

# Laporan Tugas Besar 1

## PurryLeveling

Disusun untuk memenuhi tugas mata kuliah IF2010 Pemrograman Berorientasi Objek pada Semester 2 (Genap) Tahun Akademik 2024/2025



~ CTF ~

**Adit, silakan maju ke depan untuk mengerjakan soal STL berikut**

13523002	Refki Alfarizi
13523004	Razi Rachman Widyadhana
13523011	Muhammad Ra'if Alkautsar
13523028	Muhammad Aditya Rahmadeni
13523090	Nayaka Ghana Subrata

**Asisten Pembimbing:**

13521055      Muhammad Bangkit Dwi Cahyono

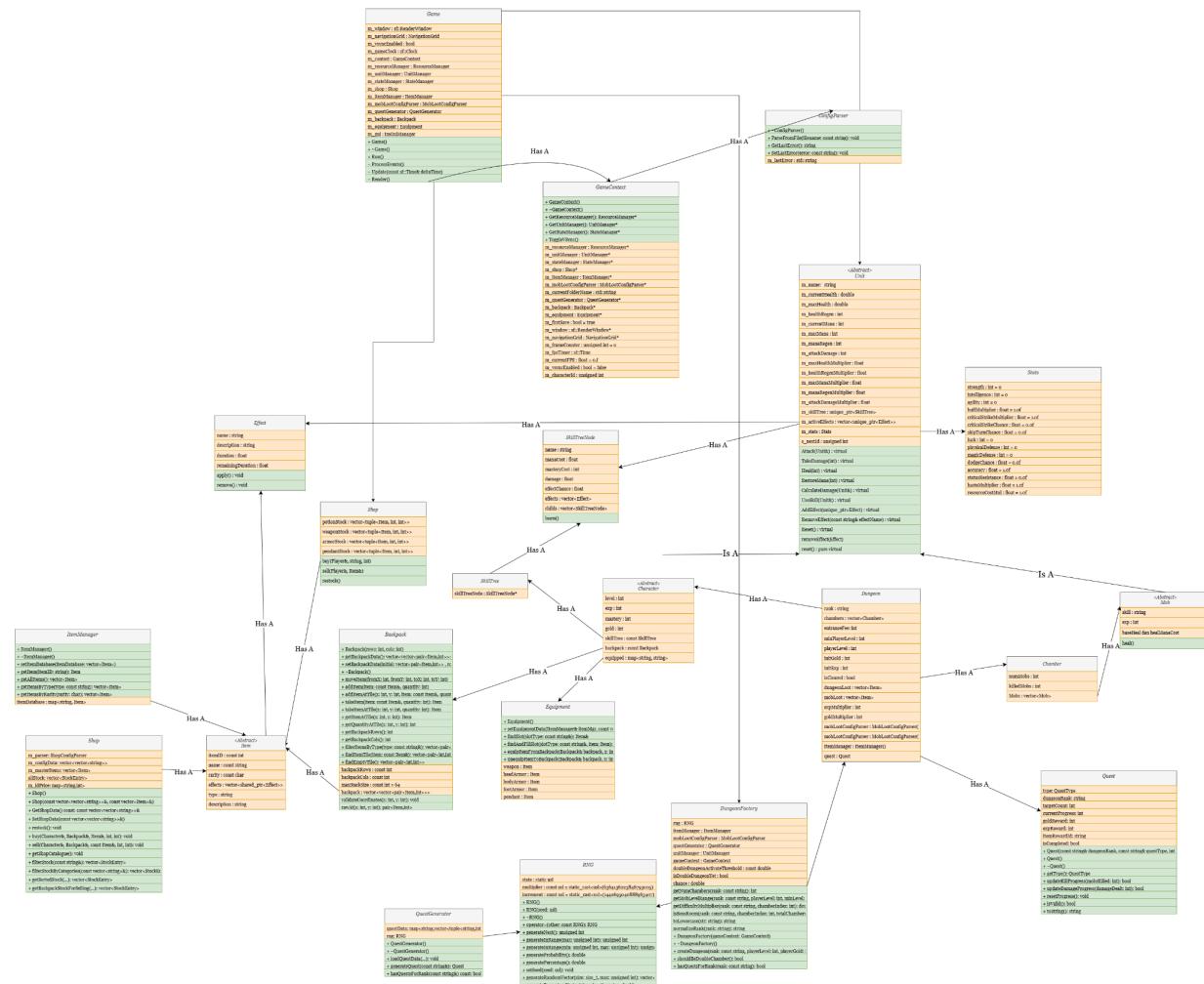
## Daftar Isi

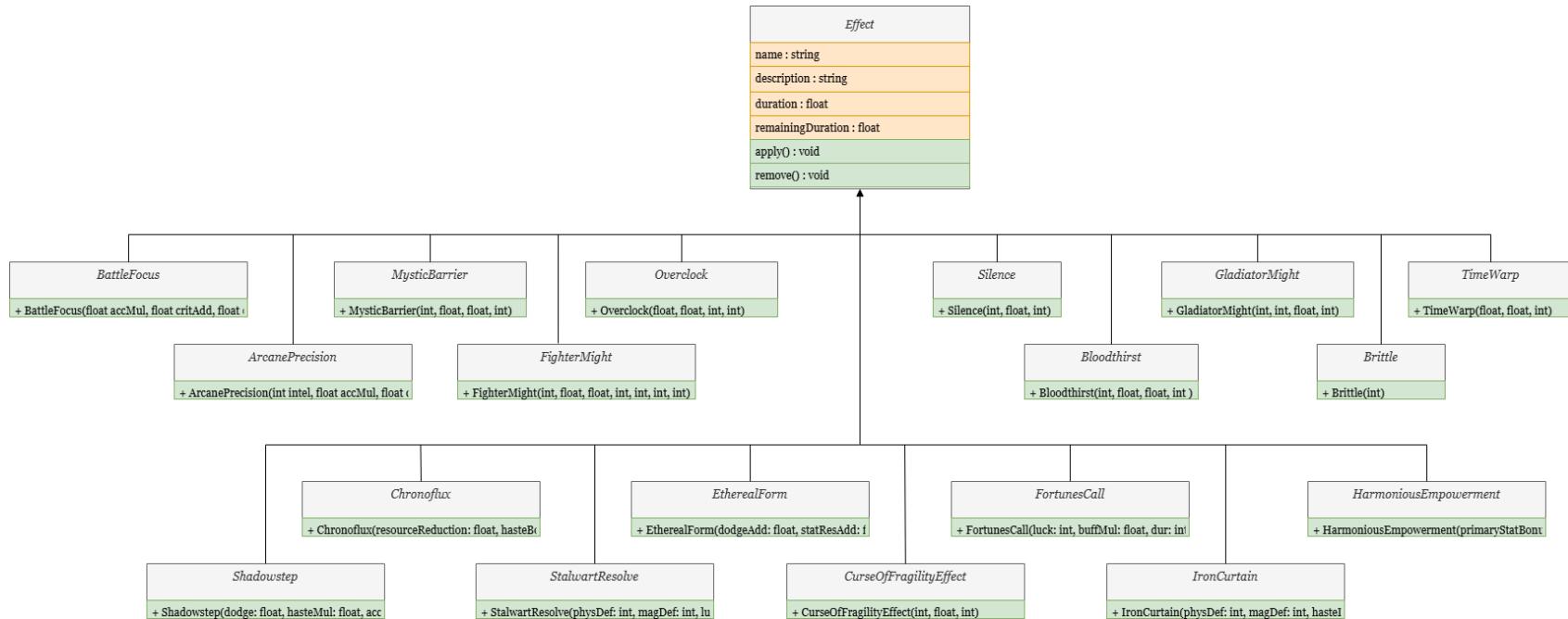
1. Diagram Kelas.....	1
2. Penerapan Konsep OOP.....	6
2.1. Inheritance & Polymorphism.....	6
2.1.1. Effect.....	6
2.1.2. Unit.....	10
2.1.3. Skill.....	19
2.2. Method/Operator Overloading.....	23
2.2.1. Item.....	24
2.2.2. Stats.....	24
2.2.3. RNG (Random Number Generator).....	27
2.3. Template & Generic Classes.....	28
2.4. Exception.....	29
2.4.1. Inventory (Backpack dan Equipment).....	30
2.4.2. Shop.....	31
2.5. C++ Standard Template Library.....	33
2.5.1. std::vector.....	33
2.5.2. std::map.....	34
2.5.3. std::set.....	36
2.5.4. std::pair.....	37
2.5.5. std::tuple.....	38
2.5.6. std::string.....	38
2.5.7. std::unique_ptr.....	39
2.5.8. std::queue.....	39
2.5.9. std::function.....	40
2.6. Konsep OOP lain.....	40
2.6.1. Abstract Base Class.....	40

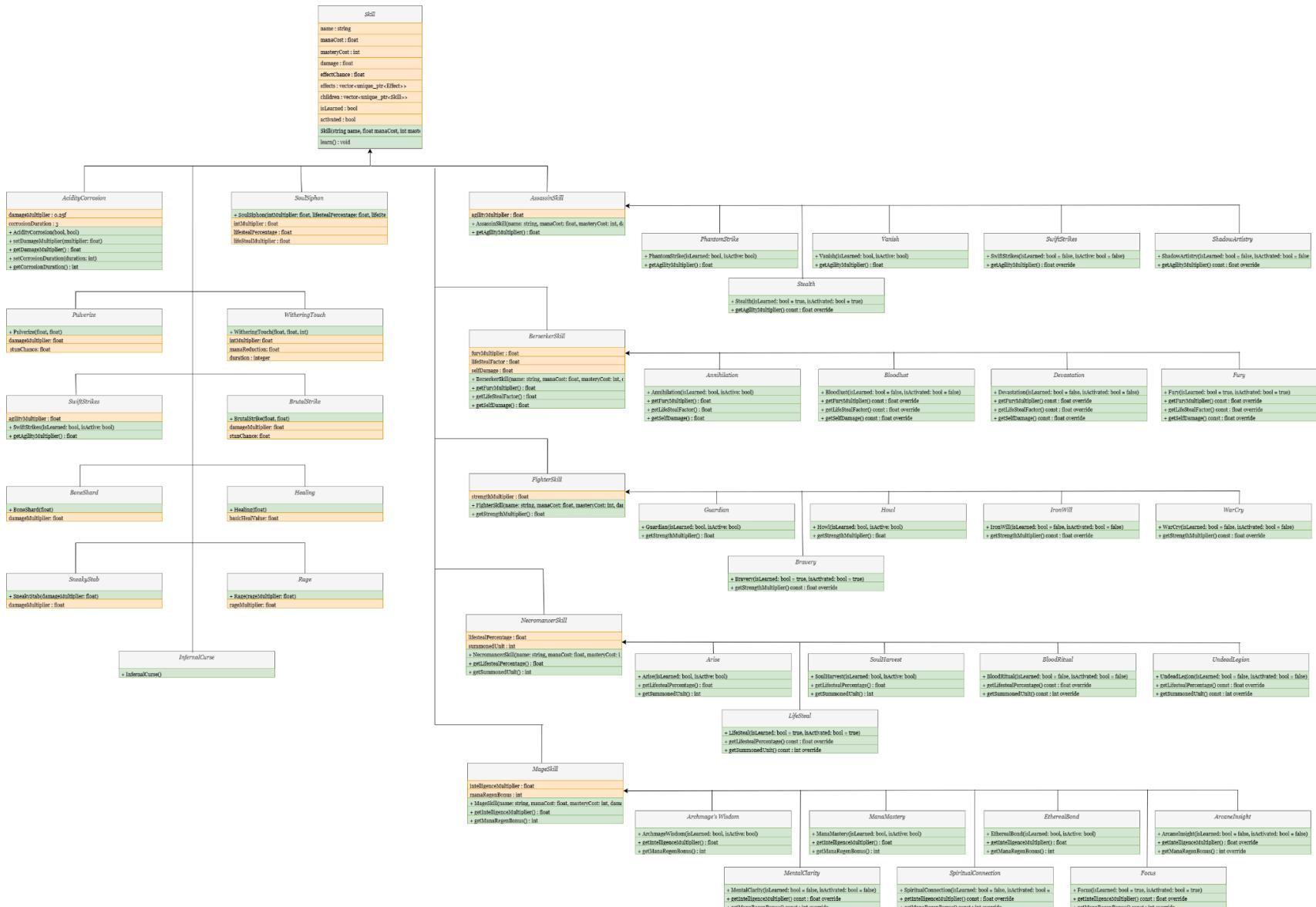
2.6.2. Dungeon.....	40
<b>3. Bonus Yang dikerjakan.....</b>	<b>42</b>
3.1. Bonus yang diusulkan oleh spek.....	42
3.1.1. Quest System.....	42
3.1.2. Cheat.....	48
3.1.3. Graphical User Interface (GUI).....	51
3.2. Bonus Kreasi Mandiri.....	62
3.2.1. Custom RNG Algorithm.....	62
3.2.2. Musik pada Game.....	67
3.2.3. Algoritma Sorting Pada Shop.....	68
3.2.4. Levelling Progression.....	72
3.2.5. Tampilan GUI yang indah dan menawan.....	73
<b>4. Pembagian Tugas.....</b>	<b>78</b>

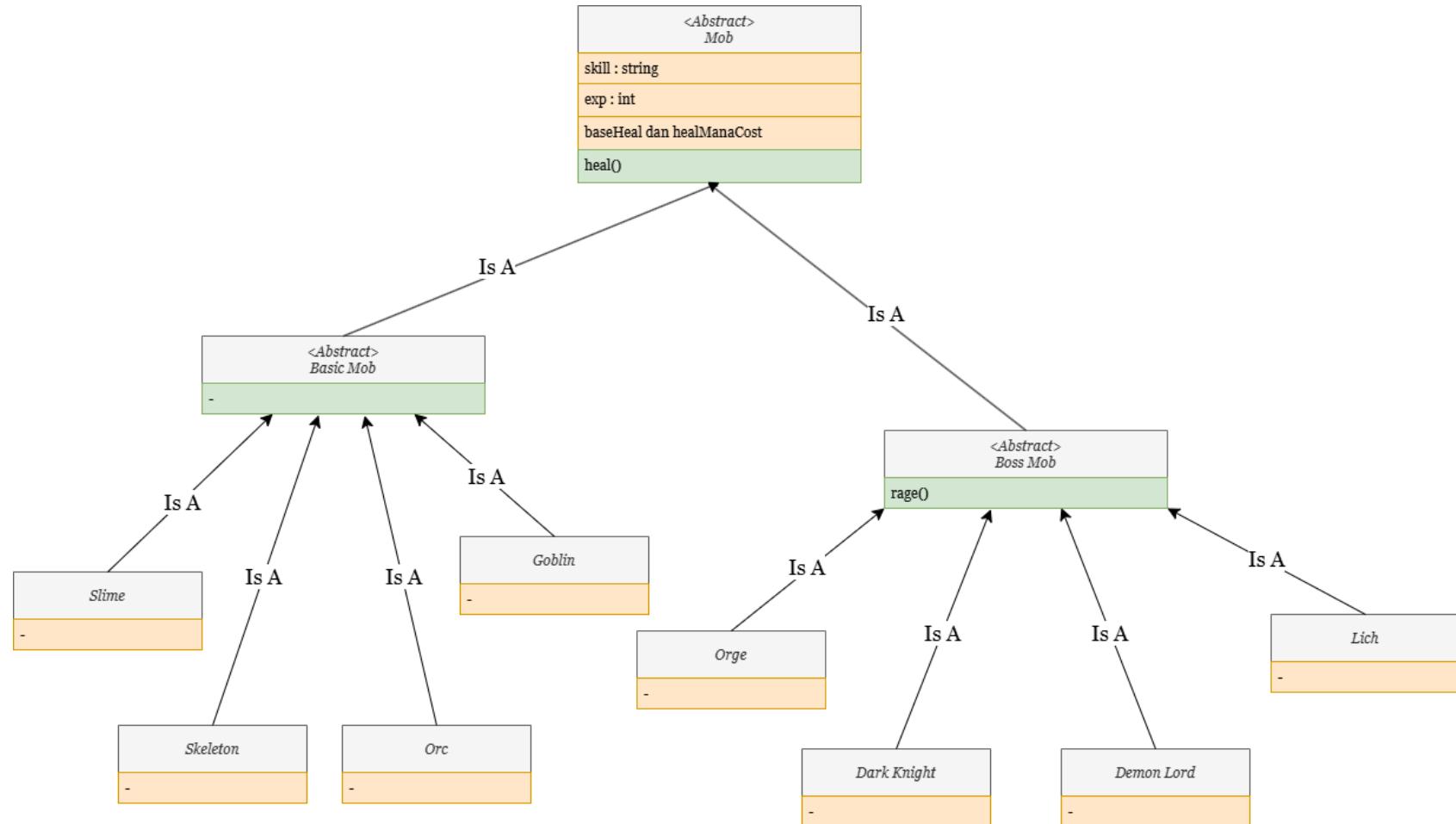
## 1. Diagram Kelas

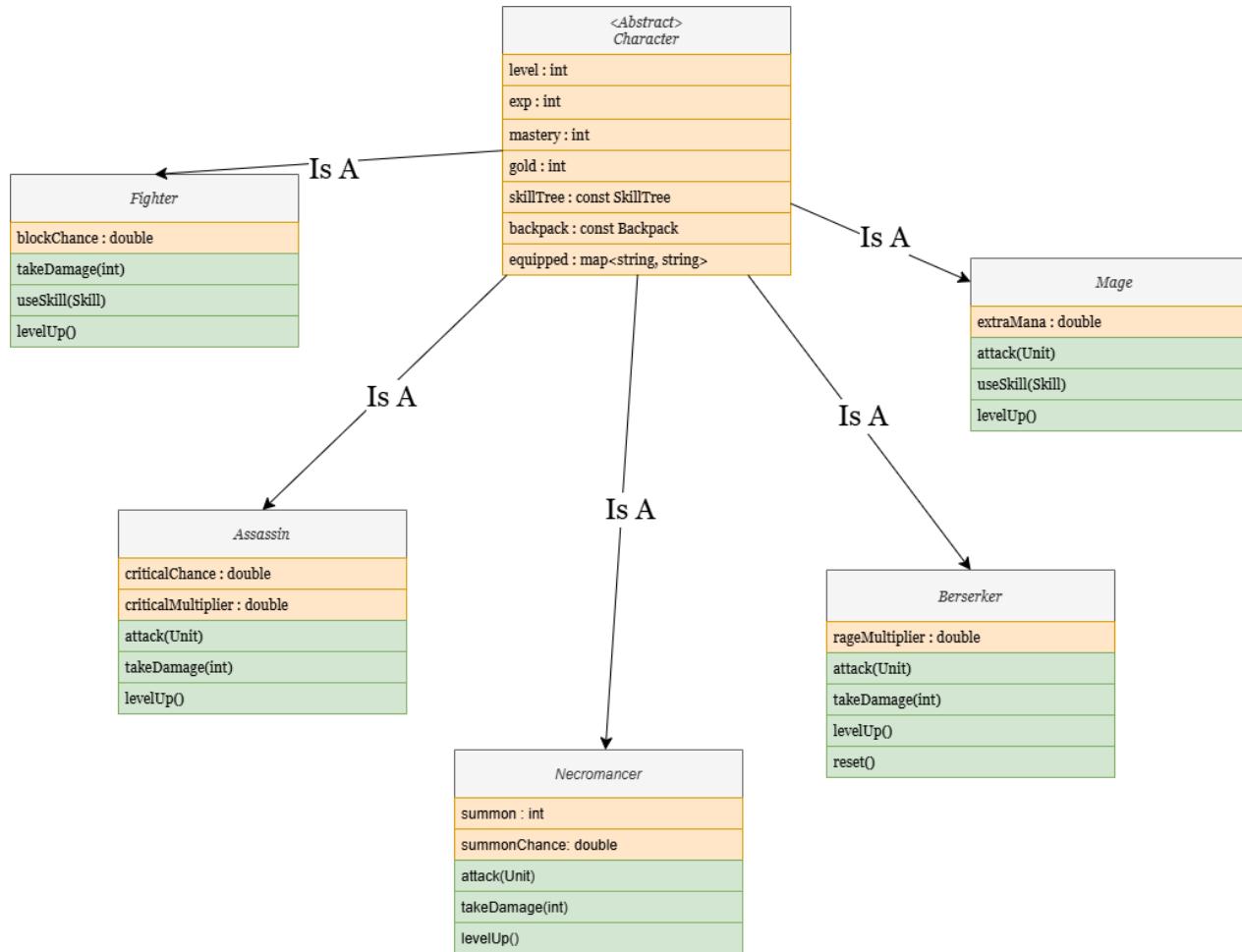
Link menuju diagram: <https://drive.google.com/file/d/15NGKehPTBax38nNTSitzeTPHxhDz2arF/view?usp=sharing>











## 2. Penerapan Konsep OOP

### 2.1. Inheritance & Polymorphism

*Inheritance* merupakan salah satu konsep dalam pemrograman berbasis objek yang memungkinkan suatu objek atau kelas memiliki kelas dan/atau objek turunan yang memiliki ciri/sifat yang diturunkan dari kelas/objek lainnya. Saat suatu kelas diturunkan dari kelas lain, kelas yang menjadi *child* (yang diturunkan) akan membawa *atribut*, *method*, dan *properties* lainnya yang dimiliki oleh kelas *parent* (kelas yang menurunkan).

#### 2.1.1. Effect

Konsep *inheritance* diterapkan melalui kelas Effect sebagai kelas dasar yang mendefinisikan properti dasar dan metode virtual. Kelas Effect sendiri digunakan untuk mendefinisikan objek yang nantinya akan mempengaruhi status yang dimiliki oleh Unit. Dengan menggunakan konsep *Inheritance*, efek yang berbeda dapat dibentuk dengan memiliki atribut dan metode yang sama namun dengan nilai yang berbeda.

```
#pragma once
#include <string>

#include "effects/Stats.hpp"

class Effect
{
public:
    Effect(const std::string& name, const std::string& description, int duration);
    virtual ~Effect() = default;

    // Prevent copying and assignment
    Effect(const Effect&) = delete;
```

```
Effect& operator=(const Effect&) = delete;
Effect(Effect&&) = default;
Effect& operator=(Effect&&) = default;

// --- Getters ---
const std::string& GetName() const;
const std::string& GetDescription() const;
int GetDuration() const;
int GetRemainingDuration() const;

// --- Setters ---
void SetName(const std::string& name);
void SetDescription(const std::string& description);
void SetDuration(int duration);
void SetRemainingDuration(int remainingDuration);

protected:
    std::string m_name;
    std::string m_description;
    int m_duration; // in turns
    int m_remainingDuration;
    Stats m_modifiers;
};
```

Kelas ini akan diturunkan menjadi beberapa kelas yaitu ArcanePrecision, BattleFocus, Bloodthirst, Brittle, Chronoflux, CurseOfFragility, EtherealForm, FlghterMight, FortuneCall, GladiatorMight, HarmoniousEmpowerment, IronCurtain,

MysticBarrier, Overclock, Shadowstep, Silence, StalwartResolve dan TimeWarp yang memiliki peningkatan status yang berbeda-beda. Contoh pada Kelas HarmoniousEmpowerment

```
class HarmoniousEmpowerment : public Effect
{
public:
    HarmoniousEmpowerment()
        : Effect("Harmonious Empowerment", "All Primary+3, Defenses+3, BuffMul×1.2", 3)
    {
        // Primary stats
        m_modifiers.strength = 3;
        m_modifiers.intelligence = 3;
        m_modifiers.agility = 3;

        // Defenses
        m_modifiers.physicalDefense = 3;
        m_modifiers.magicDefense = 3;

        // Enhanced buff multiplier
        m_modifiers.buffMultiplier = 1.2f;
    }

    HarmoniousEmpowerment(int primaryStatBonus, int defenseBonus, float buffMultBonus, int duration)
        : Effect("Harmonious Empowerment",
                  "All Primary+" + std::to_string(primaryStatBonus) + ", Defenses+" +

```

```
std::to_string(defenseBonus) + ", BuffMult" +
std::to_string(buffMultBonus),
duration)
{
    // Primary stats
    m_modifiers.strength      = primaryStatBonus;
    m_modifiers.intelligence  = primaryStatBonus;
    m_modifiers.agility        = primaryStatBonus;

    // Defenses
    m_modifiers.physicalDefense = defenseBonus;
    m_modifiers.magicDefense   = defenseBonus;

    // Enhanced buff multiplier (the main focus)
    m_modifiers.buffMultiplier = buffMultBonus;
}
};
```

Pada class ini, constructor memanggil constructor base dengan parameter khusus ("Harmonious Empowerment", deskripsi yang dibangun menggunakan nilai bonus, dan durasi) dan menginisialisasi modifier untuk *primary stats* (*strength*, *intelligence*, *agility*), *defense stats* (*physicalDefense*, *magicDefense*), serta *buff multiplier*.

Konsep *polymorphism* pada kelas ini digunakan pada berbagai kelas lainnya seperti *character*, *item*, dan *skill*. Berikut adalah contoh potongan kode yang mengimplementasi konsep tersebut.

```
void ApplyEffect(std::unique_ptr<Effect> effect);
void RemoveEffectByName(const std::string& effectName);
```

## 2.1.2. Unit

Konsep *inheritance* diterapkan melalui kelas Unit sebagai kelas dasar yang mendefinisikan properti dasar dan metode virtual. Kelas Unit digunakan untuk mendefinisikan objek yang nantinya akan mempengaruhi status tipe dari unit, yakni menjadi character atau mob. Dengan menggunakan konsep *Inheritance*, tipe unit yang berbeda dapat dibentuk dengan memiliki atribut dan metode yang sama namun dengan atribut yang berbeda.

```
class Unit
{
public:
    using ActionCompletionCallback = std::function<void()>

    Unit(const std::string& name);
    virtual ~Unit() = default;

    // Prevent copying and assignment
    Unit(const Unit&)           = delete;
    Unit& operator=(const Unit&) = delete;
    Unit(Unit&&)              = default;
```

```
Unit& operator=(Unit&&)      = default;

virtual void Attack(Unit&           target,
                     ActionCompletionCallback callback = nullptr,
                     ActionCompletionCallback onDeath = nullptr);
virtual void TakeDamage(int          damage,
                        ActionCompletionCallback callback = nullptr,
                        ActionCompletionCallback onDeath = nullptr);
virtual void Heal(int amount, ActionCompletionCallback callback = nullptr);

virtual void RestoreMana(int amount);

virtual int CalculateDamage(Unit& target);

/**
 * @brief Uses a skill, playing a generic skill animation.
 */
virtual bool UseSkill(Unit&           target,
                      ActionCompletionCallback callback = nullptr,
                      ActionCompletionCallback onDeath = nullptr) = 0;

virtual void AddEffect(std::unique_ptr<Effect> effect);
virtual void RemoveEffect(const std::string& effectName);

/**
 * @brief Resets unit state (health, mana, position, animation).
 */

```

```
virtual void Reset();

// --- Getters ---
unsigned int GetId() const;

const std::string& GetName() const;
bool IsActive() const;
int GetHealth() const;
int GetMaxHealth() const;
int GetCurrentMana() const;
int GetMaxMana() const;
int GetAttackDamage() const;
int GetHealthRegen() const;
int GetManaRegen() const;
Stats GetStats() const;
float GetMaxHealthMultiplier() const;
float GetHealthRegenMultiplier() const;
float GetMaxManaMultiplier() const;
float GetManaRegenMultiplier() const;
float GetAttackDamageMultiplier() const;
const SkillTree* GetSkillTree() const;

// --- Setters ---
void SetName(const std::string& name);
void SetActive(bool active);
virtual void SetMaxHealth(int maxHealth);
virtual void SetHealth(int health);
```

```
virtual void SetMaxMana(int maxMana);
virtual void SetCurrentMana(int currentMana);
virtual void SetAttackDamage(int attackDamage);
virtual void SetHealthRegen(int healthRegen);
virtual void SetManaRegen(int manaRegen);
void SetMaxHealthMultiplier(float maxHealthMultiplier);
void SetHealthRegenMultiplier(float healthRegenMultiplier);
void SetMaxManaMultiplier(float maxManaMultiplier);
void SetManaRegenMultiplier(float manaRegenMultiplier);
void SetAttackDamageMultiplier(float attackDamageMultiplier);
void SetSkillTree(std::unique_ptr<SkillTree> skillTree);
void SetStats(Stats stats);

void ApplyEffect(std::unique_ptr<Effect> effect);
void RemoveEffectByName(const std::string& effectName);

void RefreshTurn();

protected:
    unsigned int m_id;
    std::string m_name;
    bool m_active;

    // Base stats (will be overridden by level adjustments)
    int m_currentHealth = 100;
    int m_maxHealth     = 100;
    int m_healthRegen   = 0;
```

```
int m_currentMana    = 50;
int m_maxMana        = 50;
int m_manaRegen      = 0;
int m_attackDamage   = 10;

// Multipliers
float m_maxHealthMultiplier = 1.0f;
float m_healthRegenMultiplier = 1.0f;
float m_maxManaMultiplier = 1.0f;
float m_manaRegenMultiplier = 1.0f;
float m_attackDamageMultiplier = 1.0f;

std::unique_ptr<SkillTree>           m_skillTree = nullptr;
std::vector<std::unique_ptr<Effect>> m_activeEffects;
Stats                                m_stats;

private:
    static unsigned int s_nextId;
};
```

Contoh salah satu children dari kelas Unit (character):

```
class Character : virtual public Unit
{
```

```
public:  
    Character(const std::string& name);  
  
    ~Character() override = default;  
  
    // Prevent copying, allow moving  
    Character(const Character&) = delete;  
    Character& operator=(const Character&) = delete;  
    Character(Character&&) = default;  
    Character& operator=(Character&&) = default;  
  
    // --- RPG Stats Getters ---  
    int GetLevel() const;  
    int GetExp() const;  
    int GetGold() const;  
    int GetMastery() const;  
  
    virtual void SetLevel(int level);  
    virtual void SetExp(int exp);  
    virtual void SetGold(int gold);  
    virtual void SetMastery(int mastery);  
  
    // --- RPG Stats Modifiers ---  
    void AddExp(int amount);  
    void AddGold(int amount);  
    void AddMastery(int amount);
```

```

protected:
    /**
     * @brief Checks if enough EXP for level up and applies it.
     */
    virtual void CheckLevelUp();

    // --- RPG Stats ---
    int m_level    = 1;
    int m_exp      = 0;
    int m_gold     = 0;
    int m_mastery = 0;
};

```

Kelas Unit akan diturunkan menjadi 3 kelas, yakni kelas Character, kelas Mob, dan kelas Summon (kelas summon akan dipakai untuk character bertipe Necromancer). Selain itu, ketiga kelas turunan dari Unit juga akan diturunkan dengan multiple inheritance dengan memperhatikan diamond problem, salah satu contohnya adalah implementasi kelas Assasin, yang dapat dilihat pada cuplikan kode di bawah ini:

```

class Assassin : public Character, public AnimatedUnit
{
public:
    Assassin(const std::string& name,
              const sf::Vector2f& position,
              NavigationGrid& navGrid, // Pass grid to base

```

```
        bool           isPlayerControlled,
        const GameContext& gameContext);

~Assassin() override = default;

// Prevent copying, allow moving
Assassin(const Assassin&)          = delete;
Assassin& operator=(const Assassin&) = delete;
Assassin(Assassin&&)              = default;
Assassin& operator=(Assassin&&)    = default;

// --- Action Overrides ---
void Attack(Unit&                  target,
            ActionCompletionCallback callback = nullptr,
            ActionCompletionCallback onDeath   = nullptr) override;

bool UseSkill(Unit&                 target,
              ActionCompletionCallback callback = nullptr,
              ActionCompletionCallback onDeath   = nullptr) override;

void TakeDamage(int                  damage,
                ActionCompletionCallback callback = nullptr,
                ActionCompletionCallback onDeath   = nullptr) override;

void SetLevel(int level) override;

int CalculateDamage(Unit& target) override;
```

```
 /**
 * @brief Override RenderUI if fighter has specific visuals.
 */
// void RenderUI(sf::RenderWindow& window) override;

private:
/**
 * @brief Helper function to perform the attack animation and damage after range check.
 */
void PerformAttack(AnimatedUnit& target,
                    ActionCompletionCallback callback,
                    ActionCompletionCallback onDeath);

float m_criticalHitChance = 0.25f;
float m_criticalHitMultiplier = 2.0f;

float m_agilityMultiplier = 1.0f;

float m_attackRange = 48.0f;
};
```

Pada class Assasin, terdapat beberapa atribut baru yang tidak ada di kelas Character, sebagai contoh ada atribut m\_criticalHitChance untuk persentase critical hit, m\_agilityMultiplier untuk faktor pengali dari base stats agility, dll.

Pada kelas *Unit*, konsep abstract class juga diterapkan dan berperan sebagai kerangka dasar yang menentukan fungsionalitas dan interface umum yang harus dimiliki oleh setiap unit dalam game. Hal ini terlihat dari deklarasi fungsi *UseSkill()* yang ditetapkan sebagai virtual murni( ditandai dengan “= 0”).

```
/**
 * @brief Uses a skill, playing a generic skill animation.
 */
virtual void UseSkill(Unit& target,
                      ActionCompletionCallback callback = nullptr,
                      ActionCompletionCallback onDeath = nullptr) = 0;
```

Dengan adanya fungsi virtual murni ini, kelas *Unit* tidak dapat diinstansiasi secara langsung (menjadi objek konkret), melainkan harus diturunkan terlebih dahulu, di mana kelas turunan wajib mengimplementasikan fungsi *UseSkill()*. Pendekatan ini memastikan bahwa setiap unit dalam game memiliki kemampuan untuk menggunakan skill, namun implementasi spesifiknya diserahkan kepada kelas turunan seperti *Character*, *Enemy*, atau jenis unit lainnya

### 2.1.3. Skill

Konsep *inheritance* diterapkan melalui kelas *Skill* sebagai kelas dasar yang mendefinisikan properti dasar dan metode virtual. Kelas *Skill* sendiri digunakan untuk mendefinisikan objek yang nantinya akan mempengaruhi status yang dimiliki oleh *Skill*. Dengan menggunakan konsep *Inheritance*, efek yang berbeda dapat dibentuk dengan memiliki atribut dan metode yang sama namun dengan nilai yang berbeda.

```
#include "skill/Skill.hpp"
#include <iostream>
```

```
Skill::Skill(string name,
             float manaCost,
             int masteryCost,
             float damage,
             float effectChance,
             vector<unique_ptr<Effect>>&& effectVec,
             vector<unique_ptr<Skill>>&& treeNodeVec,
             bool learn,
             bool activate)
: name(name),
  manaCost(manaCost),
  masteryCost(masteryCost),
  damage(damage),
  effectChance(effectChance),
  effects(std::move(effectVec)),
  children(std::move(treeNodeVec)),
  isLearned(learn),
  activated(activate)
{ }

bool Skill::learn(int* masteryPoint)
{
    if(*masteryPoint >= this->getMasteryCost())
    {
        this -> isLearned = true;
        this -> activated = true;
    }
}
```

```

        *masteryPoint -= this->getMasteryCost();
        return true;
    }
    std::cout << "Skill Obj : MasteryPoint tidak cukup " << std::endl;
    return false;
}

```

Kelas ini diturunkan menjadi banyak *child*. Skill dapat dikelompokkan ke dua kelompok, yaitu *mob skill* dan *character skill*. *Mob skill* sendiri merupakan turunan langsung dari kelas *skill*. Sedangkan, *character skill* akan memiliki 5 turunan kelas, yaitu *AssassinSkill*, *BerserkerSkill*, *MageSkill*, *NecromancerSkill*, dan *FighterSkill*. Masing-masing kelas tersebut akan diturunkan lagi menjadi kelas baru yang banyak.

Berikut contoh dari *AssassinSkill*

```

class AssassinSkill : public Skill
{
protected:
    float agilityMultiplier;

public:
    AssassinSkill(string name,
                  float manaCost,
                  int masteryCost,
                  float damage,
                  float effectChance,

```

```
        vector<unique_ptr<Effect>>&& effectVec,
        vector<unique_ptr<Skill>>&& treeNodeVec,
        bool learn,
        bool activate,
        float agilMul)
: Skill(name,
        manaCost,
        masteryCost,
        damage,
        effectChance,
        std::move(effectVec),
        std::move(treeNodeVec),
        learn,
        activate)
{
    this->agilityMultiplier = agilMul;
};

virtual float getAgilityMultiplier() const = 0;
};
```

Lalu, AssassinSkill tersebut akan diturunkan ke kelas baru, contohnya ShadowArtistry berikut

```
class ShadowArtistry : public AssassinSkill
{
public:
```

```

ShadowArtistry(bool isLearned = false, bool isActivated = false)
    : AssassinSkill("Shadow Artistry", 6, 3, 0, 0.5f, {}, {}, isLearned, isActivated,
0.17f)
{
    vector<unique_ptr<Skill>> childSkills;
    childSkills.push_back(std::make_unique<Vanish>());
    childSkills.push_back(std::make_unique<PhantomStrike>());
    childSkills.push_back(std::make_unique<SwiftStrikes>());
    this->setChildren(std::move(childSkills));

    vector<unique_ptr<Effect>> effectVec;
    effectVec.push_back(std::make_unique<BattleFocus>(0.25f, 0.35f, 0.1f, 3));
    this->setEffects(std::move(effectVec));
}

float getAgilityMultiplier() const override { return this->agilityMultiplier; }
};

```

## 2.2. Method/Operator Overloading

*Method Overloading* adalah suatu konsep OOP pada bahasa pemrograman C++ yang memungkinkan suatu fungsi memiliki nama sama namun memiliki *signature* yang berbeda. Pada Tugas Besar ini, digunakan beberapa *method overloading* pada beberapa kelas:

### 2.2.1. Item

Pada implementasi Item, operator == digunakan untuk mengecek apakah dua item yang dibandingkan sama, dan operator != digunakan untuk mengecek apakah dua item yang dibandingkan berbeda.

```
bool Item::operator==(const Item& other) const
{
    return itemID == other.itemID && name == other.name && type == other.type &&
           rarity == other.rarity && effects == other.effects;
}

bool Item::operator!=(const Item& other) const
{
    return itemID != other.itemID || name != other.name || type != other.type ||
           rarity != other.rarity || effects != other.effects;
}
```

### 2.2.2. Stats

Pada implementasi Stats, operator + digunakan untuk menambah dua stats yang berbeda tanpa mengubah state dari objek tersebut, operator - digunakan untuk mengurangi dua stats yang berbeda tanpa mengubah state dari objek tersebut, operator += digunakan untuk menambah stats yang lain ke stats saat ini sehingga mengubah stats yang saat ini, operator -= digunakan untuk mengurangi stats yang lain ke stats saat ini sehingga mengubah stats yang saat ini.

```
#include "effects/Stats.hpp"
```

```
Stats Stats::operator+(const Stats& other) const
{
    Stats result = *this;
    result += other;
    return result;
}

Stats Stats::operator-(const Stats& other) const
{
    Stats result = *this;
    result -= other;
    return result;
}

Stats& Stats::operator+=(const Stats& other)
{
    // additive stats
    strength += other.strength;
    intelligence += other.intelligence;
    agility += other.agility;
    luck += other.luck;
    physicalDefense += other.physicalDefense;
    magicDefense += other.magicDefense;

    // additive chances/resistances
    criticalStrikeChance += other.criticalStrikeChance;
    skipTurnChance += other.skipTurnChance;
```

```
dodgeChance += other.dodgeChance;
statusResistance += other.statusResistance;

// multiplicative stats
buffMultiplier *= other.buffMultiplier;
criticalStrikeMultiplier *= other.criticalStrikeMultiplier;
accuracy *= other.accuracy;
hasteMultiplier *= other.hasteMultiplier;
resourceCostMul *= other.resourceCostMul;

return *this;
}

Stats& Stats::operator=(const Stats& other)
{
    // subtract additive stats
    strength -= other.strength;
    intelligence -= other.intelligence;
    agility -= other.agility;
    luck -= other.luck;
    physicalDefense -= other.physicalDefense;
    magicDefense -= other.magicDefense;

    // subtract additive chances/resistances
    criticalStrikeChance -= other.criticalStrikeChance;
    skipTurnChance -= other.skipTurnChance;
    dodgeChance -= other.dodgeChance;
```

```

statusResistance -= other.statusResistance;

// divide multiplicative stats
if (other.buffMultiplier != 0.0f)
    buffMultiplier /= other.buffMultiplier;
if (other.criticalStrikeMultiplier != 0.0f)
    criticalStrikeMultiplier /= other.criticalStrikeMultiplier;
if (other.accuracy != 0.0f)
    accuracy /= other.accuracy;
if (other.hasteMultiplier != 0.0f)
    hasteMultiplier /= other.hasteMultiplier;
if (other.resourceCostMul != 0.0f)
    resourceCostMul /= other.resourceCostMul;

return *this;
}

```

### 2.2.3. RNG (Random Number Generator)

Implementasi rng diimplementasikan dengan menggunakan algoritma Permuted Congurential Generator (PCG) dengan memanfaatkan bit-shifting dan juga konsep static pada Object Oriented Programming untuk terus mengubah state dan seednya selama program sedang berjalan. Pada kelas ini, terdapat operator = yang digunakan untuk melakukan copy pada state dari rng, yang dapat dilihat pada cuplikan kode di bawah.

```
RNG& RNG::operator=(const RNG& other)
```

```
{
    if (this != &other)
    {
        this->state = other.state;
    }
    return *this;
}
```

## 2.3. Template & Generic Classes

Generic Class adalah suatu fitur yang memungkinkan berbagai jenis tipe data menjadi parameter untuk suatu kelas ataupun fungsi. Penggunaan Generic Class dapat meningkatkan efisiensi kode dan mengurangi repetisi kode untuk tipe data yang berbeda. Implementasi Generic Class pada C++ adalah dengan menggunakan Template Class. Pada Tugas Besar ini, konsep template dan generic class diimplementasikan dengan sangat baik dalam kelas *UnitManager* untuk menangani *polymorphism* secara efisien dan *type-safe*.

*UnitManager* berperan sebagai sistem pengelolaan unit dalam game, yang memungkinkan penyimpanan, pengolahan, dan pengambilan berbagai jenis unit dengan memanfaatkan inheritance hierarki yang dimulai dari kelas dasar *Unit*. Fitur utama dari *UnitManager* yang menerapkan konsep template adalah dua metode templated: *GetUnitOfType<T>()* dan *GetAllUnitsOfType<T>()*, yang memungkinkan pengaksesan unit tertentu berdasarkan tipenya yang spesifik. Berikut adalah potongan dari file *UnitManager.hpp*

```
// --- Unit retrieval with type casting ---
/**
 * @brief Get a pointer of a specific derived type by ID
 * @tparam T The derived type of the unit to retrieve (e.g., Fighter, Character)
```

```

* @param id The ID of the unit to retrieve
* @return T* Pointer to the unit cast to type T, or nullptr if not found or type mismatch.
*/
template <typename T>
T* GetUnitOfType(unsigned int id);

/**
 * @brief Get all units of a specific derived type
 * @tparam T The derived type of the units to retrieve
 * @return std::vector<T*> Vector of raw pointers to units of the specified type.
*/
template <typename T>
std::vector<T*> GetAllUnitsOfType();

```

Metode ini menggunakan parameter *template T* yang memungkinkan client code melakukan *query* unit dengan tipe yang spesifik. Alih-alih memberikan pointer *Unit\** biasa yang kemudian perlu di-cast manual (yang rawan kesalahan), metode ini sudah melakukan pengecekan tipe dan mengembalikan pointer yang sudah bertipe sesuai. Jika unit dengan ID tersebut ditemukan tetapi bukan bertipe T, maka *dynamic\_cast* akan mengembalikan *nullptr*.

## 2.4. Exception

*Exception* adalah tipe data khusus yang mengembalikan suatu nilai saat terjadi kesalahan dalam proses *run program*. *Exception* akan dikembalikan dengan suatu keyword “*throw*” berdasarkan suatu alur yang sudah didefinisikan, dan saat terjadi suatu kesalahan yang sesuai dengan definisi *Exception* tersebut, suatu objek *Exception* yang dilempar tersebut harus ditangkap (*catch*) agar proses *run program* tidak diberhentikan karena *error*. Hal ini disebut sebagai *error handling*. Pada Tugas Besar ini, *Exception* yang dibuat, antara lain:

### 2.4.1. Inventory (Backpack dan Equipment)

Exception yang dibuat untuk membuat *error handling* pada penggunaan kelas Backpack dan Equipment.

```
// Exception ketika tidak ada item pada slot yang diminta
class EmptyCellException : public std::exception
{
public:
    const char* what() const noexcept override { return "No item at the specified position"; }
};

// Exception ketika backpack penuh dan tidak bisa menampung item baru
class BackpackOvercapacityException : public std::exception
{
public:
    const char* what() const noexcept override { return "Backpack is at full capacity"; }
};

// Exception ketika kuantitas item yang diminta lebih banyak dari total yang ada di backpack
class InsufficientQuantityException : public std::exception
{
public:
    const char* what() const noexcept override
    {
        return "Not enough items of this type in backpack";
    }
};
```

```
// Exception ketika slot yang diminta sudah terisi item lain
class ItemSlotOccupiedException : public std::exception
{
public:
    const char* what() const noexcept override
    {
        return "Cannot add a different item to an occupied slot";
    }
};

// Exception ketika item tidak sesuai dengan tipe slot yang diminta
class InvalidEquipmentTypeException : public std::exception
{
private:
    std::string message;

public:
    InvalidEquipmentTypeException(const std::string& msg) : message(msg) {}
    const char* what() const noexcept override { return message.c_str(); }
};
```

#### 2.4.2. Shop

Exception yang dibuat untuk membuat *error handling* pada penggunaan kelas Shop, dengan contoh method buy.

```
void Shop::buy(Character& player, Backpack& backpack, Item& item, int totalPrice, int
                quantity)
```

```
{  
  
    if (player.GetGold() < totalPrice)  
        throw InsufficientGoldException();  
  
    player.AddGold(-totalPrice);  
  
    try  
    {  
        backpack.addItem(item, quantity);  
    }  
    catch (const std::exception&)  
    {  
        player.AddGold(totalPrice);  
        throw;  
    }  
  
    auto it = std::find_if(allStock.begin(), allStock.end(), [&](const StockEntry& entry) {  
        return std::get<0>(entry) == item;  
    });  
  
    if (it != allStock.end())  
    {  
        int& currentQty = std::get<2>(*it);  
  
        currentQty -= quantity;  
    }  
}
```

```

        if (currentQty <= 0)
        {
            allStock.erase(it);
        }
    }
}

```

## 2.5. C++ Standard Template Library

Tugas Besar ini juga menggunakan beberapa Standard Template Library (STL) lain untuk menggunakan tipe data khusus yang memudahkan operasi fungsi dan penyimpanan. Beberapa STL yang digunakan pada Tugas Besar ini adalah:

### 2.5.1. `std::vector`

`std::vector` merupakan tipe STL yang merepresentasikan array dinamis pada c++. Seperti pada array biasa, pengaksesan elemen pada vektor juga menggunakan indeks. `std::vector` memungkinkan berbagai tipe data untuk digunakan sebagai *array*, sehingga penggunaan `std::vector` sebagai tipe data untuk penyimpanan pada *Inventory*, menyimpan data *config* dari *file*, menyimpan kumpulan *Unit*, *Effect*, dan *Skill* pada *Game*. Penggunaan `std::vector` sebagai *array* dinamis diperlukan pada kasus-kasus di atas karena jumlah elemen yang ada pada kumpulan data tersebut dapat berubah-ubah (dinamis) seiring keberjalanannya program.

```

void Dungeon::clearChamber(int index)
{
    if (index < 0 || index >= chambers.size())
    {
        return;
    }
}

```

```
Chamber& chamber = chambers[index];

if (chamber.getIsCleared())
{
    return;
}

vector<Item> chamberLoot = chamber.clearChamber();

for (const auto& item : chamberLoot)
{
    mobLoot.push_back(item);
}

if (areAllChambersCleared())
{
    isCleared = true;
}
```

## 2.5.2. `std::map`

Pada C++, `std::map` adalah salah satu kontainer asosiatif di STL yang menyimpan pasangan *key-value* (mirip kamus/dictionary). Berbeda dengan `std::vector` yang diakses lewat indeks numerik, `std::map` mengasosiasikan setiap nilai (`value`) dengan sebuah kunci unik (`key`), dan secara internal mengurutkan elemen-elemennya berdasarkan urutan kunci (biasanya dengan struktur data pohon merah-hitam)

```
    std::map<std::string /*dungeon rank*/,
              std::vector<std::tuple<std::string /*quest type*/,
                                         int /*damage dealt/mobs killed*/,
                                         int /*gold reward*/,
                                         int /*exp reward*/,
                                         std::string /*item reward (ID)*/>>>
    m_questData;
```

Sebagai struktur data database quest

```
private:
    std::map<std::string, std::map<std::string, float>> m_data;
};
```

Sebagai struktur data mob loot di kelas MobLootConfigParser

```
private:
    std::vector<std::vector<std::string>> m_equipmentData;
    std::vector<std::pair<Item, int>> m_backpackData;
    std::map<std::string, std::string> m_charstats;
    std::map<std::string, std::string> m_unitstats;
    std::set<std::string> m_skilltree;
    ItemManager& m_itemManager; // Item database for backpack items
```

```
};
```

Sebagai struktur data stats di kelas PlayerConfigParser

### 2.5.3. std::set

std::set adalah kontainer asosiatif di C++ STL yang menyimpan sekumpulan elemen unik (keys) dalam urutan terurut (menaik secara default). Internally, std::set biasanya diimplementasikan sebagai pohon merah-hitam (red-black tree), sehingga setiap operasi penting dijamin berk kompleksitas O(log n).

```
private:

    std::vector<std::vector<std::string>> m_equipmentData;

    std::vector<std::pair<Item, int>> m_backpackData;

    std::map<std::string, std::string> m_charstats;

    std::map<std::string, std::string> m_unitstats;

    std::set<std::string> m_skilltree;

    ItemManager& m_itemManager; // Item database for backpack items

};
```

Sebagai elemen penyimpanan data skill tree di kelas PlayerConfigParser

#### 2.5.4. std::pair

std::pair adalah kelas template sederhana di C++ STL yang menyimpan dua objek berpasangan—sering disebut first dan second. Ia berguna untuk mengembalikan dua nilai dari fungsi, menyimpan pasangan kunci-nilai (misalnya di dalam std::map), atau mengelompokkan dua data yang berkaitan.

```
protected:
    const int                         backpackRows;
    const int                         backpackCols;
    const int                         maxStackSize = 64;
    std::vector<std::vector<std::pair<Item, int>>> backpack;
```

Sebagai elemen penyusun matriks di kelas Backpack

```
private:
    std::vector<std::vector<std::string>> m_equipmentData;
    std::vector<std::pair<Item, int>>      m_backpackData;
    std::map<std::string, std::string>       m_charstats;
    std::map<std::string, std::string>       m_unitstats;
    std::set<std::string>                   m_skilltree;
    ItemManager&                          m_itemManager; // Item database for backpack items
};
```

Sebagai elemen penyusun vektor data Bacpack di kelas PlayerConfigParser

### 2.5.5. std::tuple

`std::tuple` adalah kelas template yang disediakan oleh pustaka standar C++ (`<tuple>`) untuk menyimpan sejumlah elemen heterogen dalam satu objek terstruktur. Berbeda dengan `std::pair`, yang membatasi diri pada dua elemen, `std::tuple` dapat menampung N elemen dengan tipe data yang berbeda-beda, di mana N dapat sama dengan nol atau lebih.

```
class Shop
{
public:
    using StockEntry = std::tuple<Item, int /*price*/, int /*qty*/>;
```

Sebagai struktur data entri stok (StockEntry) matriks di kelas Shop

### 2.5.6. std::string

`std::string` adalah kelas yang disediakan oleh pustaka standar C++ (`<string>`) untuk merepresentasikan urutan karakter secara dinamis. Dibangun di atas `std::basic_string<char>`, `std::string` memberikan antarmuka yang kaya untuk manipulasi teks tanpa harus mengelola memori mentah secara manual. Dalam proyek ini, `std::string` digunakan secara ekstensif untuk berbagai keperluan.

```
class Item
{
public:
    Item();
    Item(std::string itemID,
          const std::string& name,
          const std::string& type,
          char rarity,
          const std::vector<std::shared_ptr<Effect>> effects,
```

<pre>const std::string&amp;</pre>	<pre>description);</pre>
-----------------------------------	--------------------------

### 2.5.7. `std::unique_ptr`

`std::unique_ptr` adalah kelas di pustaka standar C++ yang mengelola kepemilikan unik terhadap objek di heap. Artinya, hanya ada satu pointer (pemilik) yang bisa memiliki objek tertentu. Jika `std::unique_ptr` keluar dari scope atau dihapus, objek yang dikelolanya juga akan dihapus secara otomatis. Hal ini membantu mencegah kebocoran memori dan membuat manajemen resource menjadi lebih aman dan otomatis. Perlu diingat bahwa `std::unique_ptr` tidak dapat disalin, tetapi bisa dipindahkan.

<pre>class Skill { private:     string name;     float manaCost;     int masteryCost;     float damage;     float effectChance;     vector&lt;unique_ptr&lt;Effect&gt;&gt; effects;     vector&lt;unique_ptr&lt;Skill&gt;&gt; children;     bool isLearned;     bool activated;</pre>
---

### 2.5.8. `std::queue`

`std::queue` adalah adaptor kontainer di pustaka standar C++ untuk menyajikan antrian (queue) dengan prinsip FIFO (First In, First Out), artinya elemen yang pertama kali ditambahkan adalah yang pertama diambil. Mirip dengan `std::stack`, `std::queue` biasanya menggunakan `std::deque` sebagai underlying container secara default.

```
BossHealthBar m_bossHealthBar;
std::queue<unsigned int> m_turnQueue;
```

### 2.5.9. std::function

std::function adalah kelas template di pustaka standar C++ yang berfungsi sebagai pembungkus (wrapper) untuk objek callable (seperti fungsi, lambda, atau fungsi anggota). Dengan std::function, kita dapat menyimpan, menyalin, dan memanggil objek callable secara generik, sehingga sangat berguna untuk callback atau fungsi yang dikonfigurasi secara dinamis.

```
public:
    using ActionCompletionCallback = std::function<void()>;
```

## 2.6. Konsep OOP lain

Sama seperti poin-poin sebelumnya, tuliskanlah konsep OOP lainnya yang ada terapkan selain dari 5 yang diwajibkan. Poin ini akan dipertimbangkan untuk menjadi **nilai bonus**. Contoh: menerapkan Abstract Base Class, Aggregation, atau Composition.

### 2.6.1. Abstract Base Class

Pada program ini, abstract base class digunakan pada kelas Unit, yang telah dijelaskan pada [poin 2.1.2](#)

### 2.6.2. Dungeon

Pada Implementasi dungeon, dungeon memiliki hubungan Composition dengan Quest, list of Chamber, dan list of Item. Dungeon juga memiliki hubungan Aggregation yakni dengan ItemManager dan MobLootConfigParser. Hubungan composition Dungeon terjadi karena list of chamber, quest, dan list of item merupakan “part-of” dari dungeon, sehingga jika dungeon

hancur, maka yang memiliki hubungan dengannya juga hancur. Hal ini berbeda dengan ItemManager dan MobLootConfigParser, meskipun dungeon hancur, tetapi objek MobLootConfigParser dan ItemManager tetap ada dan tidak hancur. Untuk hubungannya, dapat dilihat pada cuplikan kode di bawah:

```
class Dungeon
{
private:
    string rank; // The rank of the dungeon (S, A, B, C, D, E, SPECIAL)
    vector<Chamber> chambers; // Chambers in the dungeon
    int entranceFee; // Gold cost to enter the dungeon
    int minPlayerLevel; // Minimum player level required to enter
    int playerLevel; // Current player level
    int initGold; // Gold obtained from the dungeon
    int initExp; // Experience obtained from the dungeon
    bool isCleared; // Whether the dungeon has been cleared
    vector<Item> dungeonLoot; // Items obtained from completing the dungeon (completion reward)
    vector<Item> mobLoot; // Items obtained from mobs in cleared chambers
    double expMultiplier; // Experience multiplier for this dungeon
    double goldMultiplier; // Gold multiplier for this dungeon
    ItemManager& itemManager; // Pointer to the ItemManager
    MobLootConfigParser& mobLootConfigParser; // Pointer to the MobLootConfigParser
    Quest quest; // Quest associated with this dungeon
```

### 3. Bonus Yang dikerjakan

Hapuslah poin di bawah jika tidak dikerjakan. Jika dikerjakan, jelaskanlah pendekatan yang anda gunakan untuk menyelesaikan masalah tersebut berikut dengan screenshot cuplikan kode yang relevan untuk menjelaskan pendekatan anda!

#### 3.1. Bonus yang diusulkan oleh spek

##### 3.1.1. Quest System

Quest adalah fitur tambahan yang ada pada dungeon. Dalam implementasinya, sebuah dungeon hanya memiliki sebuah quest, dan jenis quest bergantung pada konfigurasi yang didefinisikan pada file bernama “quest.txt”. Jika ada quest dengan level yang sama, maka pembentukan quest akan dilakukan secara acak sesuai dengan levelnya tersebut. Sebagai contoh, jika ada quest level S kill 50 mob, dan quest level S damage 1000, maka kedua quest itu akan diacak dan akan dimasukkan pada dungeon dengan rank yang sama.

Pendekatan penyelesaian masalah dimulai dari load quest dari file menggunakan parser quest (“quest.txt”), parser quest didefinisikan di dalam folder dengan header /include/parser/QuestConfigParser dan implementasi yang ada pada /src/parser/QuestConfigParser. Berikut adalah tipe data yang digunakan parser untuk menyimpan quest dan juga bentuk dari quest:

```
QuestType type;           // Type of quest (KILL or DAMAGE)
string    dungeonRank;    // Rank of dungeon where this quest is available
int      targetCount;    // Required kills or damage to complete
int      currentProgress; // Current kills or damage accumulated
int      goldReward;     // Gold reward for completing the quest
int      expReward;      // Experience reward for completing the quest
string    itemRewardId;   // ID of the item reward
bool     isCompleted;    // Whether the quest is completed
```

```
std::map<std::string /*dungeon rank*/,
         std::vector<std::tuple<std::string /*quest type*/,
                               int /*damage dealt/mobs killed*/,
                               int /*gold reward*/,
                               int /*exp reward*/,
                               std::string /*item reward (ID)*/>>>
```

Kemudian, data dari parser akan dilempar ke kelas yang bernama QuestGenerator pada file QuestGenerator.cpp (pada folder quest). Proses pelemparan dilakukan pada file MainMenuState.cpp (file ada pada /src/states/MainMenuState.cpp, yakni pada cuplikan kode berikut:

```
{
    QuestConfigParser parser;
    try
    {
        parser.ParseFromFile(folderPath);
        GetContext().GetQuestGenerator() -> loadQuestData(parser.GetQuestData());
        std::cout << "QuestConfigParser done" << std::endl;
    }
    catch (const std::exception& e)
    {
        showError("QuestConfigParser error: " + std::string(e.what()));
        return;
```

```
    }
    catch (const char* msg)
    {
        showError(std::string("QuestConfigParser error: ") + (msg ? msg : "(null
message)"));
        return;
    }
    catch (...)
    {
        showError("QuestConfigParser unknown error");
        return;
    }
}
```

Setelah pelemparan data ke kelas QuestGenerator (yakni menggunakan method loadQuestData), reference dari objek QuestGenerator akan dimasukkan dan dipakai oleh kelas DungeonFactory (pada folder DungeonFactory.cpp). Oleh karena itu, kelas DungeonFactory memiliki reference dari objek QuestGenerator dan bisa memakai method yang ada pada objek QuestGenerator. Pembuatan quest pada dungeon ada pada method createDungeon yang ada pada kelas DungeonFactory, yakni pada bagian berikut:

```
if (hasQuestsForRank(rank))
{
    Quest quest = questGenerator.generateQuest(rank);
    dungeon.setQuest(quest);
}
```

Method createDungeon akan mengecek apakah ada quest yang valid pada dungeon dengan rank yang didefinisikan, pengecekan ini dilakukan dengan method hasQuestsForRank yang dimiliki oleh kelas DungeonFactory dengan implementasi sebagai berikut:

```
bool DungeonFactory::hasQuestsForRank(const string& rank) const
{
    return questGenerator.hasQuestsForRank(rank);
}
```

Jika ada quest dengan rank yang sama dengan rank dungeon, maka quest akan dibentuk dan dimasukkan pada dungeon, sehingga dungeon berhasil memiliki sebuah atribut dengan tipe objek Quest. Setelah quest berhasil didefinisikan pada sebuah dungeon, reward bisa diambil dari quest dengan menggunakan method yang ada di kelas dungeon, yakni method getQuestGoldReward, getQuestExpReward, getQuestItemRewardID, dan getQuestRewardItem dengan cuplikan kode sebagai berikut:

```
int Dungeon::getQuestGoldReward() const
{
    if (isQuestCompleted())
    {
        return quest.getGoldReward();
    }
    return 0;
```

```
}

int Dungeon::getQuestExpReward() const
{
    if (isQuestCompleted())
    {
        return quest.getExpReward();
    }
    return 0;
}

string Dungeon::getQuestItemRewardId() const
{
    if (isQuestCompleted())
    {
        return quest.getItemRewardId();
    }
    return "";
}

Item Dungeon::getQuestRewardItem() const
{
    if (isQuestCompleted())
    {
        try
        {
            return itemManager.getItem(getQuestItemRewardId());
        }
    }
}
```

```
    }
    catch (const char* error)
    {
        return Item();
    }
}
return Item();
```

Method di atas akan dipanggil oleh battle state jika telah menyelesaikan sebuah dungeon, selain itu battle state juga dapat melakukan update progress (seperti sudah berapa kill atau damage) dengan cara memanggil method updateDamageQuestProgress atau updateKillQuestProgress, dan juga memanggil method pengecek apakah suatu quest telah selesai atau belum dengan menggunakan method isQuestCompleted, yang dapat dilihat pada cuplikan di bawah ini:

```
bool Dungeon::isQuestCompleted() const
{
    return quest.getIsCompleted() && areAllChambersCleared();
}

bool Dungeon::updateKillQuestProgress(int mobsKilled)
{
    return quest.updateKillProgress(mobsKilled);
}
```

```
bool Dungeon::updateDamageQuestProgress(int damageDealt)
{
    return quest.updateDamageProgress(damageDealt);
}
```

### 3.1.2. Cheat

Cheat System adalah sebuah *interface* yang menyediakan *command list* untuk memanipulasi program sehingga pemain bisa *grinding* menjadi Sung Jin Woo dengan *levelling* special chamber dengan instant. Beberapa *cheat* yang diterapkan adalah sebagai berikut.

Perintah	Deskripsi
finish_quest	Menyelesaikan progres dari <i>quest</i> yang sedang berlangsung
finish_chamber	Menyelesaikan chamber saat ini (membunuh semua mob)
finish_dungeon	Menyelesaikan dungeon yang sedang dilalui, bukan hanya “skip” chamber tetapi seolah melalui semua <i>chamber</i> dan membunuh semua mob sehingga loot dan exp yang didapatkan sama seperti jika tanpa menggunakan <i>cheat</i>
hesoyam	Darah dan mana menjadi penuh serta memberikan 1000 Gold dan 1000 Exp
god_mode <on/off>	Meningkatkan <i>dodge chance</i> menjadi 100% serta <i>damage</i> menjadi 10000x lipat lebih sakit sehingga pemain menjadi se-OP Sung Jin-Katou (Sung Jin-Woo di kelompok kami dengan NIM 13523090)
reset	Me-reset semua efek yang berlaku pada karakter juga mengembalikan HP dan Mana menjadi penuh

add_exp <value>	Menambahkan <i>experience</i> sebesar <i>value</i> yang diberikan, jika tidak maka <i>default</i> 1000 <i>experience</i> akan ditambahkan
add_gold <value>	Menambahkan <i>gold</i> sebesar <i>value</i> yang diberikan, jika tidak maka <i>default</i> 1000 <i>gold</i> akan ditambahkan
clear	Karena menu <i>cheat</i> pada program berbentuk <i>console</i> (terinspirasi dari Counter Strike 1.6) kami melakukan <i>logging</i> dari perintah yang dijalankan. Perintah ini membersihkan <i>logging</i> tersebut
help	Memunculkan tabel perintah dengan deskripsinya.

Pendekatan yang digunakan adalah dengan membuat sebuah *class* yang menampung perintah-perintah sebagai *callback* sehingga pembuatan *cheat* sangat fleksibel dan mudah.

```
void CheatConsole::registerCommand(const std::string& command,
                                    CheatCommandCallback callback,
                                    const std::string& description)
{
    CommandInfo info;
    info.callback = callback;
    info.description = description;
    m_commands[command] = info;
}
```

```
// hesoyam
m_cheatConsole.registerCommand(
    "hesoyam",
    [this] (const std::vector<std::string>& args) -> std::string {
        if (m_character)
        {
            Character* character =
                GetContext().GetUnitManager()->GetUnitOfType<Character>(m_character->GetId());
            if (character)
            {
                character->SetHealth(character->GetMaxHealth());
                character->SetCurrentMana(character->GetMaxMana());
                character->AddGold(1000);
                character->AddExp(1000);
                return "HESOYAM activated";
            }
            else
            {
                return "Character not found";
            }
        }
        return "Character not initialized";
},
"Full health and mana with extra bonus wink wink");
```

### 3.1.3. Graphical User Interface (GUI)

Untuk GUI, kami menggunakan tools ImGui dan SFML namun untuk skala program yang besar akan menjadi masalah dalam mengatur dan mengorkestrasikan interaksi, proses, dan tampilan. Untuk itu, kami mengusulkan untuk menggunakan pendekatan *Process Events, Update, and Render*. Di dalam proses tersebut, kami juga menambah abstraksi lain yaitu memecah objek-objek menjadi kumpulan yang lebih spesifik seperti unit, state, dan gui. Unit adalah tampilan karakter teranimasi (yang merupakan turunan dari AnimatedUnit). State adalah tampilan “menu” yang menampung seluruh objek yang akan ditampilkan pada satu waktu. Gui adalah komponen-komponen UI dari ImGui. Urutan penampiliannya adalah state, unit, lalu gui dan masing-masing memiliki pendekatan *Process Events, Update, and Render* nya sendiri.

State menggunakan struktur data *stack*, sehingga memungkinkan untuk melakukan *stacking* state dan proses pergantian “menu” menjadi lebih mudah karena pemrogram hanya perlu memikirkan apa menu yang ingin ditampilkan saat ini (item paling atas) dan menu yang diganti ter-handle dengan baik (penggunaan *push*, *pop*, dan *replace*). Selain itu, saat proses pergantian state, terdapat pemanggilan *pause*, *resume*, dan *exit* yang diatur otomatis oleh *StateManager*.

```
void Game::Run()
{
    while (m_window.isOpen() && !m_stateManager.IsEmpty())
    {
        sf::Time dt = m_gameClock.restart();

        m_context.UpdateFPS(dt);

        ProcessEvents();
        Update(dt);
    }
}
```

```
        Render();
    }

}

void Game::ProcessEvents()
{
    while (const std::optional event = m_window.pollEvent())
    {
        m_stateManager.ProcessEvent(*event);
        m_unitManager.ProcessEvent(*event);
        m_gui.ProcessEvent(*event, m_window);

        // Close window: exit
        if (event->is<sf::Event::Closed>())
        {
            m_window.close();
        }
    }
}

void Game::Update(const sf::Time& dt)
{
    m_gui.NewFrame();
    m_stateManager.Update(dt);
    m_unitManager.Update(dt);
}
```

```
void Game::Render()
{
    m_window.clear();

    // Draw game states
    m_stateManager.Draw(m_window);
    m_unitManager.Draw(m_window);

    // Render ImGui UI
    m_stateManager.RenderUI();
    m_unitManager.DrawUI(m_window);
    m_gui.Render(m_window);

    m_window.display();
}
```

```
void StateManager::PushState(std::unique_ptr<State> state)
{
    if (!m_states.empty())
    {
        m_states.back()->Pause();
    }

    m_states.push_back(std::move(state));
    m_states.back()->Init();
}
```

```
}

void StateManager::PopState()
{
    if (!m_states.empty())
    {
        m_states.back() ->Exit();
        m_states.pop_back();

        if (!m_states.empty())
        {
            m_states.back() ->Resume();
        }
    }
}

void StateManager::ChangeState(std::unique_ptr<State> state)
{
    if (!m_states.empty())
    {
        // Call Exit on the state being replaced
        m_states.back() ->Exit();
        m_states.pop_back();
    }

    // Push and initialize the new state
    m_states.push_back(std::move(state));
}
```

```
m_states.back()->Init();  
}  
  
void StateManager::ProcessStateChanges()  
{  
    for (auto& change : m_pendingChanges)  
    {  
        ApplyStateChange(change);  
    }  
  
    m_pendingChanges.clear();  
}  
  
void StateManager::ApplyStateChange(State::StateChange& change)  
{  
    switch (change.GetAction())  
    {  
        case State::StateAction::PUSH:  
#ifdef DEBUG_MODE  
            PrintStateStack();  
#endif  
            if (change.GetNextState())  
            {  
                PushState(change.TakeNextState());  
            }  
#ifdef DEBUG_MODE  
            PrintStateStack();  
    }  
}
```

```
#endif  
        break;  
  
        case State::StateAction::POP:  
#ifdef DEBUG_MODE  
        PrintStateStack();  
#endif  
  
        PopState();  
#ifdef DEBUG_MODE  
        PrintStateStack();  
#endif  
        break;  
  
        case State::StateAction::REPLACE:  
#ifdef DEBUG_MODE  
        PrintStateStack();  
#endif  
        if (change.GetNextState())  
        {  
            ChangeState(change.TakeNextState());  
        }  
#ifdef DEBUG_MODE  
        PrintStateStack();  
#endif  
        break;
```

```
case State::StateAction::NONE:
    // No action needed
    break;
}

void StateManager::ProcessEvent(const sf::Event& event)
{
    if (m_states.empty())
        return;

    // Create a collection of states to process events
    std::vector<State*> activeStates = GetActiveStateStack();

    // Process input from top to bottom until a state handles it
    for (auto it = activeStates.begin(); it != activeStates.end(); ++it)
    {
        State* state = *it;

        // Get state change request from event handling
        State::StateChange change = state->ProcessEvent(event);

        // If there's a requested state change, queue it
        if (change.GetAction() != State::StateAction::NONE)
        {
            m_pendingChanges.push_back(std::move(change));
            break; // Stop processing input
        }
    }
}
```

```
    }

    // If state doesn't allow input propagation, stop here
    if (!state->AllowsInputPassthrough())
    {
        break;
    }
}

void StateManager::Update(const sf::Time& dt)
{
    if (m_states.empty())
        return;

    // Get active state stack
    std::vector<State*> activeStates = GetActiveStateStack();

    // Update states from top to bottom
    for (auto it = activeStates.begin(); it != activeStates.end(); ++it)
    {
        State* state = *it;

        // Update the state
        State::StateChange change = state->Update(dt);

        // Queue any requested state changes
    }
}
```

```
    if (change.GetAction() != State::StateAction::NONE)
    {
        m_pendingChanges.push_back(std::move(change));
    }

    // If state doesn't allow updates to pass through, stop here
    if (!state->AllowsUpdatePassthrough())
    {
        break;
    }
}

// Process any pending state changes
ProcessStateChanges();
}

void StateManager::Draw(sf::RenderWindow& window)
{
    if (m_states.empty())
        return;

    // Create a vector of states to render
    std::vector<State*> visibleStates;

    // Collect states that should be rendered
    // Start from the top of the stack
    std::vector<State*> stateStack = GetStateStack();
```

```
for (auto it = stateStack.begin(); it != stateStack.end(); ++it)
{
    State* state = *it;
    visibleStates.push_back(state);

    // If state is not transparent, don't render states beneath it
    if (!state->IsTransparent())
    {
        break;
    }
}

// Draw states from bottom to top
for (auto it = visibleStates.rbegin(); it != visibleStates.rend(); ++it)
{
    (*it)->Draw(window);
}

void StateManager::RenderUI()
{
    if (m_states.empty())
        return;

    // Create a vector of states that should render UI
    std::vector<State*> uiStates;
```

```
// Get states with visible UI
std::vector<State*> stateStack = GetStateStack();

for (auto it = stateStack.begin(); it != stateStack.end(); ++it)
{
    State* state = *it;
    uiStates.push_back(state);

    // If state doesn't have transparent UI, don't render UI beneath it
    if (!state->HasTransparentUI())
    {
        break;
    }
}

// Render UI from bottom to top
for (auto it = uiStates.rbegin(); it != uiStates.rend(); ++it)
{
    (*it)->RenderUI();
}
```

## 3.2. Bonus Kreasi Mandiri

Daftarkan setiap kreasi tambahan yang anda buat dibagian ini serta deskripsi singkat mengenai implementasinya. Setiap kreasi tambahan yang anda buat akan dihargai jadi daftarkanlah selengkap-lengkapnya. Poin ini akan dipertimbangkan sebagai nilai tambahan ke nilai akhir kelompok dan individu.

### 3.2.1. Custom RNG Algorithm

RNG diimplementasikan menggunakan algoritma Permuted Congruential Generator, yakni algoritma pRNG (*pseudo Random Number Generator*), yang biasa digunakan pada game. Implementasinya cukup sederhana, yakni definisi state, dan state tersebut akan diacak menggunakan *bit-shifting*. Selain itu, implementasi ini juga dinormalisasi sesuai kebutuhan. Sebagai contoh, kelas RNG bisa generate angka random, list of random number, generate probability (0-1 in double scale), generate percentage (0-100 in double scale), dan generate angka random pada suatu range yang didefinisikan oleh user. Implementasi dari RNG dapat dilihat pada cuplikan kode di bawah.

```
#include "rng/rng.hpp"

usl RNG::state = 0;

RNG::RNG()
{
    if (state == 0)
    {
        auto now = std::chrono::high_resolution_clock::now().time_since_epoch().count();
        state = static_cast<usl>(now);
    }
}
```

```
RNG::RNG(ul seed)
{
    if (state == 0)
    {
        auto now = std::chrono::high_resolution_clock::now().time_since_epoch().count();
        state = static_cast<ul>(now);
    }
    setSeed(seed);
}

void RNG::setSeed(ul seed)
{
    generateNext();
    state += seed;
    generateNext();
}

RNG& RNG::operator=(const RNG& other)
{
    if (this != &other)
    {
        this->state = other.state;
    }
    return *this;
}
```

```
RNG::~RNG() { }

unsigned int RNG::generateNext()
{
    usl oldState = state;

    state = oldState * multiplier + increment;

    unsigned int shifted = ((oldState >> 18) ^ oldState) >> 27;
    unsigned int rot      = oldState >> 59;

    return (shifted >> rot) | (shifted << ((-rot) & 31));
}

unsigned int RNG::generateInRange(unsigned int max)
{
    if (max == 0)
        return 0;

    unsigned int threshold = -max % max;

    while (true)
    {
        unsigned int r = generateNext();
        if (r >= threshold)
        {
            return r % max;
        }
    }
}
```

```
        }
    }
}

unsigned int RNG::generateInRange(unsigned int min, unsigned int max)
{
    if (min >= max)
        return min;

    return min + generateInRange(max - min + 1);
}

double RNG::generateProbability()
{
    return generateNext() / (double)4294967296; // 2^32
}

double RNG::generatePercentage()
{
    return generateProbability() * 100.0;
}

// Vector of random numbers
vector<unsigned int> RNG::generateRandomVector(size_t size, unsigned int max)
{
    vector<unsigned int> randomVector;
    randomVector.reserve(size);
```

```
        for (size_t i = 0; i < size; i++)
        {
            randomVector.push_back(generateInRange(max));
        }

        return randomVector;
    }

// Vector of percentages
vector<double> RNG::generatePercentageVector(size_t size)
{
    vector<double> percentageVector;
    percentageVector.reserve(size);

    for (size_t i = 0; i < size; i++)
    {
        percentageVector.push_back(generatePercentage());
    }

    return percentageVector;
}
```

### 3.2.2. Musik pada Game

Musik pada program ini diimplementasikan dan diintegrasikan melalui sumber daya terpusat dengan memanfaatkan kelas `ResourceManager`. Musik latar belakang akan diinisialisasi dengan memuat file `sound buffer` yang kemudian diputar menggunakan objek `sf::sound` dari pustaka SFML. Musik juga dikendalikan secara dinamis melalui `StateManagement` sehingga ketika state tertentu dijalankan, musik latar dapat dihentikan, diputar atau dijeda. Berikut adalah kode inisialisasi musik yang digunakan didalam program tepatnya pada file `MainMenuState.hpp`

```
/*
 * @brief Exit the main menu state
 *
 * This method is called when the main menu state is exited.
 * It handles thread cleanup and resource unloading.
 */
void Exit() override; // Important for thread cleanup!
void Pause() override;
void Resume() override;

private:
    sf::Sound m_backsound;           ///< Background music sound
    sf::Sound m_buttonHoverSound;   ///< Button hover sound
    sf::Sound m_buttonClickSound;   ///< Button click sound
    sf::Font  m_font;
```

### 3.2.3. Algoritma Sorting Pada Shop

Sorting diimplementasikan dengan menggunakan std::sort yang menerima fungsi dengan definisi yang telah diberikan, sorting tidak dapat dilakukan tanpa if-else sebab terdapat *rarity S* yang menjadi *rarity* tertinggi, sedangkan secara ASCII berada dengan urutan yang berbeda.

```
std::vector<Shop::StockEntry> Shop::getSortedStock(const std::vector<std::string>&
                                                     categories,
                                                     const std::string& sortBy,
                                                     bool
                                                     isDescending)
{
    std::vector<StockEntry> filteredStock = filterStockByCategories(categories);

    if (sortBy == "Rarity")
    {
        std::sort(filteredStock.begin(),
                  filteredStock.end(),
                  [isDescending](const StockEntry& a, const StockEntry& b) {
                    // Rarity order: E < D < C < B < A < S for ascending
                    // For descending, we reverse this: S < A < B < C < D < E
                    char ra = std::get<0>(a).getRarity();
                    char rb = std::get<0>(b).getRarity();

                    // For ascending sort
                    if (!isDescending)
                    {
```

```
        if (ra == 'E' && rb != 'E')
            return true;
        if (ra != 'E' && rb == 'E')
            return false;
        if (ra == 'D' && rb != 'D' && rb != 'E')
            return true;
        if (ra != 'D' && ra != 'E' && rb == 'D')
            return false;
        if (ra == 'C' && rb != 'C' && rb != 'D' && rb != 'E')
            return true;
        if (ra != 'C' && ra != 'D' && ra != 'E' && rb == 'C')
            return false;
        if (ra == 'B' && rb != 'B' && rb != 'C' && rb != 'D' && rb != 'E')
            return true;
        if (ra != 'B' && ra != 'C' && ra != 'D' && ra != 'E' && rb == 'B')
            return false;
        if (ra == 'A' && rb == 'S')
            return true;
        if (ra == 'S' && rb == 'A')
            return false;
    }
    // For descending sort
} else
{
    if (ra == 'S' && rb != 'S')
        return true;
    if (ra != 'S' && rb == 'S')
```

```
        return false;
    if (ra == 'A' && rb != 'A' && rb != 'S')
        return true;
    if (ra != 'A' && ra != 'S' && rb == 'A')
        return false;
    if (ra == 'B' && rb != 'B' && rb != 'A' && rb != 'S')
        return true;
    if (ra != 'B' && ra != 'A' && ra != 'S' && rb == 'B')
        return false;
    if (ra == 'C' && rb != 'C' && rb != 'B' && rb != 'A' && rb != 'S')
        return true;
    if (ra != 'C' && ra != 'B' && ra != 'A' && ra != 'S' && rb == 'C')
        return false;
    if (ra == 'D' && rb == 'E')
        return true;
    if (ra == 'E' && rb == 'D')
        return false;
}

return false;
}) );
}
else if (sortBy == "Price")
{
    std::sort(filteredStock.begin(),
              filteredStock.end(),
              [isDescending](const StockEntry& a, const StockEntry& b) {
```

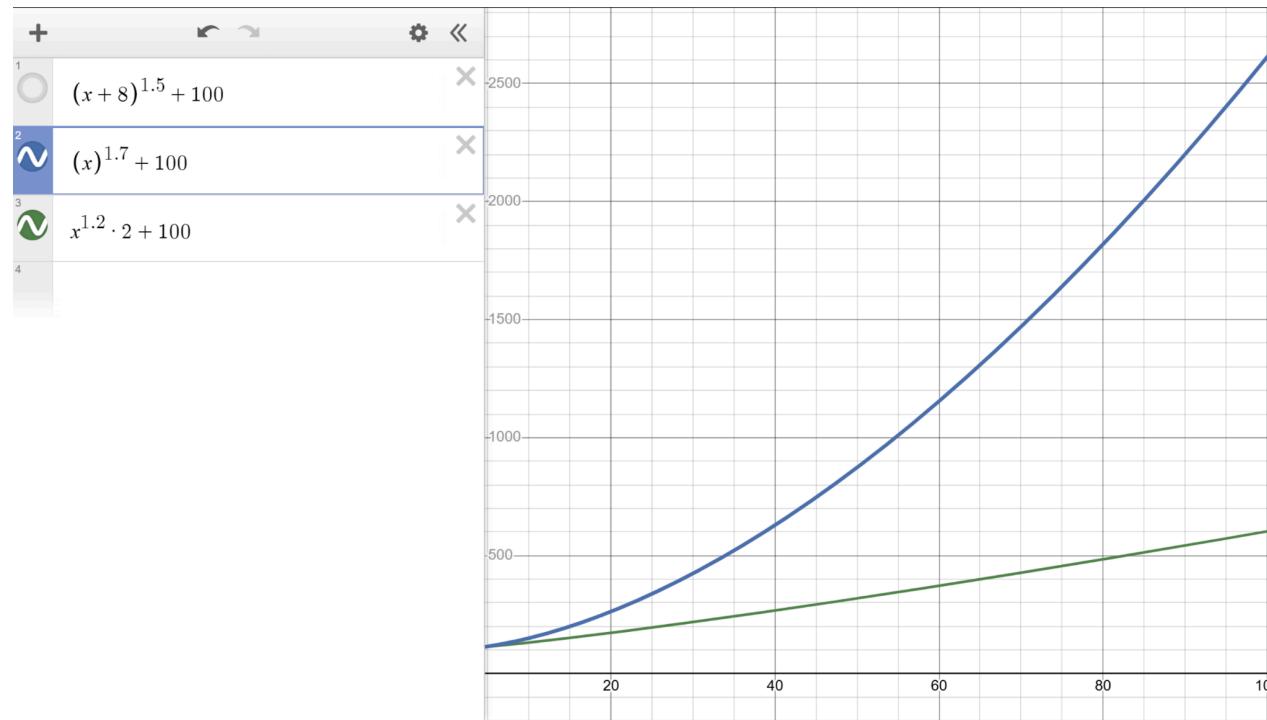
```
        if (!isDescending)
            return std::get<1>(a) < std::get<1>(b); // Ascending (lowest to
highest)
        else
            return std::get<1>(a) > std::get<1>(b); // Descending (highest to
lowest)
    } );
}
else if (sortBy == "Stock")
{
    std::sort(filteredStock.begin(),
              filteredStock.end(),
              [isDescending](const StockEntry& a, const StockEntry& b) {
        if (!isDescending)
            return std::get<2>(a) < std::get<2>(b); // Ascending (lowest to
highest)
        else
            return std::get<2>(a) > std::get<2>(b); // Descending (highest to
lowest)
    } );
}

return filteredStock;
}
```

### 3.2.4. Levelling Progression

Purry Levelling, tidak lengkap rasanya jika tidak diikuti dengan progresi level yang baik. Untuk hal ini, kami mengimplementasikan

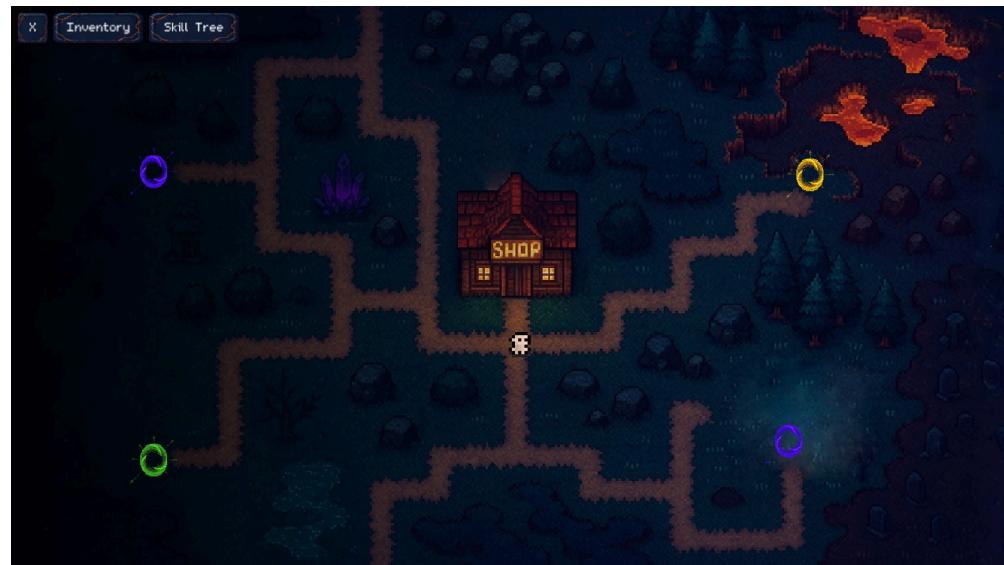
Level progression dalam game mendeskripsikan dua hal utama: (1) EXP requirement, berapa EXP yang dibutuhkan untuk naik dari level  $x$  ke level  $x + 1$  dan (2) EXP reward, berapa EXP yang pemain dapatkan dari satu dungeon di level  $x$ . Pada grafik di bawah, yang berwarna biru menunjukkan fungsi *requirement*, dan yang hijau adalah *base reward per dungeon*.

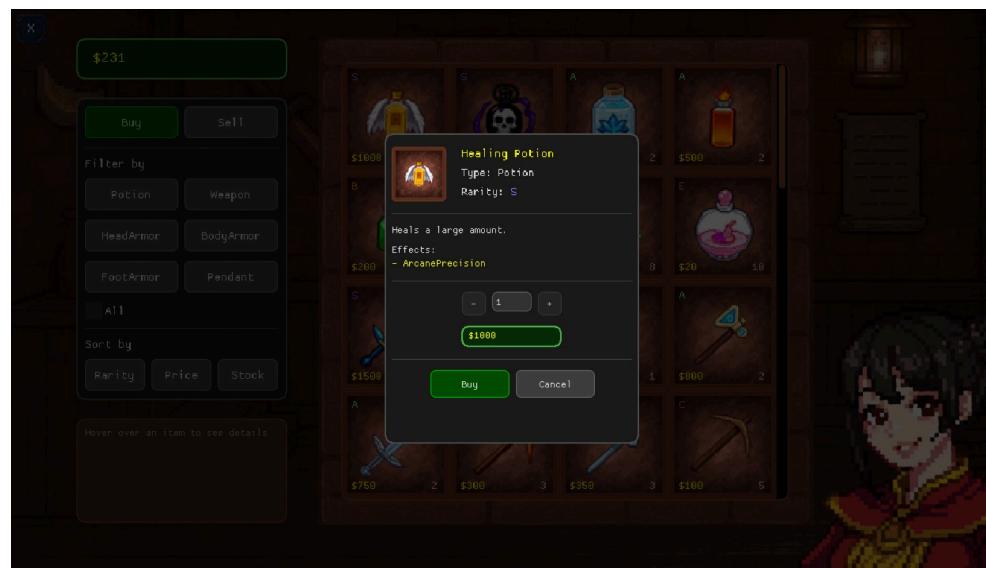


Di *early-game*, kurva biru (EXP requirement) dan hijau (EXP reward) berpotongan di sekitar level pertama, keduanya bernilai sekitar 100, sehingga satu kali clear dungeon hampir selalu cukup untuk naik satu level. Hal ini menciptakan pengalaman on-boarding yang mulus, di mana pemain langsung merasakan kemajuan signifikan tanpa frustrasi. Namun begitu memasuki *mid-game* (misalnya level 10–30), kebutuhan EXP yang dipacu oleh fungsi  $x^{1.7} + 100$  mulai tumbuh lebih cepat daripada reward  $2x^{1.2} + 100$ , sehingga celah antara EXP yang dibutuhkan dan yang diperoleh melebar: pemain memerlukan beberapa dungeon untuk satu level, memperkenalkan elemen grind yang terukur. Di *late-game* (level di atas 50), perbedaan ini semakin dramatis, requirement melonjak ke angka ribuan sementara reward hanya ratusan, menciptakan tantangan nyata dan rasa pencapaian tinggi ketika akhirnya berhasil naik level. Pola power-law ini (eksponen 1.7 vs. 1.2) memastikan progresi awal terasa mudah dan memotivasi, lalu berangsurn menantang di level tinggi, menjaga keseimbangan antara fun dan grind sepanjang perjalanan pemain.

### 3.2.5. Tampilan GUI yang indah dan menawan











#### 4. Pembagian Tugas

Modul (dalam poin spek)	Implementer	Tester
Unit	13523002	13523002
Character	13523002	13523002
Skill & Skill Path	135230028	13523004, 135230028
Effect	13523002, 13523028	13523002
Dungeon & Chamber	13523090	13523028, 13523090
Mobs	13523002	13523002, 13523090
Shop	13523004, 13523011	13523004, 13523011
Items	13523004, 13523011	13523002, 13523004, 13523011
Damage	13523028	13523011, 13523028
Inventory	13523002, 13523004, 13523011	13523004, 13523004, 13523011
Save & Load	13523011	13523011
Quest	13523090	13523002, 13523090
Cheat	13523002	13523028, 13523090
GUI	13523002, 13523004, 13523011, 13523028, 13523090	13523002, 13523004, 13523011, 13523028, 13523090