경력기술서

게임클라이언트 최한일

경력

GitHub:
https://github.com/hanil12/Portfolio.git

언리얼과 유니티 경우 일부분의 코드만
올렸습니다.

1. SGA 게임 아카데미(1년 6개월)
   - 기초 수업(C++ ~ DirextX2D)
   - 알고리즘
   - Unity, Unreal 게임엔진
   - DataBase 기초 Query문 작성

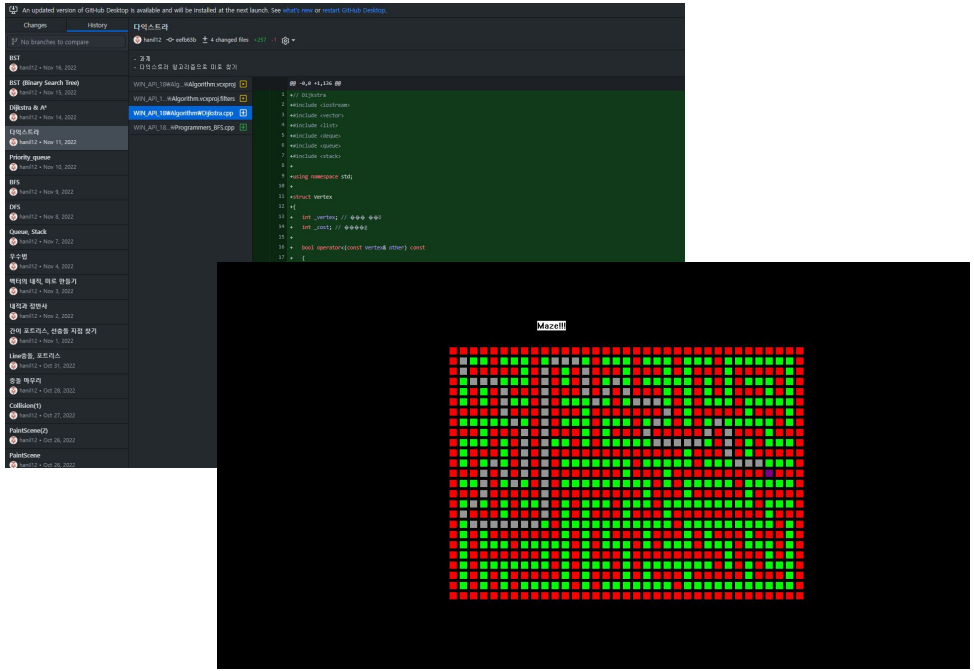
2. 주식회사 세오(3개월)
   - Unity 시각화

SGA 게임아카데미
기초 C++ ~ DirextX

	1010	OBB 충돌 마무리	ekdrhaod	2022.12.20.	6	0
	1008	AABB와 World	ekdrhaod	2022.12.19.	7	0
	881	Right Shift, Smart Pointer	ekdrhaod	2022.10.19.	7	0
	877	Modern(1)	ekdrhaod	2022.10.17.	7	0
	873	std::algorithm	ekdrhaod	2022.10.14.	3	0
	870	std::algorithm	ekdrhaod	2022.10.13.	6	0
	869	Iterator, Container	ekdrhaod	2022.10.12.	6	0
	862	vector와 list	ekdrhaod	2022.10.07.	11	0
	858	함수객체, 템플릿	ekdrhaod	2022.10.06.	6	0
	855	함수포인터	ekdrhaod	2022.10.05.	4	0
	851	Battle 과제 풀이	ekdrhaod	2022.10.04.	8	0
	848	디버깅(2), TextRPG, 추상클래스	ekdrhaod	2022.09.30.	18	0
	845	디버깅	ekdrhaod	2022.09.29.	6	0
	842	DeepCopy	ekdrhaod	2022.09.28.	10	0
	838	SingleTon, RTTI와 vftable (가상함수 테이블)	ekdrhaod	2022.09.27.	7	0

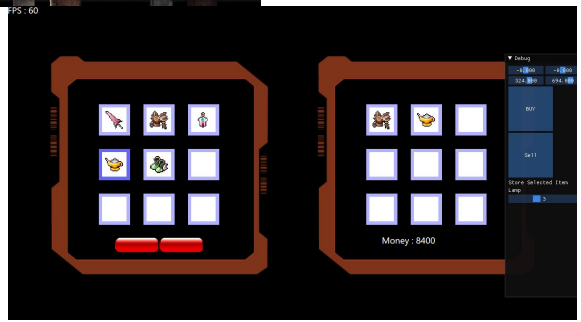
네이버 카페로 수업을 관리하며
변수, 메모리 영역, 클래스 등 기초적인 부분부터
시작하여 STL과 Modern(C++11)까지 수업을 성실히
진행하였습니다.

SGA 게임아카데미
기초 C++ ~ DirextX



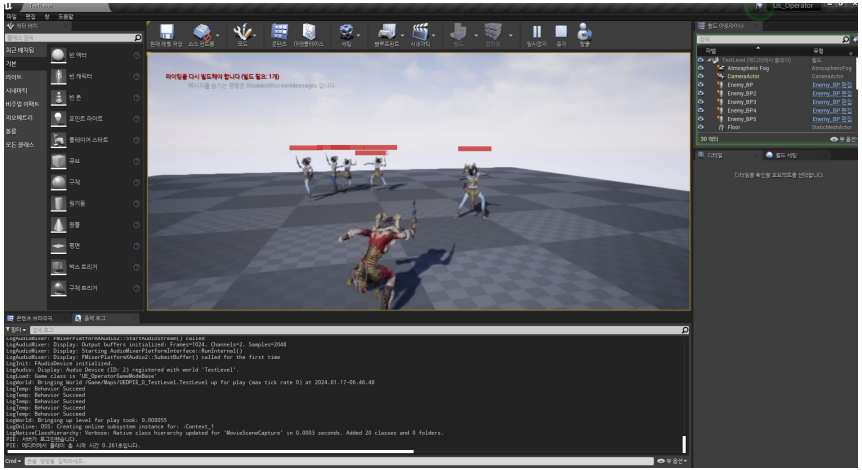
이후 Window API를 활용하여 기초적인 게임수학, 자료구조와 알고리즘에 대하여 수업하며, 미로를 만들고(Kruskal Algorithm) 탐색하는(A* Algorithm) 것을 목표로 수업을 진행하였습니다.

SGA 게임아카데미
기초 C++ ~ DirextX



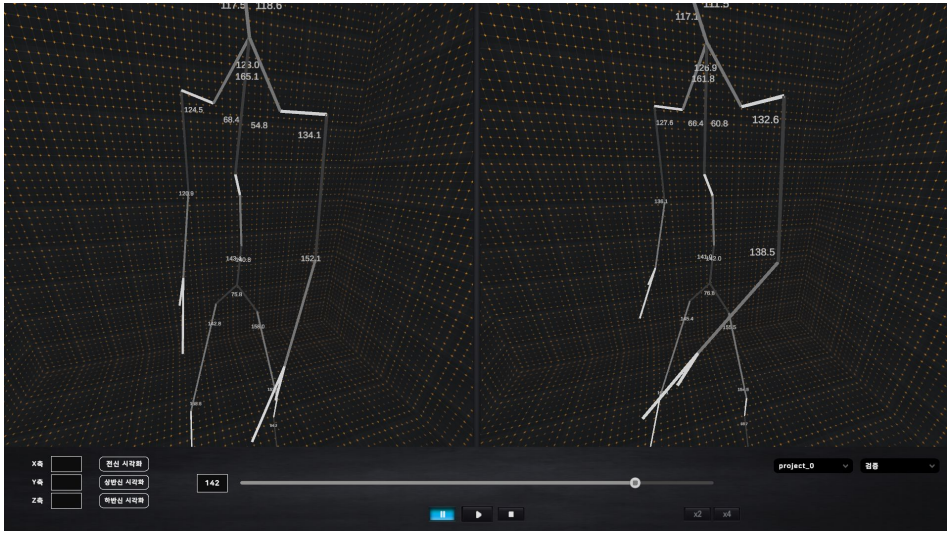
DirextX 교육과정에선 목표를 2D 게임제작엔진을 설계해보고, 직접 제작하기 / 인벤토리 구현해보기로 잡았습니다. 수업 내용으로는 선형대수학(행렬), CPU와 GPU, DirectX의 렌더링파이프라인에 대해서 수업을 진행하였습니다.

SGA 게임아카데미
기초 C++ ~ DirextX



DirectX 과정이 끝난 후 언리얼4에서 C++로 간단한
TPS 슈팅게임 제작을 목표로 수업을 진행하였습니다.

주식회사 세오
유니티 시각화



인공지능으로 도출된 관절점 프레임들을 이용하여 Unity에서 이를 3D로 시각화하여 보행을 확인할 수 있는 프로그램 제작.
주요업무로는

- 알고리즘을 이용하여 각 관절점을 이어 선으로 표시
- 전반적인 UI 프레임워크 제작
- 비디오편집 기능 추가

서버 포트폴리오
IOCP 서버 구현
간단한 채팅프로그램 제작

2024.01.20 ~
2024.04.03

목표

1. 서버에 대한 이해
2. 멀티쓰레드 환경에 대해서 이해하고 활용
3. 패킷 설계, 직렬화에 대한 이해와 응용

IOCP 기반 서버

```cpp
bool IocpCore::Register(IocpObjectRef iocpObject)
{
    return ::CreateIoCompletionPort(iocpObject->GetHandle(), _iocpHandle, 0, 0);
}

bool IocpCore::Dispatch(uint32 timeoutMs)
{
    DWORD numOfBytes = 0;
    ULONG_PTR key = 0;
    IocpEvent* iocpEvent = nullptr;

    if (::GetQueuedCompletionStatus(_iocpHandle, OUT & numOfBytes, OUT & key, OUT reinterpret_cast<LPOVERLAPPED*>(&iocpEvent), timeoutMs))
    {
        IocpObjectRef iocpObject = iocpEvent->owner;
        iocpObject->Dispatch(iocpEvent, numOfBytes);
    }
    else
    {
        int32 errCode = ::WSAGetLastError();
        switch (errCode)
        {
        case WAIT_TIMEOUT:
            return false;

        default:
            // TODO : 로그 찍기
            IocpObjectRef iocpObject = iocpEvent->owner;
            iocpObject->Dispatch(iocpEvent, numOfBytes);
            break;
        }
    }


    return false;
}
```

여러가지 서버 모델(OverlapEvent, OverlapCallBack)

을 공부해보고, IOCP모델을 채택하여

서버라이브러리를 제작하였습니다.

멀티쓰레드에 대한 이해

```
for (int32 i = 0; i < 5; i++)
{
    GThreadManager->Launch([&service]()
        {
            DoWorkerJob(service);
        });
}
```

```
void DoWorkerJob(ServerServiceRef& service)
{
    while (true)
    {
        LEndTickCount = ::GetTickCount64() + WORKER_TICK;

        // 네트워크 입출력 처리 -> 인게임 로직까지
        service->GetIocpCore()->Dispatch(10);

        // 예약된 일감 처리
        ThreadManager::DistributeReservedJobs();

        // 글로벌 큐
        ThreadManager::DoGlobalQueueWork();
    }
}
```

Atomic, SpinLock에 대해서 이해하고,

GlobalQueue를 만들어 클라이언트나 서버에서 생기는

Job들을 멀티쓰레드환경에서 처리하도록 했습니다.

패킷설계, 직렬화

```
struct MyPKT_S_TEST
{
public:
    struct BuffListItem
    {
        uint64 buffId;
        float remainTime;

        // victim List
        uint16 victimsOffset;
        uint16 victimsCount;
    };

    uint16 packetSize; // 공용헤더
    uint16 packetId; // 공용 헤더
    uint64 id;
    uint32 hp;
    uint16 attack;

    uint16 buffsOffset; // 가변 데이터의 시작 위치이며 고정길이데이터들의 합
    uint16 buffsCount;
};
```

```
class PKT_S_TEST_WRITE
{
public:
    using BuffsListItem = MyPKT_S_TEST::BuffListItem;
    using BuffsList = PacketList<MyPKT_S_TEST::BuffListItem>;
    using BuffsVictimList = PacketList<uint64>;

    PKT_S_TEST_WRITE(uint64 id, uint32 hp, uint16 attack)
    {
        _sendBuffer = GSendBufferManager->Open(4096);
        _bw = BufferWriter(_sendBuffer->Buffer(), _sendBuffer->AllocSize());

        _pkt = _bw.Reserve<MyPKT_S_TEST>();
        _pkt->packetSize = 0; //
        _pkt->packetId = S_TEST;
        _pkt->id = id;
        _pkt->hp = hp;
        _pkt->attack = attack;
        _pkt->buffsOffset = 0; //
        _pkt->buffsCount = 0; //
    }

    BuffsList ReserveBuffsList(uint16 buffCount)
    {
        BuffsListItem* firstBuffList = _bw.Reserve<BuffsListItem>(buffCount);
        _pkt->buffsCount = buffCount;
        _pkt->buffsOffset = (uint64)firstBuffList - (uint64)_pkt;

        return BuffsList(firstBuffList, buffCount);
    }

    BuffsVictimList ReserveBuffsVictimsList(BuffsListItem* buffsItem, uint16 vic
```

가변배열을 가진 패킷을 설계해보고,

이를 직렬화하여 패킷 송수신을 해봤습니다.

패킷설계, 직렬화
ProtoBuf

```
syntax = "proto3";
package Protocol;

import "Enum.proto";
import "Struct.proto";

message S_TEST
{
    uint64 id = 1;
    uint32 hp = 2;
    uint32 attack = 3;
    repeated BuffData buffs = 4;

    //enum PacketId { NONE = 0; PAC
}

message C_CREATE_ACCOUNT
{
    string id = 1;
    string passWord = 2;
    string name = 3;
}

message S_CREATE_ACCOUNT
{
    bool success = 1;
    string id = 2;
    string passWord = 3;
    string name = 4;
}

message C_LOGIN
{
    string id = 1;
    string passWord = 2;
}
```

```
// Custom Handlers
bool Handle_INVALID(PacketSessionRef& session, BYTE* buffer, int32 len);

// Python
bool Handle_C_CREATE_ACCOUNT(PacketSessionRef& session, Protocol::C_CREATE_ACCOUNT& pkt);
bool Handle_C_LOGIN(PacketSessionRef& session, Protocol::C_LOGIN& pkt);
bool Handle_C_ENTER_GAME(PacketSessionRef& session, Protocol::C_ENTER_GAME& pkt);
bool Handle_C_CHAT(PacketSessionRef& session, Protocol::C_CHAT& pkt);

class ClientPacketHandler
{
public:
    static void Init()
    {
        for (int32 i = 0; i < UINT16_MAX; i++)
        {
            GPacketHandler[i] = Handle_INVALID;
        }

// Python
        GPacketHandler[PKT_C_CREATE_ACCOUNT] = [](PacketSessionRef& session, BYTE* buffer, int32 len)
        {
            return HandlePacket<Protocol::C_CREATE_ACCOUNT>(Handle_C_CREATE_ACCOUNT, session, buffer, len);
        };
        GPacketHandler[PKT_C_LOGIN] = [](PacketSessionRef& session, BYTE* buffer, int32 len)
        {
            return HandlePacket<Protocol::C_LOGIN>(Handle_C_LOGIN, session, buffer, len);
        };
        GPacketHandler[PKT_C_ENTER_GAME] = [](PacketSessionRef& session, BYTE* buffer, int32 len)
        {
            return HandlePacket<Protocol::C_ENTER_GAME>(Handle_C_ENTER_GAME, session, buffer, len);
        };
        GPacketHandler[PKT_C_CHAT] = [](PacketSessionRef& session, BYTE* buffer, int32 len)
        {
            return HandlePacket<Protocol::C_CHAT>(Handle_C_CHAT, session, buffer, len);
        };
```

패킷에 가변배열이 중첩해 있을 경우, 프로토콜 많을 경우

이를 편하게 서버,클라이언트에서 편하게 직렬화하기 위해 ProtoBuf를 사용하였습니다.

봐주셔서 감사합니다.