게임 요소

1. 장르 : 슈팅게임
2. 컨셉 : 1 vs 1
3. 플레이어 수 : 2인
4. 개발자 : 서동우, 3D게임프로그래밍 과제로 제작.

게임 소개

* 규칙 : 두 플레이어는 각자의 탱크를 조종하여 상대의 탱크를 파괴해야 함, 포탄에 피격당할 때마다 체력이 감소하며, 체력이 0이 된 플레이어는 패배하게 됨.
* 조작 : WASD(이동), 마우스(화면, 포탑 회전), SPACE(발사)

하이 레벨 디자인

스크린샷, 텍스트, 그래픽, 도표이(가) 표시된 사진

자동 생성된 설명

로우 레벨 디자인

Object\_ID

// Server to Client

constexpr int SC\_CREATE\_OBJECT = 0

constexpr int SC\_MOVE\_OBJECT = 1

constexpr int SC\_DELETE\_OBJECT = 2

constexpr int SC\_COLLISION\_OBJECT = 3

constexpr int SC\_PLAYER\_ROLE =4

//Client to Server

constexpr int CS\_PLAYER\_MOVE = 0

Packet ID

Key ID

constexpr int KEY\_UP = 0

constexpr int KEY\_DOWN =1

constexpr int KEY\_LEFT =2

constexpr int KEY\_RIGHT =3

constexpr int KEY\_SHOT = 4

Player Role

constexpr int TURRET = 0

constexpr int BODY = 1

constexpr int PLAYER = 0

constexpr int ENEMY = 1

constexpr int BULLET = 2

constexpr int MISSILE = 3

constexpr int TREE = 4

protocol.h에 정의된 상수

(서버와 클라이언트가 공유)

struct sc\_parent\_packet{

int packet\_type;

}

protocol.h에 정의된 패킷 구조체. list로 관리하기 위해 상속을 통해 구현함.

(서버와 클라이언트가 공유)

struct sc\_delete\_object\_packet : public sc\_parent\_packet {

int object\_id;

}

struct sc\_collision\_object\_packet : public sc\_parent\_packet {

int object\_id1;

int object\_id2;

}

struct cs\_player\_move\_packet : public sc\_parent\_packet {

int input\_event;

}

struct sc\_player\_role\_packet : public sc\_parent\_packet {

int role;

}

struct sc\_create\_object\_packet : public sc\_parent\_packet {

int object\_type;

XMVECTOR pos;

}

struct sc\_move\_object\_packet : public sc\_parent\_packet {

int object\_id;

XMVECTOR pos;

}

서버에서 InputEventQueue에 저장된 Event를 처리하는 함수.

서버의 전역 변수

WorkerThread와 RecvThread를 제어하기 위한 이벤트, 클라이언트 둘이 모두 접속을 완료하면 신호 상태로 바뀌어 쓰레드가 작업을 시작하도록 한다.

서버의 전역 변수

접속한 클라이언트의 소켓을 저장하고 관리하는 배열.

클라이언트에서 수신 받은 데이터를 EVENT 형식으로 queue에 삽입한다.

서버의 전역 변수

InputEventQueue 내부에 클라이언트에서 받은 인풋을 EVENT 형식으로 저장한다.

WorkerThread와 두 클라이언트의 RecvThread 모두에게 사용되기 때문에 임계영역을 통해 동기화한다.

void HandleInputEvent(queue<EVENT> q)

{

while(!q.empty()) {

EVENT event = q.pop();

switch(event) {

case KEY\_UP :

break;

case KEY\_DOWN :

break;

.

.

}

}

}

HANDLE stop;

stop = CreateEvent(NULL, false,false);

arry<2, SOCKET> client\_socket;

queue<EVENT> InputEventQueue;

typedef struct EVENT {

int event\_id;

int clinet\_id;

} EVENT;

Worker Thread

DWORD WINAPI WorkerThread()

{

WaitForSingleObject(stop);

while(true) {

// 임계 영역 진입

HandleInputEvent(InputEventQueue);

// 임계 영역 탈출

step1. Transform Matrix Update

step2. Collider Transform Update

step3. Collision check

step4. Collision handling

send(client\_socket[0], (char\*)packet\_num,sizeof(int), 0);

send(client\_socket[1], (char\*)packet\_num,sizeof(int), 0);

for(int i = 0 ; i < packet\_num ; i++)

send(client\_socket[0],(char\*)BUF,sizeof(),0);

send(client\_socket[1],(char\*)BUF,sizeof(),0);

}

}

WorkerThread 게임에서 필요한 행렬 변환이나 충돌 처리를 하며 두 클라이언트에게 계산 결과를 전송한다.

클라이언트 소켓은 array로 관리

worker thread에서 전송될 패킷들은 list를 통해서 관리한다.

각 클라이언트에서 recv 받는 쓰레드 함수 이때 클라이언트 구별을 위해 client\_id를 인자로 받았다.

InPutEventQueue 동기화를 위해서 임계영역을 사용. (Recv Thread, HandleInputEvent에 사용)

DWORD WINAPI RecvThread(int client\_id)

{

WaitForSingleObject(stop);

while(true) {

char buf;

int retval = recv(client\_socket[client\_id],buf,BUFSIZE,MSG\_WAITALL);

if (retval == 0) return 0;

// 임계 영역 진입

cs\_player\_move\_packet packet = reinterpret\_cast<cs\_player\_move\_packet>(buf);

EVENT ev;

ev.event\_id = ev.event\_id = packet.input\_evnet;

ev.client\_id = client\_id;

InputEventQueue.insert(&ev);

// 임계 영역 탈출

}

}

Recv Thread

클라이언트

SendHandleInptuEvent( Input ) {

If( 무슨 키가 눌렸는지 || 마우스 관련 이벤트가 있는지 )

send(소켓, (char\*) cs\_player\_move\_packet , sizeof(int), 0);

}

클라이언트에서 수신 받은 패킷을 프로토콜에 따라 각 오브젝트에 적용하는 함수.

RecvPacketType(packetType) {

switch(packetType){

case Create:

RecvCreateObject()

case Delete:

RecvDeleteObject()

case Update:

RecvUpdateTransform()

}

}

RecvUpdateTransform(sc\_move\_object\_packet ) {

if( id에 해당하는 오브젝트가 있다면) {

objMgr.UpdateTransform(sc\_move\_object\_packet );

}

}

RecvDeleteObject(sc\_delete\_object\_packet ) {

if(id에 해당하는 오브젝트가 있다면)

objMgr.Delete(sc\_delete\_object\_packet );

}

RecvCreateObject(sc\_create\_object\_packet ) {

if(id에 해당하는 오브젝트가 없다면)

objMgr.CreateObject(sc\_create\_object\_packet );

}

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 11/7(화) | 11/8(수) | 11/9(목) | 11/10(금) | 11/11(토) | 11/12(일) | 11/13(월) |
|  | 프로토콜 완성 | 임시 클라이언트 생성 후 간단한 send recv 테스트 |  | worker thread 내부에서 eventqueue에서 이벤트를 꺼내는 기능 구현 (동기화 고려) | 1차 진행사항 체크 및 피드백 | 겜소공 시험 준비 |
| 11/14 | 11/15 | 11/16 | 11/17 | 11/18 | 11/19 | 11/20 |
| 데베 시험 준비 | 데베 시험 준비 | HandleInput  Event() 함수에서 queue 사용 동기화 구현 | 클라이언트에 소켓 추가 및 현재까지 개발된 한도 내에서 send recv 구현 | 클라이언트에 서버에서 전송한 이벤트 처리 결과  수신 받는 기능 추가 | 2차 진행사항 체크 및 피드백 | 졸작 회의 |
| 11/21 | 11/22 | 11/23 | 11/24 | 11/25 | 11/26 | 11/27 |
| 이벤트 처리 결과를 통해 생성된 변환 행렬을 packet list에 추가하는 기능 구현 |  | 충돌 처리를 통해 발생하는 전송해야할 데이터를 packet\_list에 추가하는 기능 구현 | recvthread 마무리 |  | 3차 진행사항 체크 및 피드백 | 졸작 회의 |

임윤수 -

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 11/28 | 11/29 | 11/30 | 12/1 | 12/2 | 12/3 | 12/4 |
| 기능 상 미흡한 부분 점검 후 보완 | 기능 상 미흡한 부분 점검 후 보완 |  | 디버깅 및 버그 수정 | 디버깅 및 버그 수정 | 완성된 프로젝트 점검 및 디버깅 | 졸작 회의 |
| 12/5 | 12/6 | 12/7 | 12/8 |  |  |  |
| 1차 플레이 테스트 및 버그 수정 | 2차 플레이 테스트 및 버그 수정 | 마지막 플레이 테스트 및 버그 수정 | 제출 |  |  |  |

김동재 -

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 11/7(화) | 11/8(수) | 11/9(목) | 11/10(금) | 11/11(토) | 11/12(일) | 11/13(월) |
|  | worker thrad와 recvthread 생성 및 대기 구현 | 서버 전역변수 생성 |  | recvthread 에서 패킷을 수신 받아 queue에 삽입하는 기능 구현 (동기화 고려) | 1차 진행사항 체크 및 피드백 | 겜소공 시험 준비 |
| 11/14 | 11/15 | 11/16 | 11/17 | 11/18 | 11/19 | 11/20 |
| 데베 시험 준비 | 데베 시험 준비 | workerthread 내부에 클라이언트에게 보낼 패킷들을 관리하는 list 추가 및 테스트 | workerthread에서 이벤트 처리 후 전송할 패킷 생성 및 list 관리 구현 | 이벤트 처리를 통한 좌표 이동 결과를 클라이언트에게 전송하는 기능 추가 | 2차 진행사항 체크 및 피드백 | 졸작 회의 |
| 11/21 | 11/22 | 11/23 | 11/24 | 11/25 | 11/26 | 11/27 |
| 좌표를 전송하는 기존 구현에서 변환 행렬을 전송하도록 바꾸고 클라이언트에 적용 테스트 |  | 충돌 처리를 전송하고 클라이언트에서 수신받아 적용하는지 테스트 | workerthread 마무리 |  | 3차 진행사항 체크 및 피드백 | 졸작 회의 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 11/28 | 11/29 | 11/30 | 12/1 | 12/2 | 12/3 | 12/4 |
| 기능 상 미흡한 부분 점검 후 보완 | 기능 상 미흡한 부분 점검 후 보완 |  | 디버깅 및 버그 수정 | 디버깅 및 버그 수정 | 완성된 프로젝트 점검 및 디버깅 | 졸작 회의 |
| 12/5 | 12/6 | 12/7 | 12/8 |  |  |  |
| 1차 플레이 테스트 및 버그 수정 | 2차 플레이 테스트 및 버그 수정 | 마지막 플레이 테스트 및 버그 수정 | 제출 |  |  |  |

서동우 -

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 11/7(화) | 11/8(수) | 11/9(목) | 11/10(금) | 11/11(토) | 11/12(일) | 11/13(월) |
|  | 서버에 프레임워크 이식 | 졸작 회의 | workerthread 로직 구현 | HandleEvent함수 및  workerthread 로직 완성 | 1차 진행사항 체크 및 피드백 | 겜소공 시험 준비 |
| 11/14 | 11/15 | 11/16 | 11/17 | 11/18 | 11/19 | 11/20 |
| 데베 시험 준비 | 데베 시험 준비 | 졸작 회의 | 서버의 게임 메니저 클래스 구현 | 서버의 게임 메니저 클래스 완성 | 2차 진행사항 체크 및 피드백 | 클라이언트에 송신 받은 데이터 관리할 class 구현  (오브젝트 관리 class 자체 구현) |
| 11/21 | 11/22 | 11/23 | 11/24 | 11/25 | 11/26 | 11/27 |
| 클라이언트에 송신받은 데이터 관리할 class 완성  (오브젝트 관리 class 자체 구현) |  | 졸작 회의 | RecvCreate  Object(), RecvDelete  Object() 구현 | RecvUpdate  Transform()구현 | 3차 진행사항 체크 및 피드백 | SendHandle  InputEvent() 함수 구현 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 11/28 | 11/29 | 11/30 | 12/1 | 12/2 | 12/3 | 12/4 |
| 기능 상 미흡한 부분 점검 후 보완 | 기능 상 미흡한 부분 점검 후 보완 | 졸작 회의 | 디버깅 및 버그 수정 | 디버깅 및 버그 수정 | 완성된 프로젝트 점검 및 디버깅 |  |
| 12/5 | 12/6 | 12/7 | 12/8 |  |  |  |
| 1차 플레이 테스트 및 버그 수정 | 2차 플레이 테스트 및 버그 수정 | 마지막 플레이 테스트 및 버그 수정 | 제출 |  |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 11/7(화) | 11/8(수) | 11/9(목) | 11/10(금) | 11/11(토) | 11/12(일) | 11/13(월) |
|  |  |  |  |  |  |  |