hcsv

[c1, c2, c3, c4 [c41, c42, c43], c5, c6, c7[c71[c711, c712, c713], c72[c721, c722, c723], c73 [c731, c732, c734]], c8, c9]

위와 같이 계층적으로 정의된 csv이다. excel에서 볼 때 일반 csv와 차이는 없다. 스키마에 의해 의미가 결정된다. 타잎에 따라 struct나 배열이 될 수 있다. 각 컬럼은 타잎을 갖는다. 기본 타잎들과 enum 값일 수 있다. enum은 별도 csv 파일에 정의한다.

하나의 hcsv 파일은 내부에 스키마 정의를 갖고 있다. 간단하고 강력하며 기존 툴들을 활용할 수 있다. 전체는 struct이고 내부에 배열과 struct를 가진다. son과 거의 유사한 기능을 제공할 수 있으며 다른 hcsv의 필드 참조 타잎으로 유효성 체크와 손쉬운 편집 기능을 제공할 수 있다.

이 방향으로 생각하고 정리해서 만든다. 당분간은 이 쪽으로 충분할 것 같다.

스키마 언어

예시를 만들면서 아이디어를 정리하고 BNF로 최종 정리한다. 코드 생성은 정리되어야 알 수 있다.

Design By Example

```
include "file";

table L2.Game.Skill;

enum SkillType
{
    Passive = 1,
    ActiveAttack,
    ActiveDefense,
    ActiveAttackDebuff,
    MonsterSkill,
}

struct Ready
{
    float speed1;
    float speed2;
    float speed3;
    float move;
};

struct Projectile
{
    bool straight;
    float speed;
    float extent;
```

```
    u32 ttl;
    u32 skillId;
    u32 multiArrive;
    u32 interval;
    u32 angle;
    float minRange;
    float maxRange;
    float flyRange;
}

constraint on Projectile  {
    range   ( index, 1);
    default ( ttl, 0 );
    default ( maxCount, 0);
    foreign ( skillId, Game.Skill.index );
}

desc on Projectile {
    straight : "설명";
}

const {
    u32 passiveSkillCount = 3;
}

main {
    u32             index;
    u32             denominator;
    string          nameKr;
    SkillType       type;
    u32             level;
    bool            isMovingSkill;
    u32             passiveValues[PassiveCount];
    ConditionEquip   equipRight;
    Ready           ready;
    sub<Projectile>  projectiles;
}

constraint on Skill {
    primary ( index );
    index   { nameKr );
    range   ( level, 1, 200 );
    default ( level, 1);
}

desc on Skill {
    필드별 설명
}
```

스펙

키

단일 필드 또는 여러 필드에 대한 키이다. Find() 함수를 정의한다.

- primary
- unique
- index
    - non-unique index
- foreign
    - 다른 테이블의 키 (unique or primary)

범위

각 타잎의 범위는 .으로 구분한다. 없으면 현재 이름공간 (현재 파일)에서 찾는다. 글로벌 공간은 없다. 하나의 파일은 하나의 테이블을 정의하는 것으로 한다.

배열

passiveValues.1, passiveValues.2, passiveValues.3 와 같이 필드명으로 컬럼을 생성한다. 실제 코드는 다음 형태가 된다.

```
/// 값 접근
const uint32_t& GetPassiveValues( const uint32_t index ) const;

/// 개수 접근
uint32_t GetPassiveValuesCount() const;
```

하위 반복 필드

sub 키워드로 지정. sub 은 Projectile을 하위 구분자로 구분된 필드를 추가. 복잡도를 줄이기 위해 한 단계의 반복 필드만 가능하다. 코드 생성은 C++ 기준으로 대략 다음과 같다.

```
// 데이터 멤버
private:
    std::vector<Projectile> projectiles_;

// 값 접근자. 배열과 유사

const Projectile& GetProjectile( const uint32_t index ) const;

uint32_t GetProjectileCount() const;
```

코드 생성

table을 class로 하고 함수를 생성한다. 이름 규칙에 따라 생성한다. (PascalCase를 가정한다)

```
#include "Table/Game/Common.h"

namespace L2 {
namespace Game {

class SkillElem : public Element
{
public:
    const uint32_t& GetIndex() const;

    // ....
};

class TableSkill
{
public:
    TableSkill();

    ~TableSkill();

    bool Load( const std::string& file );

    bool Reload( const std::string& file );

    bool Validate();

    SkillElem::Ref GetByIndex( uint32_t v ) const;

    SkillElem::Vec GetByNameKr( const std::string& name ) const;
};

} // Game
} // L2
```

Validate() 는 모든 테이블을 로딩한 후 호출한다. Load 중에는 range, 키 값, 기본 값 등을 검증한다.

TableManager에서 로딩을 전부 한다. Reload를 위해 TableManager를 통해 다시 가져와야할 지 여부를 설정한 빠른 포인터 싸개인 Ref를 사용한다. 쓰레드에 안전해야 한다. shared_ptr과 현재 세대를 기억하는 정도면 가능하다.

엑셀 파일

과제들

- 기획자가 수정한 내용의 반영
  - 스키마 변경 시에도 기존 데이터와 엑셀 내용이 변경되지 않아야 함
  - 엑셀의 프로그래밍 기능 등

- 주석 등 표시 방법
- 실제 RPG 게임의 구성