollide, POSXYZ characterXYZ },

SCObstaclePacket{BYTE type, POSXYZ obstacleXYZ}

SCResumePacket{ BYTE type, bool winner },

SCWinnerPacket{ BYTE type, POSXYZ obstacleXYZ },

SCRankedInfo{ int meter, char name[STRLEN] },

SCRankinPacket{ BYTE type, RankedInfo rankings[RANKERS] }

}

Client {

void CreateSphere(GLuint&EBO, GLuint& VBO);

void Jump\_n\_Hide\_Sound();

void playingBgm();

void draw\_land();

void draw\_cube(posXYZ pos);

void draw\_normal\_cube(posXYZ pos);

void draw\_hard\_cube(posXYZ pos);

void draw\_hard\_cube2(posXYZ pos);

void draw\_universe();

void draw\_miniMap();

void draw\_sphere(posXYZ pos);

void InitSocket(); // 소켓 생성

bool IsMultiPlayer(); // 멀티플레이 확인

* 플레이어가 1명 or 2명 인지

void SendKeyEvent(int key); // 키값 서버에 보내기

* 키값 받아서 서버로 send

void ClientProcess(); // recv()후 패킷 처리하는 과정

* 클라이언트의 수신 버퍼를 peeking하여 패킷 타입을 확인 후,
* 서버로부터 장애물들의 xyz 값 recv후 위치 설정
* 타입에 따른 recv패킷의 종류 확인후 recv 처리

void setPlayerPos(posxyz pos); // 플레이어 위치 설정

* 플레이어xyz값을 recv하여 플레이어 위치 설정

void setObstaclePos(posxyz pos); // 장애물 위치 설정

* 장애물xyz값을 recv하여 장애물 위치 설정

void setEnemyPos(posxyz pos); // 적 위치 설정

* 적플레이어Xyz,값을 recv하여 적 플레이어 위치 설정

void Gameover(bool winner); // 승자확인 및 게임종료

* 누가 이겼는지 패킷을 recv, 승자확인 후 게임 종료

Server {

void InitGamePlayer(); //게임 실행시작

void InitPlayer(); //캐릭터 위치 생성

* 플레이어 초기 위치값 설정

void InitObstacle(); //장애물 위치 생성

* 장애물 초기 위치값 설정

void moveCharater(int keytype, char\* ip); // 캐릭터 움직임변화

* 어떤 keytype인지 파악후 플레이어 위치값 변경,
* Ip를 통해 맞는 플레이어에게 변경된 위치값 send

void moveObstacle(); // 장애물 움직임 변화

* 장애물 위치값 변경,
* 각 플레이어에게 변경된 위치값 send

Void ServerProcess(); // 서버 동작

* recevieProcess, moveObstacle, moveCharacter, isDead, setRankedinfo 등의 과정들을 반복처리하는 함수

void ReceiveProcess(SOCKET socekt); // recv()후 패킷 처리하는 과정

* 수신 버퍼의 peeking을 통해 타입을 확인 하고,
* 타입에 따라 recv패킷의 종류 확인 후 해당 패킷에 맞는 recv처리

void setRankInfo(int meter, char\* nameinitial); // 순위권 확인 및 기록

* rankInfoPacket에 담아 클라이언트로 send

bool isDead(); //캐릭터 사망판정

* 일정 충돌 횟수 -> true 리턴
* overGame() 처리

void Restart(); // 게임 재실행 확인

* resumePacket 클라이언트로 send

void OverGame(); // 게임 종료 및 처리

* 사망 판정 시
* winnerPacket 클라이언트로 send

void game\_over\_timer(int value); // 게임오버 로고의 움직임 구현

void cube\_move\_timer(int value); // 시간경과에 따른 큐브(장애물)의 이동 구현

void sphere\_jump\_timer(int value); // 시간경과에 따른 캐릭터 점프 구현

void sphere\_hide\_timer(int value); // 시간경과에 따른 캐릭터 은신 구현

bool handle\_collide(int i); // 큐브에 맞는 충돌판정 함수

bool normal\_handle\_collide(int i);

bool hard\_handle\_collide(int i);

bool hard2\_handle\_collide(int i);

}

DWORD WINAPI MoveThread(LPVOID arg, SOCKET socket)

DWORD WINAPI ClientThread(LPVOID arg)

// 서버의 클라이언트 스레드 동작

DWORD WINAPI MoveThread (LPVOID arg, SOCKET socket){

각 cube\_move\_timer 함수를 이용,

->장애물 위치변경

Sphere\_move\_timer 함수 이용

->캐릭터 위치변경(캐릭터가 sphere)

어느 플레이어의 캐릭터인지 알기 위해 socket을 인자로 받아 구분

}

DWORD WINAPI ClientThread(LPVOID arg){

movethread에서 움직인 큐브의 좌표값과 캐릭터의 충돌판정 계산

충돌판정 및 위치값 클라이언트로 send

죽었을때의 승자 패킷 send

}

// 서버의 메인 스레드 동작

MainThread{

소켓 생성 및 초기화, 연결, 스레드 생성

}

// moveThread , clientThread 동기화 문제,

// 한쪽 플레이어가 일방적으로 종료됬을 경우의 문제

// 문서 구체화

1. 팀원 별 역할 분담

오세현: draw함수 구조 변경, 서버 싱글 플레잉 기능 구현

김도성: collide함수 구조 변경, 서버 멀티 플레잉 통신기능 구현

송주석: timer함수 구조 변경, 클라이언트 통신기능구현

1. 개발 환경

Visual Studio 2022

OpenGL

Github

TCP/IP Protocol

1. 개발 일정

김도성

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 일 | 월 | 화 | 수 | 목 | 금 | 토 |
|  |  |  | 11.8  handle\_collide() | 11.9  normal\_handle\_collide() | 11.10  hard\_handle\_collide() | 11.11  hard2\_handle\_collide() |
| 11.12  시험준비 | 11.13  시험준비 | 11.14 | 11.15  충돌검사 오류 수정 및 보완 | 11.16  충돌검사 오류 수정 및 보완 | 11.17  ServerProcess() | 11.18  팀원 코드 병합 및 오류 검사 |
| 11.19  팀원 코드 병합 및 오류검사 | 11.20  ServerProcess() + Restart() | 11.21  moveObstacle() | 11.22  moveObstacle() | 11.23  Overgame() | 11.24  현 코드까지 오류 수정 및 보완 | 11.25  팀원 코드 병합 및 오류 검사 |
| 11.26  팀원 코드 병합 및 오류 검사 | 11.27  기존 코드 보완 및 수정 | 11.28  팀원 코드 병합 및 최종 검사 | 11.29 | 11.30  ~~~ | 12.1  추가사항 검토 후 기능 추가 | 12.2  ~~~ |

오세현

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 일 | 월 | 화 | 수 | 목 | 금 | 토 |
|  |  |  | 11.8  draw\_land() | 11.9  draw\_universe() | 11.10  draw\_board() | 11.11  draw\_sphere() |
| 11.12  시험 준비 | 11.13  시험 준비 | 11.14  draw\_cube() | 11.15  draw\_normal\_cube() | 11.16  draw\_hard\_cube() | 11.17  draw\_hard\_cube2() | 11.18 팀원 코드 병합 및 오류 검사 |
| 11.19  팀원코드 병합 및 오류 검사 | 11.20  InitSocket() + InitPlayer() | 11.21  InitObstacle() | 11.22 InitGamePlayer() | 11.23  SetRankedInfo()+ IsDead() | 11.24  ReceiveProcess() | 11.25  팀원 코드 병합 및 오류 검사 |
| 11.26  팀원코드 병합 및 오류 검사 | 11.27 기존 코드 보완 및 수정 | 11.28  팀원 코드 병합 및 제출 | 11.29 | 11.30  ~~~ | 12.1  추가사항 검토 후 기능 추가 | 12.2  ~~~ |

송주석

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 일 | 월 | 화 | 수 | 목 | 금 | 토 |
|  |  |  | 11.8 game\_over\_timer() 수정 | 11.9  cube\_move\_timer() 수정 | 11.10  sphere\_jump\_timer()수정 | 11.11  sphere\_hide\_timer()수정 |
| 11.12  시험준비 | 11.13  시험준비 | 11.14  InitSocket() 구현 | 11.15  IsMultiPlayer() 구현 | 11.16  SendKeyEvent() 구 | 11.17  ClientProcess() 구현 | 11.18  팀원 코드 병합 및 오류 검사 |
| 11.19  팀원 코드 병합 및 오류 검사 | 11.20  SetPlayerPos() 구현 | 11.21  SetObstaclePos() 구현 | 11.22  SetEnemyPos() 구현 | 11.23  Gameover() 구현 | 11.24  기존 코드 보완 및 수정 | 11.25  팀원 코드 병합 및 오류 검사 |
| 11.26  팀원 코드 병합 및 오류 검사 | 11.27  기존 코드 보완 및 수정 | 11.28  팀원 코드 병합 및 최종 검사 | 11.29  추가사항 검토 후 기능 추가 | 11.30  추가사항 검토 후 기능 추가 | 12.1  추가사항 검토 후 기능 추가 | 12.2  추가사항 검토 후 기능 추가 |