LAPORAN TUGAS BESAR
IF2210 Pemrograman Berorientasi Objek

Willy Wangky and the Engimon Factory

Dipersiapkan Oleh:
Kelas 04
Kelompok Mangga

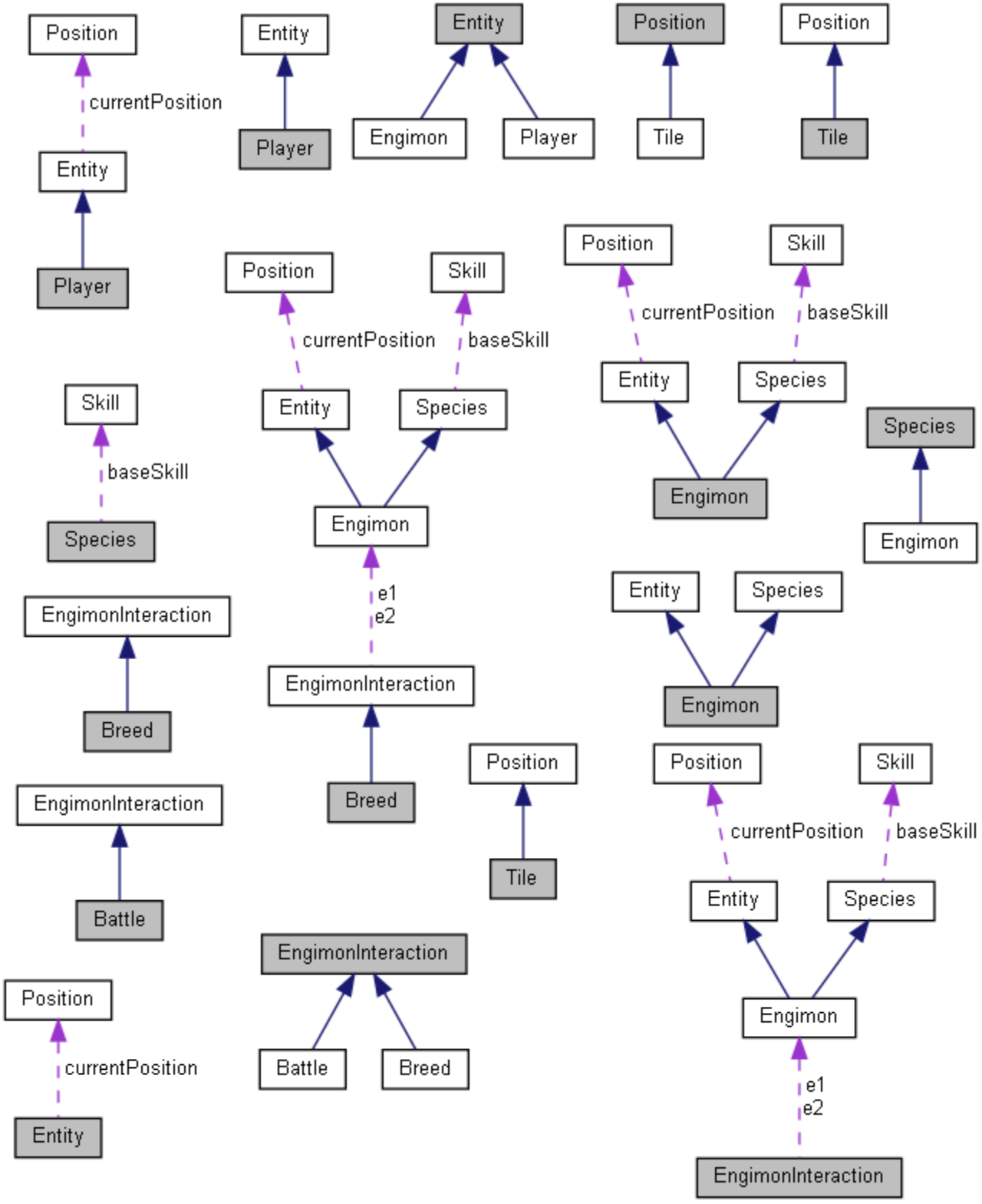
Gregorius Dimas Baskara	13519190
Christian Gunawan	13519199
Muhammad Fawwaz Naabigh	13519206
M. Alfandavi Aryo Utomo	13519211
Tanur Rizaldi Rahardjo	13519214

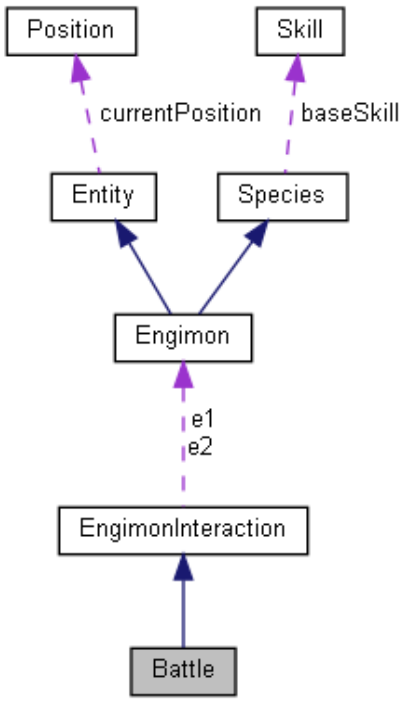
Asisten:
Muhammad Fariz Luthfan Wakan

Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung
Jl. Ganesha 10, Bandung 40132

Daftar Isi

1. Diagram Kelas





Gambar 1. Diagram Kelas

Alasan kami memilih diagram tersebut karena jika kita melakukan implantasi, maka kita dapat menjadikan intermediate output. Kemudian kelas hanya perlu di link ke kelas intermediate output. Kelebihan dari desain ini adalah waktu kompilasi yang lebih cepat. Kekurangan dari desain ini adalah jika kita merubah kelas *header*-nya, maka akan berpengaruh kepada kelas *class*-nya.

2. Penerapan Konsep OOP

2.1. Inheritance & Polymorphism

Berikut ini merupakan beberapa contoh aplikasi konsep Inheritance & Polymorphism pada program:

2.1.1. Parent Class Entity

```
class Engimon : public Entity, Species
```

```
class Player : public Entity
```

Gambar 2 - Inheritance dan Polymorphism pada parent class Entity

Kelas **Entity** merupakan parent dari kelas **Engimon** dan kelas **Player** yang mewarisi atribut-atribut dan method yang dimiliki oleh kelas **Entity** tersebut. Dengan penggunaan Inheritance melalui Parent Class **Entity**, maka atribut-atribut dan method yang dimiliki oleh objek-objek yang akan ditampilkan ke dalam **Map** hanya perlu dituliskan sekali, sehingga code menjadi DRY (Don't Repeat Yourself). Selain itu, melalui polymorphism dimungkinkan pula perlakuan objek berkelas **Engimon** sebagai parentnya (objek berkelas **Entity**).

2.1.2. Parent Class Position

```
class Tile : public Position
```

Gambar 3 - Inheritance pada parent class Position

Kelas **Position** merupakan parent dari kelas **Tile** yang mewarisi atribut-atribut dan method yang dimiliki oleh kelas **Position** tersebut. Dengan penggunaan Inheritance melalui Parent Class **Position**, maka code menjadi DRY terlebih-lebih karena Kelas **Position** akan digunakan pula sebagai **atribut composition** dari kelas **Entity**.

2.1.3. Parent Class Species

```
class Engimon : public Entity, Species
```

Gambar 4 - Inheritance dan Polymorphism pada parent class Species

Kelas **Species** merupakan parent dari kelas **Engimon** yang mewarisi atribut-atribut dan method yang dimiliki oleh kelas **Species** tersebut.

2.2. Method/Operator Overloading

Berikut ini merupakan beberapa contoh dari aplikasi method/operator overloading:

2.2.1. Operator Overload Kelas Tile

```
bool Tile::operator==(TileType target) {
    return this->tileType == target;
}
```

*Gambar 5 - Operator overload pada kelas **Tile***

Terdapat satu operator overload pada kelas **Tile**, yaitu operator ==. Operator ini digunakan untuk mengembalikan suatu boolean yang berisi nilai kebenaran apakah atribut tileType dari objek Tile terkait sama dengan target atau tidak. Dengan

menggunakan operator ini, maka pemeriksaan tileType menjadi lebih mudah untuk dibaca dan dimengerti, serta juga dapat memangkas penulisan kode.

2.2.2. Operator Overload Kelas Position

```
Position operator+(const Position &pos2);
Position operator+(const Tile &pos2);
// Addition operator
const Position& operator+=(const Position &pos2);
const Position& operator+=(const Tile &pos2);
// Addition Assignment operator
const Position& operator=(const Position &pos2);
const Position& operator=(const Tile &pos2);
// Assignment operator
bool operator==(const Position &pos2);
// Return true if and only if both posX and posY are equal
```

Gambar 6 - Operator overload pada class Position

Kelas Position memiliki 7 operator overload, yaitu 2 operator+ dengan parameter yang berbeda, 2 operator+= dengan parameter yang berbeda, 2 operator= dengan parameter yang berbeda, dan 1 operator ==. Semua operator overload digunakan untuk mengoperasikan dua objek position yang masing-masing memiliki atribut koordinat posX dan posY.

2.3. Template & Generic Classes

```
template<class T>
class Inventory {
    private:
        std::list<T> itemList;
        const unsigned int maxCapacity;

    public:
        Inventory(unsigned int maxCap) : maxCapacity(maxCap), itemList() {
        }

        std::list<T> getItemList() {
            return itemList;
        }
        // Getting entire copy list of inventory

        bool addItem(int skillID) {

        }
        // Return false if overcapacity
        bool deleteItem(int skillID) {

        }
        // Return false if not found item
};
```

Gambar 7 - Template & Generic class pada class Inventory

Pada spesifikasi, disebutkan bahwa **Inventory** dapat menyimpan informasi mengenai benda-benda yang dimiliki **Player** yang merupakan instansiasi dari kelas **Engimon** dan **Skill Item**. Sebab karakteristik **Inventory** yang dibutuhkan untuk menyimpan kedua kelas tersebut adalah sama dan pembeda hanya terdapat pada tipe kelas, sehingga **Inventory** diimplementasikan berupa template & generic class.

2.4. Exception

2.4.1. Exception pada Konstruktor Kelas Map (map.cpp)

```
Map::Map(string filename) : randomEngimonMoveProbability(20) {
    // Unused
    seaStartX = 0;
    seaStartY = 0;
    // TODO : Engimon loading (???), maybe not needed
    ifstream mapFile = ifstream(filename);
    if (mapFile.is_open()) {
        vector<vector<Tile>> flippedMap;
        string mapRow;
        unsigned int i = 0;
        while (getline(mapFile, mapRow)) {
            vector<Tile> column;
            sizeX = mapRow.length();
            for (unsigned int j = 0; j < mapRow.length(); j++) {
                if (mapRow[j] == 'o')
                    column.push_back(Tile(i, j, Sea));
                else
                    column.push_back(Tile(i, j, Grass));
            }
            i++;
            flippedMap.push_back(column);
        }
        sizeY = i;
        mapFile.close();

        // Flipping map
        for (i = 0; i < sizeX; i++) {
            vector<Tile> row;
            for (unsigned int j = 0; j < sizeY; j++)
                row.push_back(flippedMap[j][i]);
            tileMatrix.push_back(row);
        }
    }
    else
        throw filename;
}
```

Gambar 8 - Exception pada constructor class Map

Pada exception ini, akan di-`throw` sebuah **filename** ketika file yang akan dibuka tidak ditemukan. **Filename** yang di-`throw` tersebut nantinya akan "ditangkap" di engine.cpp dan diteruskan ke *constructor*. Dengan menggunakan exception ini, maka error handling dapat dilakukan dengan lebih terstruktur

2.4.2. Exception pada method loadSkillDatabase() dalam kelas SkillDatabase

```
void SkillDatabase::loadSkillDatabase(string filename) {
    ifstream skillFile = ifstream(filename);
    if (skillFile.is_open()) {
        string skillName;
        int skillPower;
        int skillID;
        string typeString;
        ElementType skillElement;
        while (!skillFile.eof()) {
            skillFile >> skillID;
            skillFile >> skillName;
            skillFile >> skillPower;
            skillFile >> typeString;

            if (typeString == "Ground")
                skillElement = Ground;
            else if (typeString == "Electric")
                skillElement = Electric;
            else if (typeString == "Fire")
                skillElement = Fire;
            else if (typeString == "Water")
                skillElement = Water;
            else if (typeString == "Ice")
                skillElement = Ice;
            else
                skillElement = NoElement;

            addSkill(Skill(skillID, skillPower, skillName, skillElement));
        }

        skillFile.close();
    }
    else
        throw filename;
}
```

Gambar 9 - Exception pada method loadSkillDatabase pada class SkillDatabase

Sebagaimana pada map, pada method ini, exception digunakan apabila file tidak ditemukan, dengan **filename** yang akan di-`throw`.

2.4.3. Exception pada method getSkill() dalam kelas SkillDatabase dan method getSpecies dalam kelas SpeciesDatabase

```
Skill SkillDatabase::getSkill(int skillID) {
    for (unsigned int i = 0; i < skillDatabase.size(); i++)
        if (skillDatabase[i].getSkillID() == skillID)
            return skillDatabase[i];
    throw skillID;
}
```

Gambar 10 - Exception pada method getSkill pada class SkillDatabase

Berbeda dengan dua exception sebelumnya, pada method ini exception digunakan ketika **skillID** yang dicari tidak ditemukan.

2.5. C++ Standard Template Library

Berikut ini beberapa contoh dari banyak penggunaan STL pada aplikasi:

2.5.1. Penggunaan STL Vector pada kelas Map

```
#include <vector>
#include <string>

class Map {
    private:
        std::vector<std::vector<Tile>> tileMatrix;
        unsigned int sizeX;
        unsigned int sizeY;
        unsigned int seaStartX;
        unsigned int seaStartY;
        const unsigned short int randomEngimonMoveProbability;
        // Probability in percent, default value is 15%
```

Gambar 11 - Implementasi STL Vector pada class Map

Program diatas dapat dilihat pada inventory. Penggunaan Standard Template Library vector dan string. Pada Vector untuk menyimpan maps pada bentuk 2d. Terlihat pada Vector<Vector<Name>> yang pada program ini name adalah Tile

2.5.2. Penggunaan STL Queue pada kelas Message

```
class Message {
    private:
        std::queue<std::string> messageQueue;
        const unsigned int maxQueue;
        const unsigned int maxStringLength;
```

Gambar 12 - Implementasi STL Queue pada class Message

Pada kelas **Message** digunakan STL C++ berupa Queue yang digunakan untuk menyimpan antrian pesan/message yang akan ditampilkan ke layar dalam box message. Penggunaan STL Queue di sini memberikan keuntungan karena menghemat banyak kode yang harus dibuat secara manual. Selain itu, STL Queue juga memiliki standar kualitas yang teruji sehingga hampir dipastikan tidak akan terjadi bug dari dalam objek queue itu sendiri. Pengaksesan pun mudah karena sudah disediakan antar muka di dalam dokumentasi.

2.6. Konsep OOP lain

Pada tugas ini, kami menerapkan konsep OOP lain yang tidak dicantumkan pada spesifikasi tugas ini. Adapun konsep tersebut adalah Abstract Base Class dan Aggregation.

2.6.1. Abstract Base Class

```
class EngimonInteraction {
    protected:
        Engimon* e1;
        Engimon* e2;

    public:
        EngimonInteraction(Engimon* engimon1, Engimon* engimon2);
        virtual Engimon* getEngimonOne() = 0;
        virtual Engimon* getEngimonTwo() = 0;
};
```

Gambar 13 - Header dari Abstract Base Class EngimonInteraction

Terdapat penggunaan abstract base class bernama Engimon Interaction yang memberikan interface dan mengeneralisasi beberapa atribut yang dimiliki kelas **Breed** dan **Battle** yang memiliki kesamaan atribut utama berupa pointer yang menunjuk ke dua engimon. Terlihat getEngimonOne() dan getEngimonTwo() merupakan kedua method yang pure virtual dan harus diimplementasikan pada kedua child, yaitu kelas **Breed** dan **Battle.**

2.6.2. Aggregation

```
class Species {
    protected:
        int speciesID;
        std::string speciesName;
        Skill baseSkill;
        std::set<ElementType> elements;
        std::string interactionDescription;
        char speciesChar;
```

```
Species::Species(int id, string name, Skill skill, string desc, ElementType element1, ElementType element2)
```

Gambar 14 - Implementasi Aggregation pada property dan constructor class Species

Terdapat beberapa penggunaan aggregation dalam beberapa kelas pada aplikasi. Contohnya, kelas Species memiliki sebuah atribut berupa objek berkelas Skill yang bernama baseSkill. Dalam kata lain, dapat dikatakan bahwa 'Species has a Skill'. Meskipun demikian, ketika objek Species dihancurkan, objek Skill dapat berdiri secara independen dan tidak ikut dihancurkan. Dapat dilihat pula pada tangkap gambar di atas bahwa saat objek Species dibuat, maka objek skill sebenarnya sudah diciptakan di luar kelas dan tidak dependen pada kelas tersebut.

```
class Entity {
    protected:
        EntityID entityID;
        Position currentPosition;
        char entityChar;
        static int entityCount;
```

```
Entity::Entity(Position pos, EntityID eID, char eChar)
    entityCount++;
}
```

Gambar 15 - Implementasi Aggregation pada property dan constructor class Entity

Berikutnya, terdapat contoh penerapan lainnya pula pada kelas Entity, di mana setiap objek berkelas Entity memiliki atribut berupa objek berkelas Position bernama currentPosition yang juga bersifat independen dari kelas Entity (Saat objek Entity diciptakan, objek Position sudah diciptakan sebelumnya). Dalam kata lain dapat dikatakan bahwa 'Entity has a Position'.

3. Bonus Yang dikerjakan

3.1. Bonus yang diusulkan oleh spek

Dari ketiga bonus yang diusulkan pada spesifikasi tugas ini, dengan mempertimbangkan waktu, kami mengerjakan dua dari tiga bonus yang diusulkan, yaitu *dual breeding* dan *purely random wild engimon generation*.

3.1.1. Dual Breeding

Dual breeding diimplementasikan dengan mengambil elemen mana yang paling menguntungkan untuk si anak hasil dual breeding dengan fungsi get element advantage, dan menyimpan spesies dari parent yang memiliki elemen paling menguntungkan sebagai spesies anak tersebut.

```
set<ElementType> parent1_element = parent1->getElements();
set<ElementType> parent2_element = parent2->getElements();
ElementType child_element1 = NoElement;
ElementType child_element2 = NoElement;

auto it1 = parent1_element.begin(), it2 = parent2_element.begin();
ElementType el1 = *it1;
ElementType el2 = *it2;
float parent1ElementAdvantage = getElementAdvantage(el1, el2);
float parent2ElementAdvantage = getElementAdvantage(el2, el1);

bool advantageBranch = false;
fromP1 = false, fromP2 = false;
if(el1==el2){
    child_element1 = el1;
    fromP1 = true;
}else{
    advantageBranch = true;
    if(parent1ElementAdvantage>parent2ElementAdvantage) {
        child_element1 = el1;
        fromP1 = true;
    }
    else if(parent1ElementAdvantage<parent2ElementAdvantage) {
        child_element1 = el2;
        fromP2 = true;
    }
    else{
        child_element1 = el1;
        child_element2 = el2;
    }
}
```

Gambar 16 - Dual breeding pada class Breeding

3.1.2. Purely Random Wild Engimon Generation

Engimon starter yang dimiliki oleh player dan wild engimon yang di-*spawn* ke dalam map merupakan engimon yang dipilih secara purely random. Masalah diselesaikan menggunakan pendekatan random generator menggunakan *seed* berupa timestamp saat ini. Penggunaan seed ini ditujukan agar memastikan aspek random terjaga dan tidak mengikuti suatu siklus tertentu. Setelah dilakukan *seeding*, integer yang dihasilkan kemudian akan di modulo dan menghasilkan index random untuk pengambilan wild engimon dari database.

```
void Engine::evaluateTick() {
    map.wildEngimonRandomMove();
    unsigned int randomNumber = rand() % 100;
    if (Entity::getEntityCount() < entitySpawnLimit && randomNumber < wildEngimonSpawnProbability) {
        unsigned int randomSpeciesID = (rand() % (speciesDB.getSpeciesCount() - 1)) + 1;
        // TODO : Extra, fix mod operator
        Engimon *spawnedEngimon = map.spawnWildEngimon(speciesDB.getSpecies(randomSpeciesID), spawnLevelCap);

        // Random skill adder, 3x roll chance to get additional skill
        // Higher level cap also grant higher chance to get multiple skill,
        // up to 100% chance getting 4 skill
        set<ElementType> engimonElements = spawnedEngimon->getElements();
        for (int i = 0; i < 3; i++) {
            if ((rand() % 100) < 10 + spawnLevelCap) {
                // Try catch block random skill
                Skill generatedSkill = skillDB.getSkill(1); // Placeholder
                bool isSkillGeneratedValid = false;
                while (not isSkillGeneratedValid) {
                    try {
                        generatedSkill = skillDB.getSkill(rand() % maxSkillID);
                        if (generatedSkill.isElementCompatible(*engimonElements.begin())) {
                            isSkillGeneratedValid = true;
                        }
                        else if (++engimonElements.begin() != engimonElements.end()) {
                            if (generatedSkill.isElementCompatible(*(++engimonElements.begin())))
                                isSkillGeneratedValid = true;
                        }

                        if (isSkillGeneratedValid) {
                            int masteryLevelUp = (rand() % (1 + spawnLevelCap/5)) + 1;
                            for (int j = 0; j < masteryLevelUp; j++)
                                generatedSkill.levelUpMastery();

                            spawnedEngimon->addSkill(generatedSkill);
                        }
                    }
```

Gambar 17 - Implementasi Purely Random Wild Engimon Generator pada class Engine

3.1.3. Unit Testing Implementation

Unit testing tidak diimplementasikan karena terlalu banyak untuk setiap sub-program cpp yang telah dibuat.

3.2. Bonus Kreasi Mandiri

Selain bonus yang diusulkan di spesifikasi tugas, ada beberapa hal lain yang dikerjakan di luar spesifikasi untuk menunjang tugas ini, baik dari segi fungsional maupun dekoratif agar tidak terlalu membosankan.

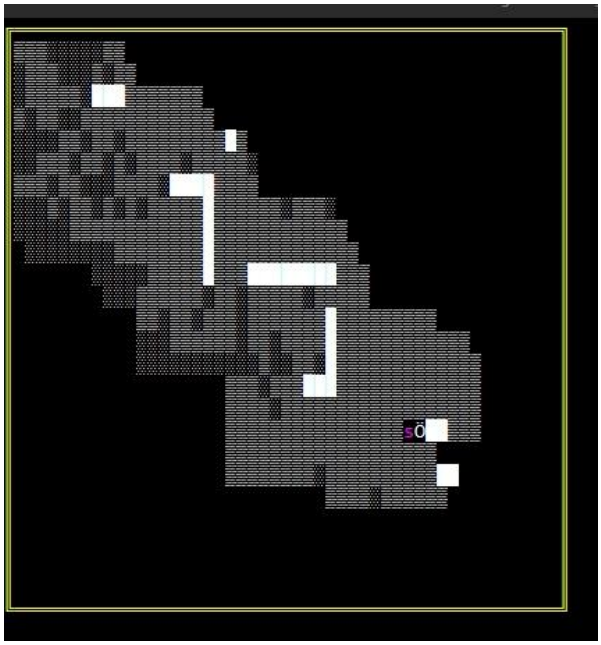
3.2.1. Penerapan multi-threading dan mutex sederhana untuk input.

```
void PlayerInput::inputLoop() {
    while (isRunning) {
        if (inputBuffer.size() < maxInputBuffer && isReading) {
            // Critical section, blocking input buffer processing
            inputLock.lock();
            if ((GetKeyState(VK_UP) & 0x8000) || (GetKeyState('W') &
0x8000))
                inputBuffer.push(Up);
            if ((GetKeyState(VK_DOWN) & 0x8000) || (GetKeyState('S')
& 0x8000))
                inputBuffer.push(Down);
            if ((GetKeyState(VK_LEFT) & 0x8000) || (GetKeyState('A')
& 0x8000))
                inputBuffer.push(Left);
            if ((GetKeyState(VK_RIGHT) & 0x8000) || (GetKeyState('D'
) & 0x8000))
                inputBuffer.push(Right);
            if ((GetKeyState('1') & 0x8000))
                inputBuffer.push(Number1);
            if ((GetKeyState('2') & 0x8000))
                inputBuffer.push(Number2);
            if ((GetKeyState('3') & 0x8000))
                inputBuffer.push(Number3);
            if ((GetKeyState('4') & 0x8000))
                inputBuffer.push(Number4);
            if ((GetKeyState('E') & 0x8000)) {
                // Preventing multiple command mode request
                if ((!inputBuffer.empty() && inputBuffer.front() != K
eyboardE) || inputBuffer.empty())
                    inputBuffer.push(KeyboardE);
            }
            if ((GetKeyState(VK_ESCAPE) & 0x8000))
                inputBuffer.push(EscKey);
            inputLock.unlock();

            std::this_thread::sleep_for(std::chrono::milliseconds(inp
utDelayMillisecond));
            // If input reading condition satisfied, then wait for fe
w ms before register new input
        }
    }
}

void PlayerInput::startReadInput() {
    // TODO : Extra, potential memory leak, use .join() at stop
    inputThread = new std::thread(&PlayerInput::inputLoop, this);
    isRunning = true;
    isReading = true;
}
```

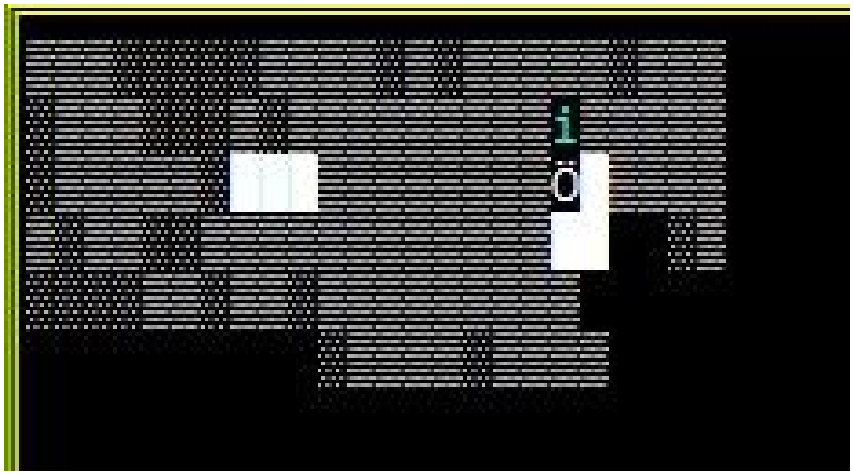
Gambar 18 - Multi-thread dan mutex pada class PlayerInput

Pada playerinput.cpp, saat **startReadInput** dipanggil, sebuah thread baru akan dibuat. Thread tersebut akan melakukan **inputLoop** dan meneruskan hingga terdapat sinyal untuk berhenti. Mutex digunakan untuk mencegah

3.2.2. Penambahan Fog of War dan Line of Sight



Gambar 19 - Fog of War pada saat permainan dimulai



Gambar 20 - Line of sight ketika terblok tembok

Pada mulai permainan, player tidak dapat melihat isi penuh dari suatu map dan hanya dapat melihat petak yang sudah ia lintasi saja. Oleh karena itu, bila ingin menemukan Wild Engimon pada awal permainan, maka player harus menerka/menebak-nebak posisi Wild Engimon itu terlebih dahulu. Dengan *line of sight* player tidak akan selalu bisa melihat area yang ada. Jika posisi jauh dari player

atau terdapat tembok yang menghalangi pandangan player, maka pemain tidak akan bisa melihat lokasi petak tersebut. Pada tahap pengkalkulasian digunakan algoritma *raycasting* sederhana.

3.2.3. Penambahan auto-tick increment

Apabila player dalam status AFK (Away From Keyboard) atau tidak bergerak selama satu detik penuh, maka setiap Wild Engimon akan bergerak secara otomatis dengan interval satu detik. Wild Engimon akan berhenti bergerak secara otomatis/bergerak secara normal apabila sebelum timer 1 detik berakhir, player sudah mendapatkan input kembali untuk bergerak.

```
while (isEngineRunning) {
    // Drawing map and message box
    renderer.drawMap(map);
    renderer.drawMessageBox(messageList);
    statRenderer.drawMessageBox(statMessage);
    battleRenderer.drawMessageBox(battleMessage);
    std::this_thread::sleep_for(std::chrono::milliseconds(10));
    if (evaluteInput() && not isCommandMode) {
        evaluteTick();
    }
    else if (isCommandMode) {
        commandMode();
        // Call command mode
    }

}
userInput.stopReadInput();
```

Gambar 21 - Implementasi auto-tick increment

3.2.4. Wild Engimon Level and Skill Scaling

Apabila engimon player mengalami level up, maka level cap dari seluruh engimon yang berada di map (baik engimon milik player maupun wild) akan meningkat pula. Wild Engimon yang di-*generate* akan memiliki level dan skill yang random, namun memiliki level maksimal = level cap. Meningkatnya level cap juga berpengaruh pada skill yang diberikan pada Wild Engimon yang digenerasi. Semakin tinggi level cap, maka semakin besar peluang Wild Engimon mendapatkan skill baru dengan level skill yang tinggi.

```cpp
// Level up checking
int levelGained = player.getCurrentEngimon()->xpGain(targetEngimon->getLevel()*xpMultiplier);
if (levelGained > 0) {
    spawnLevelCap += levelGained;
}

// Random mastery level up
vector<Skill>& SkillReference = player.getCurrentEngimon()->getSkillListRef();
for (unsigned i = 0; i < SkillReference.size(); i++) {
    if ((rand() % 100) < 40)
        SkillReference[i].levelUpMastery();
}

// Random skill drop
int dropRoll = rand() % 100;
if ((unsigned) dropRoll < wildEngimonDropProbability) {
    SkillItem droppedSkill = 1; // Placeholder
    string droppedSkillName = "";
    bool isSkillDropValid = false;
    set<ElementType> targetEngimonElements = targetEngimon->getElements();
    while (not isSkillDropValid) {
        // Try catch block for random skill picking
        try {
            Skill randomSkill = skillDB.getSkill(rand() % maxSkillID);
            if (randomSkill.isElementCompatible(*(targetEngimonElements.begin()))) {
                isSkillDropValid = true;
            }
            else if (++(targetEngimonElements.begin()) != targetEngimonElements.end()) {
                // If multi element, check other one
                if (randomSkill.isElementCompatible(*(++(targetEngimonElements.begin()))))
                    isSkillDropValid = true;
            }
```

Gambar 22 - Level dan Skill Scaling pada wild engimon

3.2.5. Penambahan warna

Pemberian warna pada tampilan permainan dengan menggunakan HANDLE hstdout = GetStdHandle(STD_OUTPUT_HANDLE) yang menggunakan library Windows.h. Setiap icon engimon pada map akan ditampilkan berdasarkan warna elemennya.

Gambar 23 - Hasil penambahan warna

4. Pembagian Tugas

Modul (dalam poin spek)	Designer	Implementer
1.a. Engimon bermacam-macam spesies	13519190	13519190
1.b. Spesies dengan elemen konsisten	13519190	13519190
1.c. Informasi wajib spesies	13519190	13519190
1.d. Level up engimon	13519199	13519190
1.e. Level max engimon	13519199	13519190
1.f. 1 jenis spesies untuk tiap element	13519206	13519214
2.a. Jenis skill	13519199	13519211
2.b. Informasi wajib skill	13519211	13519211
2.c. Skill bawaan unik engimon	13519190	13519190
3.a. Commands player	13519190	13519199
3.b. Inventory player	13519211	13519206
3.c. Active engimon	13519206	13519211
4.a. Hitung power	13519211	13519211
4.b. Total power level	13519211	13519211
4.c. Element advantage	13519211	13519211
4.d. Kondisi engimon kalah	13519211	13519211
4.e. Kondisi engimon menang	13519211	13519211
4.f. Battle multiple element engimon	13519211	13519211
5.a. Level minimum breeding	13519206	13519214
5.b. Parent level setelah breeding	13519206	13519214
5.c. Nama anak hasil breeding	13519206	13519214
5.d. Inherit skill	13519206	13519214
5.e. Resulting child species & element	13519206	13519214

6.a. Spawn engimon dengan huruf kapital	13519199	13519214
6.b. Spawn engimon dengan huruf kecil	13519211	13519214
6.c. Tampilan berdasarkan element	13519190	13519190
6.d. Lingkungan peta	13519190	13519190
6.e. Tile player	13519214	13519214
6.f. Perpindahan engimon	13519214	13519214
6.g. Batas spawn engimon	13519214	13519214
6.h. Spawn engimon random	13519214	13519214
6.i. Perpindahan engimon random	13519214	13519214
6.j. Load peta	13519214	13519214
6.k. Kasus tepi	13519199	13519211
BONUS 1: Dual element breeding	13519199	13519206
BONUS 2: Purely Random Wild Engimon Generation	13519206	13519211
BONUS 3: Unit Testing Implementation	-	-
BONUS Kreasi Mandiri: Fog of war dan Line of Sight sederhana	13519214	13519214
BONUS Kreasi Mandiri: Penambahan Auto-tick increment	13519214	13519214
BONUS Kreasi Mandiri: Multi-threading dan mutex sederhana	13519214	13519214
BONUS Kreasi Mandiri: Penambahan Warna	13519190	13519190