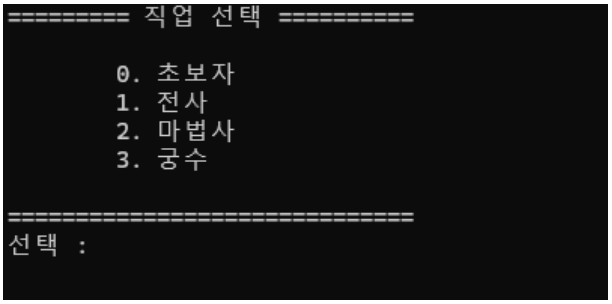
TEXT RPG

develop period	@2024년 6월 1일 → 2024년 6월 2일
roll	프로그래밍
category	텍스트 게임
FrameWork	MySQL Workbench
언어	C++ MySQL

개요



전남대학교 데이터베이스 시스템 수업 중 DBMS 구축 과제를 받아 C++로 직접 TEXT 게임을 제작해보았습니다. TEXT 게임에서 나온 데이터를 제가 만든 DBMS에 저장하고 게임을 껐다 켰을 때 불러오기 기능 구현을 목표로 DBMS와 게임을 설계 했습니다.

구동 환경

- Visual Studio 2022
- MySQL server 8.0.36

개발 내용

- TEXT RPG
- DBMS

게임 소개

전체 프로젝트:

https://github.com/kimkyungjae1112/TEXTRPG

데이터베이스를 설계하기 전 설계한 데이터베이스를 사용할 수 있는 실제 프로그램이 있는게 설계가 제대로 되었는지 판단하기 쉽다고 생각했습니다. 그래서 설계를 마친 후 C++를 사용해 간단한 TEXT RPG 게임을 제작했습니다.

게임은 단순히 마지막 보스를 처치하면 이기는 게임입니다. 플레이어는 x(0~10), y(0~10) 좌표공간을 이동하며 몬스터를 처치 혹은 도망가며 성장할 수 있고 플레이어의 x 좌표가 10에 도달하게 되면 다음 맵으로 넘어갑니다. 마지막 맵에 도달해 x 좌표가 10에 도달하게 되면 보스 몬스터를 만나며 처치 시 게임을 클리어 할 수 있습니다.

```
//Mysql
void CleanupDatabase(); // 프로그램이 끝날 때 데이터베이스 데이터 삭제
void InsertPlayerStat(); // PlayerStat 테이블에 데이터 삽입
void InsertPlayerXp(); // PlayerXp 테이블에 데이터 삽입
void InsertPlayer(); // Player 테이블에 데이터 삽입
void InsertMonster(MonsterType* Mt); // Monster 테이블에 데이터 삽입
void InsertJob(CharacterJob* Cj); // Job 테이블에 데이터 삽입
void InsertBattle(const int RunFlag); // Battle 테이블에 데이터 삽입
void InsertMapManager(); // MapManager 테이블에 데이터 삽입
void InsertManage(); // Manage 테이블에 데이터 삽입
void InsertMap(Region region); // Map 테이블에 데이터 삽입
```

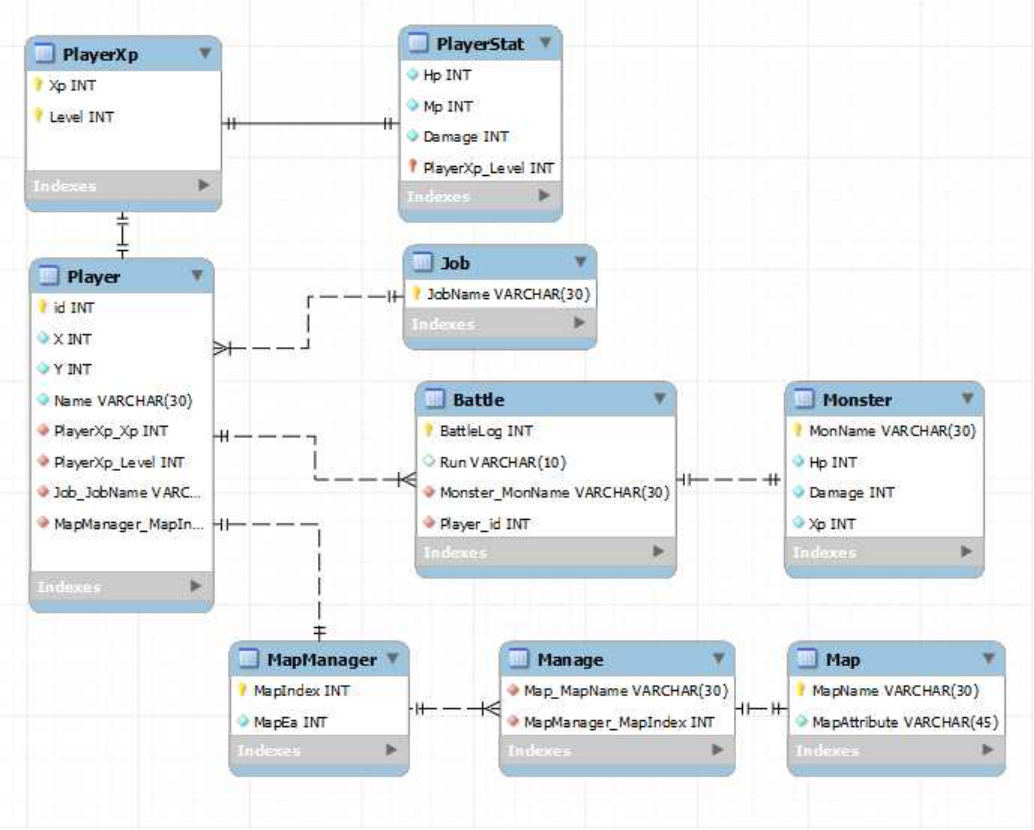
C++과 MySQL을 연동하여 게임이 진행될 때 저장되어야 할 데이터를 데이터베이스에 저장할 수 있도록 코드를 작성했습니다.

플레이 영상

https://prod-files-secure.s3.us-west-2.amazonaws.com/7ebf7000-f855-492a-973c-2e002d905ac7/d4993394-2156-4b82-896b-6e9b0919afad/%ED%94%8C%EB%A0%88%EC%9D%B4%EC%98%81%EC%83%81.mp4

DBMS

DBMS를 구축하기 위해 MySQL Workbench를 사용했습니다.



MySQL Workbench에서 ER 다이어그램을 만들면 MySQL 코드로 바꿔주는 도구를 사용해 구축할 수 있었습니다. 게임을 진행하며 데이터베이스에 데이터가 쌓이는 것을 확인할 수 있습니다.

1. 게임 시작시

```
MYSQL 초기화 완료
INSERT Map 성공
INSERT Map 성공
INSERT Map 성공
INSERT Job 성공
INSERT Job 성공
INSERT Job 성공
INSERT Job 성공
INSERT Monster 성공
INSERT Monster 성공
INSERT Monster 성공
INSERT MapManager 성공
INSERT Manage 성공
========= 직업 선택 =========

        0. 초보자
        1. 전사
        2. 마법사
        3. 궁수

==============================
선택 :
```

게임 시작시 Map, Job, Monster,MapManager, Manage 테이블 등에 기본 데이터가 들어가게 됩니다.

	MapName	MapAttribute
▶	FROZEN	fro
	GROUND	gr
	VOLCANO	vol
*	NULL	NULL

Map 테이블

	MapIndex	MapEa
▶	0	3
*	NULL	NULL

MapManager 테이블

	JobName
▶	Archer
	Beginner
	Warrior
	Wizard
*	NULL

Job 테이블

	MonName	Hp	Damage	Xp
▶	Boss	500	30	20
	Goblin	200	10	5
	Oak	350	15	10
*	NULL	NULL	NULL	NULL

Monster 테이블

	Map_MapName	MapManager_MapIndex
▶	Ground	0

Manage 테이블

MySQL 서버에 데이터가 삽입된 모습

2. 직업 선택시

```
========== 직업 선택 ==========

        0. 초보자
        1. 전사
        2. 마법사
        3. 궁수

==============================
선택 : 2
캐릭터 닉네임 설정 : KIM

========== 마법사 선택 ==========

        체력 : 80
        마나 : 150
        공격력 : 100

==============================
INSERT Xp 성공
INSERT Player 성공
INSERT Stat 성공
```

직업 선택시 PlayerXp, Player, PlayerStat 테이블 등에 기본 데이터가 들어가게 됩니다.

	id	X	Y	Name	PlayerXp_Xp	PlayerXp_Level	Job_JobName	MapManager_MapIndex
▶	0	0	0	KIM	0	1	Wizard	0
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Player 테이블

	Hp	Mp	Damage	PlayerXp_Level
▶	80	150	100	1
*	NULL	NULL	NULL	NULL

PlayerStat 테이블

	Xp	Level
▶	0	1
*	NULL	NULL

PlayerXp 테이블

MySQL 서버에 데이터가 삽입된 모습

3. 전투 상황

```
====================== Player ======================
이름 : KIM
레벨 : 1         경험치 : 0
공격력 : 100
체력 : 80 /     마나 : 150
====================== Monster =====================
이름 : Goblin
경험치 : 5
공격력 : 10
체력 : 200

======================== 선택 ========================
1. 공격
2. 방어
3. 도망
선택 : 1
=========== KIM의 공격 ===========
Goblin의 체력 : 200
Goblin의 남은 체력 : 100

=========== Goblin의 공격 ===========
KIM의 체력 : 80
KIM의 남은 체력 : 70
INSERT Battle 성공
```

몬스터와 마주쳐 전투 상황시 Battle 테이블에 전투 기록이 남게됩니다.

	BattleLog	Run	Monster_MonName	Player_id
▶	0	No	Goblin	0
*	NULL	NULL	NULL	NULL

Battle 테이블

MySQL 서버에 데이터가 삽입된 모습

4. 몬스터 처치시

```
========== 몬스터 처치 완료 ==========
INSERT Xp 성공


=========== 레벨업 ===========
        Hp 증가 + 30
        Mp 증가 + 30
        공경력 증가 + 10

INSERT Xp 성공
INSERT Stat 성공
```

몬스터 처치시 Xp가 오르게 되며 일정 수준 오르면 레벨업을 하게 됩니다. 따라서 몬스터를 처치하거나 레벨업시 PlayerStat 테이블과 PlayerXp 테이블에 데이터가 저장됩니다.

	Hp	Mp	Damage	PlayerXp_Level
▶	80	150	100	1
	110	180	110	2
*	NULL	NULL	NULL	NULL

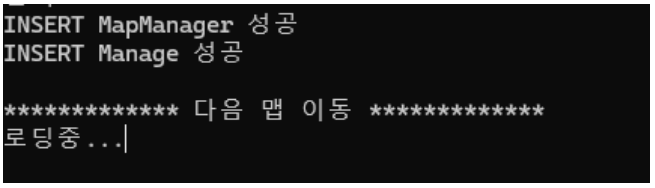
PlayerXp 테이블

	Xp	Level
▶	0	1
	5	1
	0	2
	5	2
*	NULL	NULL

PlayerStat 테이블

MySQL 서버에 데이터가 삽입된 모습

5. 맵 이동시



맵 이동시 MapManager, Manager 테이블에 추가 데이터가 입력됩니다.

	Map_MapName	MapManager_MapIndex
▶	Ground	0
	Frozen	1
	Volcano	2

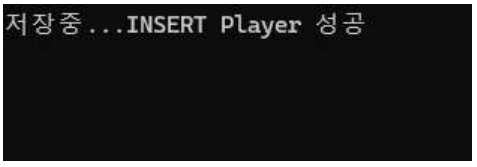
Manage 테이블

	MapIndex	MapEa
▶	0	3
	1	3
	2	3
*	NULL	NULL

MapManager 테이블

MySQL 서버에 데이터가 삽입된 모습

6. 저장시



저장시 Player 테이블에 현재 플레이어의 상태를 저장합니다. 플레이어의 상태에는 현재 좌표와 캐릭터 이름, 현재 레벨과 경험치, 직업, 맵의 위치 정보가 있습니다.

id	X	Y	Name	PlayerXp_Xp	PlayerXp_Level	Job_JobName	MapManager_MapIndex
0	0	0	KIM	0	1	Wizard	0
1	0	0	KIM	15	4	Wizard	2
2	0	0	KIM	15	4	Wizard	2
3	0	0	KIM	15	4	Wizard	2
4	0	0	KIM	15	4	Wizard	2
5	0	0	KIM	15	4	Wizard	2
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Player 테이블

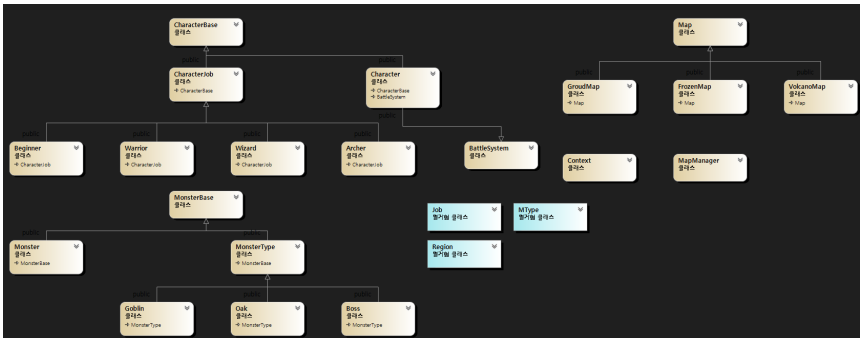
7. 클리어 및 종료시

```
DELETE battle 성공
DELETE manage 성공
DELETE playerstat 성공
DELETE player 성공
DELETE playerxp 성공
DELETE monster 성공
DELETE job 성공
DELETE map 성공
DELETE mapmanager 성공
```

클리어 및 종료시 모든 테이블에 있는 데이터가 삭제되며 프로그램이 종료됩니다.

구현 기술

클래스 구조



디자인 패턴

- 파사드 패턴
  - 파사드 패턴은 일련의 저수준 인터페이스들을 하나의 고수준 인터페이스로 묶어주는 패턴이다.
  - 파사드 패턴이 적용된 클래스 : Context
  - 캐릭터 직업 선택, 공격, 이동, 몬스터 생성, 맵 이동 등 모든 객체의 행동을 Context 클래스로 감싸, 게임의 메인 루프에서 Context 객체만으로 간단하게 구현함

    - Context 클래스에 서버에 데이터를 등록하는 로직을 작성해 각각의 객체에서 MySQL문을 사용하지 않고 Context 클래스에서 작업을 가능하게 함
- 싱글톤 패턴
    - 싱글톤 패턴이 적용된 클래스 : Context
- 팩토리 패턴
    - CharacterBase와 MonsterBase는 추상 팩토리로 작동하며, 캐릭터와 몬스터 생성을 위한 인터페이스를 아래와 같은 순수 가상 함수를 통해 제공함

```
// 캐릭터의 팩토리 메소드
virtual void BaseSetCharacter(Job job) = 0;

// 몬스터의 팩토리 메소드
virtual void BaseSetMonster(MType Type) = 0;
```

    - 위의 함수들을 각각 Character 클래스와 Monster 클래스에서 재정의 하여 해당 인터페이스를 통해 객체를 생성할 수 있음
    - Character는 Job 열거형, Monster는 MType 열거형으로 직업을 구체적으로 가짐

▼ 종합 코드

```
void Character::MeetMonster()
{
    int MeetRatio = rand() % 3;
    int Type = rand() % 6;
    if (MeetRatio == 0)
    {
        RunFlag = true;
        monster = new Monster();
        if(Type >= 0 || Type <= 3) monster->BaseSetMonster(MType::GOBLIN);
        else if(Type >= 4 || Type <= 5)monster->BaseSetMonster(MType::OAK);
        DetectedMonster(monster);

        Context* context = Context::GetContext();

        context->BattleInterface();
    }
}
```

- 캐릭터가 맵을 이동하면 특정 확률로 몬스터를 만나게 됨
- 몬스터를 만나면 일정 확률에 따라 만나는 몬스터의 종류가 결정되고, 이때 팩토리 메소드를 통해 몬스터를 결정
- 캐릭터는 만난 몬스터를 싸울 대상으로 등록
- 싱글톤 패턴으로 작성된 Context 포인터를 받아와서, Context의 BattleInterface() 함수 호출로 전투 장면으로 화면이 전환됨

인터페이스 사용

▼ BattleSystemInterface

```
class BattleSystem
{
public:
    virtual void Attack() = 0;
    virtual void Defend() = 0;
    virtual void Run() = 0;
    virtual ~BattleSystem() {}
};
```

- 캐릭터 클래스에 공격, 방어, 도망 기능을 하드 코딩 하지 않고, 인터페이스를 구현하여 기능들을 구현함
- 현재 몬스터의 공격 방식이 캐릭터의 Attack 함수 안에 구현되어 있지만, 후에 몬스터의 공격 패턴을 바꾸고 싶을 때 해당 인터페이스를 구현함으로써 기능 확장을 열어놓음

회고

알게된 점

1. 그 당시 이론으로만 배웠던 디자인 패턴들을 직접 작성한 코드에 적용해보며 막연히 어렵게만 느껴졌던 디자인 패턴을 사용한 깔끔한 코드 설계에 자신감을 얻게 되었다.
2. 특정 엔진을 사용하는 것이 아닌 C++ 언어만을 사용해 게임을 만들며 메모리 할당과 해제의 중요성, 댕글링 포인터와 참조같이 자원 관리의 중요성과 게임의 메인 루프같이 엔진의 주요 기능에 대해 알 수 있었다.

아쉬운점

1. 데이터베이스에 데이터를 넣을 때 인코딩 방식이 달라 한글로는 데이터를 넣지 못했다. Workbench의 인코딩 바꾸는 방법을 찾지 못해 모든 데이터를 영어로 넣게 되어 아쉬웠다.
2. 맵을 종류별로 나눴는데, 맵별 특징을 넣었다면 게임의 컨텐츠가 더욱 풍부했을 것 같다.
3. 데이터 저장 기능만 존재하고 불러오기 기능을 넣지 못했다.