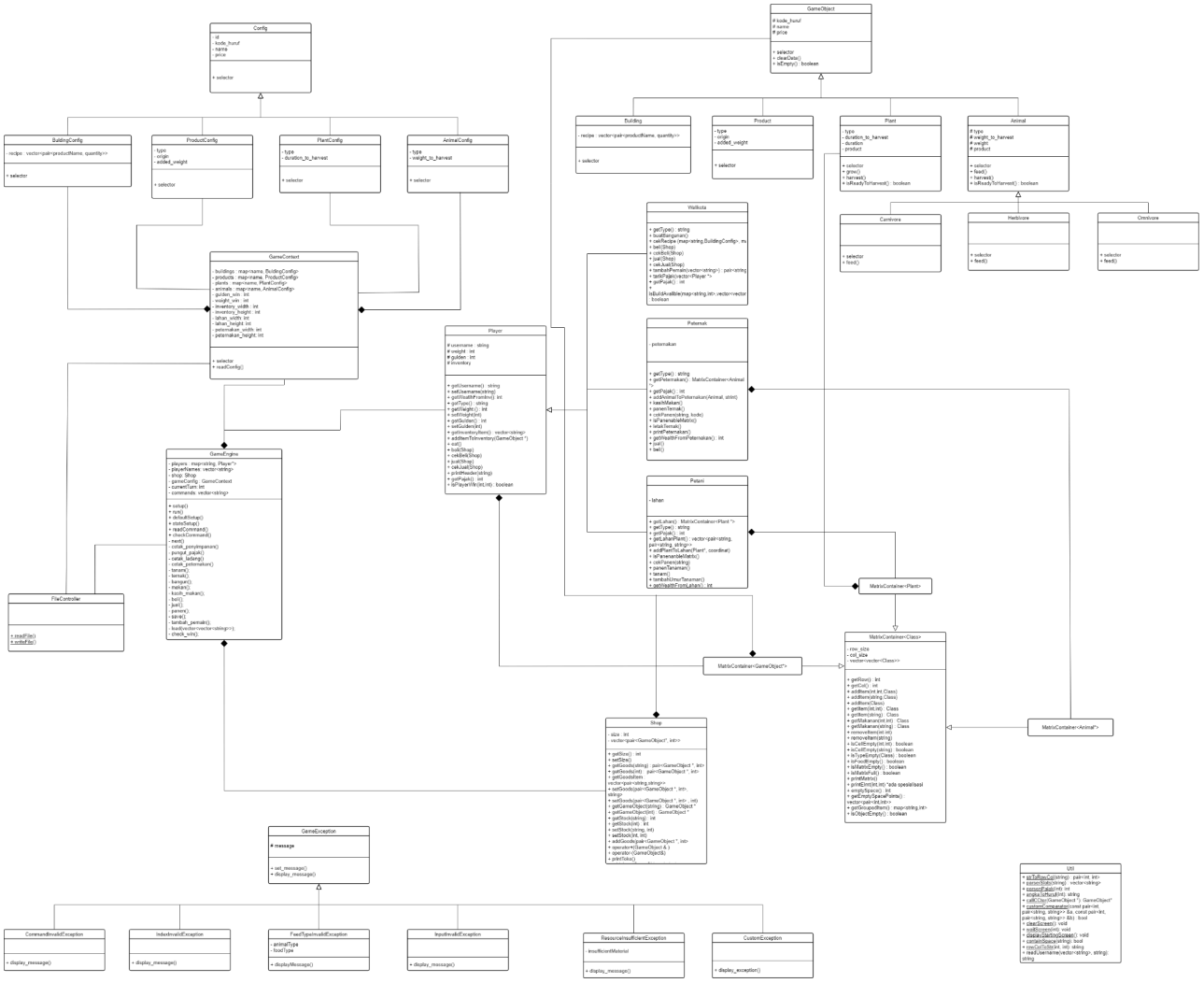
Kode Kelompok : HEY
Nama Kelompok : Hala_Emyu_YNWA

1. 10023637 / Zidan BSA
2. 13522054 / Benjamin Sihombing
3. 13522068 / Adril Putra Merin
4. 13522098 / Suthasoma Mahardhika Munthe
5. 13522110 / Marvin Scifo Y. Hutahaean
6. 13522118 / Berto Richardo Togatorop

Asisten Pembimbing : Primanda Adyatma Hafiz

1. Diagram Kelas



Gambar diatas adalah gambaran sekilas dari Diagram Kelas yang kita buat. Untuk melihat diagram lebih lengkap dan lebih detil, bisa dilihat di tautan berikut ini.

Kami memilih desain kelas diagram di atas setelah dilakukan beberapa modifikasi atas saran asisten pembimbing dan beberapa kebutuhan yang diperlukan saat proses implementasi. Hasil abstraksi dari deskripsi spesifikasi dikelompokkan menjadi beberapa kelompok kelas utama yaitu *GameObject*, *Player*, *Shop*, *MatrixContainer*, *Config*, *GameContext*, dan *GameEngine*.

Setelah itu, dengan menerapkan konsep OOP dengan beberapa batasan dan ketentuan sesuai spesifikasi kami menerapkan polimorfisme, *inheritance,* dan *dynamic binding* pada player yang terdiri dari Walikota, Petani, dan Peternak, dan pada *GameObject* yang terdiri dari *Building, Product, Plant,* dan *Animal*.

Animal kemudian diturunkan menjadi beberapa tipe sesuai jenis product yang dapat mereka makan. Ada pula beberapa fungsi yang di-*override* karena perilaku *Animal* bergantung sesuai jenisnya (Karnivora, Herbivora, dan Omnivora).

Kami merasakan kemudahan dalam membuat penyimpanan untuk setiap pemain yang dapat menampung beberapa objek yang berbeda jenis. Hal ini yang menyebabkan kami menurunkan objek yang yang dapat disimpan pada penyimpanan dari *GameObject* agar dapat menggunakan kemampuan polimorfisme dan *dynamic binding*. Hal yang sama kami lakukan pada kontainer yang menyimpan player saat ini.

Kendala yang pernah kami alami adalah beberapa kali perubahan besar-besaran karena kami tidak memanfaatkan semua konsep OOP yang tersedia sehingga desain yang kami buat melanggar batasan dan tidak sesuai dengan konsep OOP. Setelah melakukan beberapa kali iterasi perbaikan desain, kami memilih desain di atas.

2. Penerapan Konsep OOP

2.1. Inheritance & Polymorphism

Pada program yang dibuat, terdapat beberapa kelas atau objek yang perlu melakukan penurunan (Inheritance). Hal ini dilakukan karena terdapat beberapa kelas yang definisinya sama dengan kelas lain tetapi hanya terdapat tambahan definisi yang membuat kelas itu berbeda dengan kelas parent-nya. Contoh kelas tersebut adalah Objek Permainan (Game Object), Konteks Permainan (Game Context), dan Pemain (Player).

2.1.1. Objek Permainan (Game Object)

Pada permainan yang dibuat, Game Object terdiri dari hewan, tanaman, produk, dan resep bangunan. Keempat hal tersebut memiliki atribut yang mirip dan tambahan atribut yang berbeda dengan satu sama lain. Berikut adalah cuplikan kode dari Konsep Inheritance dan Polymorphism yang diimplementasikan pada program.

1. Definisi ‘GameObject’

   GameObject memiliki derived class yang terdiri dari Animal, Plant, Product, dan Building yang mempunyai atribut yang sama dan ditambah atribut tambahan dan method tambahan yang cukup berbeda. Dengan melakukan inheritance dari GameObject, polymorphism terhadap derived classnya dapat tecapai.

```
// GameObject.hpp
#ifndef _GAMEOBJECT_HPP_
#define _GAMEOBJECT_HPP_

#include <iostream>
using namespace std;

class GameObject { // Kelas Parent
    protected:
        string kodeHuruf;
        string name;
        int price;
    public:
        GameObject();
        GameObject(const string &kodeHuruf_, const string &name_, int price_);
        GameObject(const GameObject& other);
        GameObject& operator=(const GameObject& other);
        virtual ~GameObject();
        string getKodeHuruf() const;
        void setKodeHuruf(const string &kodeHuruf_);
        string getName() const;
        void setName(const string & name_);
        int getPrice() const;
        void setPrice(int price_);
        bool isEmpty() const;
```

```
        void clearData();
};

#endif
```

2. Definisi ‘Animal’

Animal merupakan parent class dari Carnivore, Omnivore, dan Herbivore. Dengan inheritance, polymorphisme untuk method feed dapat tercapai.

```
// Animal.hpp (Cuplikan class Animal saja)
#ifndef _ANIMAL_HPP_
#define _ANIMAL_HPP_

#include "GameObject.hpp"
#include "Product.hpp"
#include <vector>
#include "../../GameException/header/GameException.hpp"

class Animal : public GameObject { // Inheritance GameObject
protected:
    string type;
    int harvestWeight;
    int weight;
    vector<Product> products;

public:
    Animal();
    Animal(
        const string& kodeHuruf_,
        const string& name_,
        int price_,
        const string& type_,
        int harvestWeight_,
        int weight_,
```

```
        const vector<Product>& products_
    );
    Animal(const Animal& other);
    Animal& operator=(const Animal& other);
    ~Animal();
    string getType() const;
    void setType(const string &type_);
    int getHarvestWeight() const;
    void setHarvestWeight(int harvestWeight_);
    int getWeight() const;
    void setWeight(int weight_);
    vector<Product> getProduct() const;
    void setProduct(const vector<Product> &products_);
    virtual void feed(const Product& food) = 0;
    virtual void operator+=(const Product& food) = 0;
    bool isReadyToHarvest();
    vector<Product> harvest();
};
.
. // Lanjutan Implementasi Kelas Lainnya
.
#endif
```

```
// Animal.hpp (class Carnivore)
class Carnivore: public Animal { // Inheritance Animal
public:
    Carnivore();
    Carnivore(
        const string& kodeHuruf_,
        const string& name_,
        int price_,
        const string& type_,
        int harvestWeight_,
        int weight_,
        const vector<Product>& products_
    );
```

```
    Carnivore(const Carnivore& other);
    Carnivore& operator=(const Carnivore& other);
    ~Carnivore();
    void feed(const Product &food);
    void operator+=(const Product &food);
};
```

```
class Herbivore: public Animal { // Inheritance Animal
public:
    Herbivore();
    Herbivore(
        const string& kodeHuruf_,
        const string& name_,
        int price_,
        const string& type_,
        int harvestWeight_,
        int weight_,
        const vector<Product>& products_
    );
    Herbivore(const Herbivore& other);
    Herbivore& operator=(const Herbivore& other);
    ~Herbivore();
    void feed(const Product &food);
    void operator+=(const Product &food);
};
```

```
class Omnivore: public Animal { // Inheritance Animal
public:
    Omnivore();
    Omnivore(
        const string& kodeHuruf_,
        const string& name_,
        int price_,
```

```
        const string& type_,
        int harvestWeight_,
        int weight_,
        const vector<Product>& products_
    );
    Omnivore(const Omnivore& other);
    Omnivore& operator=(const Omnivore& other);
    ~Omnivore();
    void feed(const Product &food);
    void operator+=(const Product &food);
};
```

3. Definisi ‘Plant’

```
// Plant.hpp
#ifndef _PLANT_HPP_
#define _PLANT_HPP_

#include "GameObject.hpp"
#include "Product.hpp"

class Plant: public GameObject { // Inheritance GameObject
private:
    string type;
    int harvestDuration;
    int duration;
    Product product;

public:
    Plant();
    Plant(const string& kodeHuruf_, const string& name_, int price_, const string& type_,
int harvestDuration_, int duration_, const Product& product_);
    Plant(const Plant& other);
    Plant& operator=(const Plant& other);
    ~Plant();
    string getType();
```

```
    void setType(const string &type_);
    int getHarvestDuration() const;
    void setHarvestDuration(int harvestDuration_);
    int getDuration() const;
    void setDuration(int duration_);
    Product getProduct() const;
    void setProduct(const Product &product_);
    bool isReadyToHarvest();
    Product harvest();
    void grow();
    void operator++(int);
};
#endif
```

4. Definisi ‘Building’

```
// Building.hpp
#ifndef _BUILDING_HPP_
#define _BUILDING_HPP_

#include "GameObject.hpp"
#include <vector>

class Building: public GameObject { // Inheritance GameObject
private:
    vector<pair<string,int>> recipe;

public:
    Building();
    Building(const string& kodeHuruf_, const string& name_, int price_, const
vector<pair<string, int>> &recipe_);
    Building(const Building& other);
    Building& operator=(const Building& other);
    ~Building();
    vector<pair<string,int>> getRecipe() const;
    void setRecipe(const vector<pair<string,int>> &recipe_);
```

```
};
#endif
```

5. Definisi ‘Product’

```
// Product.hpp
#ifndef _PRODUCT_HPP_
#define _PRODUCT_HPP_

#include "GameObject.hpp"

class Product: public GameObject { // Inheritance GameObject
private:
    string type;
    string origin;
    int addedWeight;

public:
    Product();
    Product(const string& kodeHuruf_, const string& name_, int price_, const string& type_,
const string& origin_, int addedWeight_);
    Product(const Product& other);
    Product& operator=(const Product& other);
    ~Product();
    string getType() const;
    void setType(const string &type_);
    string getOrigin() const;
    void setOrigin(const string &origin_);
    int getAddedWeight() const;
    void setAddedWeight(int addedWeight_);
};
#endif
```

2.1.2. Konteks Permainan (Game Context)

Pada permainan yang dibuat, Game Context terdiri dari konfigurasi hewan, konfigurasi tanaman, konfigurasi produk, dan konfigurasi resep bangunan. Keempat hal tersebut memiliki atribut yang mirip dan tambahan atribut yang berbeda dengan satu sama lain. Berikut adalah cuplikan kode dari Konsep Inheritance dan Polymorphism yang diimplementasikan pada program.

1. Definisi ‘Config’

```
// Config.hpp
#ifndef CONFIG_HPP
#define CONFIG_HPP

#include <iostream>
#include <string>
using namespace std;

class Config { // Kelas Parent
    protected:
        int id;
        string kode_huruf;
        string nama;
        int price;

    public:
        Config();
        Config(int, string, string, int);
        int getID() const;
        void setID(int);
        string getKodeHuruf() const;
        void setKodeHuruf(string);
        string getNama() const;
        void setNama(string);
        int getPrice() const;
        void setPrice(int);
};
#endif
```

2. Definisi ‘AnimalConfig’

```
// AnimalConfig.hpp
#ifndef ANIMALCONFIG_HPP
#define ANIMALCONFIG_HPP

#include <iostream>
#include "Config.hpp"
using namespace std;

class AnimalConfig : public Config { // Inheritance Config
    private:
        string type;
        int weight_to_harvest;

    public:
        AnimalConfig();
        AnimalConfig(int id, string kode_huruf, string nama, int price, string type, int
weight_to_harvest);
        string getType() const;
        void setType(string);
        int getWeightToHarvest() const;
        void setWeightToHarvest(int);
       void setAll(int,string,string,int,string,int);
};

#endif
```

3. Definisi ‘PlantConfig’

```
// PlantConfig.hpp
#ifndef PLANTCONFIG_HPP
#define PLANTCONFIG_HPP
```

```
#include <iostream>
#include "Config.hpp"
using namespace std;

class PlantConfig : public Config { // Inheritance Config
    private:
        string type;
        int duration_to_harvest;

    public:
        PlantConfig();
        PlantConfig(int,string,string,int,string,int);
        string getType() const;
        void setType(string);
        int getDurationToHarvest() const;
        void setDurationToHarvest(int);
       void setAll(int,string,string,int,string,int);
};
#endif
```

4. Definisi ‘BuildingConfig’

```
// BuildingConfig.hpp
#ifndef BUILDINGCONFIG_HPP
#define BUILDINGCONFIG_HPP

#include <iostream>
#include <vector>
#include <map>
#include "Config.hpp"
using namespace std;

class BuildingConfig : public Config { // Inheritance Config
    private:
        vector<pair<string,int>> recipe;
```

```
    public:
        BuildingConfig();
        BuildingConfig(int id, string kode_huruf, string nama, int price,
vector<pair<string,int>> recipe);
        vector<pair<string,int>> getRecipe() const;
        void setRecipe(const vector<pair<string,int>>);
        pair<string,int> getRecipeComp(int) const;
        void setRecipeComp(pair<string,int>,int);
        string getProductName(int) const;
        void setProductName(string,int);
        int getQuantity(int) const;
        void setQuantity(int,int);
        void setAll(int,string,string,int,vector<pair<string,int>>);
};

#endif
```

5. Definisi ‘ProductConfig’

```
// ProductConfig.hpp
#ifndef PRODUCTCONFIG_HPP
#define PRODUCTCONFIG_HPP

#include <iostream>
#include "Config.hpp"
using namespace std;

class ProductConfig : public Config { // Inheritance Config
    private:
        string type;
        string origin;
        int added_weight;

    public:
        ProductConfig();
        ProductConfig(int,string,string,int,string,string,int);
```

```
        void setType(string);
        string getOrigin() const;
        void setOrigin(string);
        int getAddedWeight() const;
        void setAddedWeight(int);
        void setAll(int,string,string,int,string,string,int);
};

#endif
```

2.1.3. Pemain (Player)

Pada permainan yang dibuat, Player terdiri dari petani, peternak, dan walikota. Ketiga hal tersebut memiliki atribut yang mirip dan tambahan atribut yang berbeda dengan satu sama lain. Berikut adalah cuplikan kode dari Konsep Inheritance dan Polymorphism yang diimplementasikan pada program.

1. Definisi ‘Player’

```
#ifndef _PLAYER_HPP_
#define _PLAYER_HPP_

using namespace std;
#include <iostream>
#include <string>
#include <map>
#include <cmath>
#include "../../Container/header/MatrixContainer.hpp"
#include "../../Shop/header/Shop.hpp"
#include "../../GameObject/header/GameObject.hpp"
#include "../../GameObject/header/Animal.hpp"
#include "../../GameObject/header/Plant.hpp"
#include "../../GameObject/header/Product.hpp"
```

```
#include "../../GameObject/header/Building.hpp"
#include "../../GameException/header/GameException.hpp"
#include "../../Util/header/Util.hpp"

class Player { // Kelas Parent
protected:
    string username;
    int weigth;
    int gulden;

public:
    MatrixContainer<GameObject *> inventory;
    Player(string username_, int invRow, int invCol);
    ~Player();
    string getUsername() const;
    void setUsername(string username_);
    int getWealthFromInv() const;
    virtual string getType() const;
    int getWeight() const;
    void setWeight(int weight_);
    int getGulden() const;
    void setGulden(int gulden_);
    void eat();
    virtual void jual(Shop &toko);
    virtual void beli(Shop &toko );
    virtual void cekBeli(Shop &toko);
    virtual void cekJual(Shop &toko);
    void printHeader(string judul);
    void printInventory();
    virtual int getPajak() const;
    bool isPlayerWin(int guldenToWin, int weigthToWin);
};
#endif
```

2. Definisi ‘Petani’

```
// Petani.hpp
#ifndef _PETANI_HPP_
#define _PETANI_HPP_

#include "Player.hpp"
#include "../../GameObject/header/Plant.hpp"
#include <set>
class Petani : public Player { // Inheritance Player
private:
public:
    MatrixContainer<Plant *> lahan;
    Petani(string username_, int invRow, int invCol, int lahanRow, int lahanCol);
    ~Petani();
    MatrixContainer<Plant *> getLahan() const;
    string getType() const;
    int getPajak() const;
    bool isPanenableMatrix();
    void cekPanen(string cell);
    void panenTanaman();
    void tanam();
    void tambahUmurTanaman();
    int getWealthFromLahan() const;
    void printLahan();
};
#endif
```

3. Definisi ‘Peternak’

```
// Peternak.hpp
#ifndef _PETERNAK_HPP_
#define _PETERNAK_HPP_

#include <set>
#include "Player.hpp"
#include "../../GameObject/header/Animal.hpp"
class Peternak : public Player { // Inheritance Player
```

```
private:
public:
    MatrixContainer<Animal *> peternakan;
    Peternak(string username_,
             int invRow, int invCol,
             int peternakanRow, int peternakanCol);
    ~Peternak();
    string getType() const;
    MatrixContainer<Animal *> getPeternakan() const;
    int getPajak() const;
    void kasihMakan();
    void panenTernak();
    void cekPanen(string cell);
    bool isPanenableMatrix();
    void letakTernak();
    void printPeternakan();
    int getWealthFromPeternakan() const;
};
#endif
```

4. Definisi ‘Walikota’

```
// Walikota.hpp
#ifndef _WALIKOTA_HPP_
#define _WALIKOTA_HPP_

#include "Player.hpp"
#include "../../GameContext/header/BuildingConfig.hpp"
#include "../../GameContext/header/Config.hpp"
#include "../../GameObject/header/Building.hpp"
#include <algorithm>
#include <set>
#include <algorithm>
#include <queue>
using namespace std;
```

```
class Walikota : public Player { // Inheritance Player
private:
public:
    Walikota(string username_, int invRow, int invCol);
    string getType() const;
    void tarikPajak(vector <Player *> &players);
    void buatBangunan(const map<string, BuildingConfig> &);
    void cekRecipe(const map<string, BuildingConfig> &building, map<string, int>
&materials);
    void beli(Shop &toko);
    void cekBeli(Shop &toko);
    void jual(Shop &toko);
    void cekJual(Shop &toko);
    pair<string, string> tambahPemain(vector<string> &setNama);
    void tarikPajak(vector<Player *> &listOfPlayers);
    int getPajak() const;
    vector<pair<string, int>> getInsufficientMaterial(
        map<string, int> &materials,
        const pair<vector<pair<string, int>>, int> &recipe);
    bool isBuildAvailable(
        map<string, int> &materials,
        const vector<pair<vector<pair<string, int>>, int>> &listOfRecipes);
    pair<string, string> tambahPemain(vector<string> &setNama);
};

#endif
```

2.1.4. Exception

Setiap Exception yang dibuat mempunyai pesan yang berbeda-beda dan cara penggunaan yang berbeda-beda juga. Oleh karena itu, perlu dibuat kelas Exception dan kelas turunan dari Exception tersebut berdasarkan pesan dan alasan penggunaan Exception tersebut.

1. GameException (Parent Class)

```
class GameException { // Parent Class
protected:
    string message;

public:
    GameException();
    GameException(const string &);
    virtual ~GameException();
    void setMessage(const string &);
    virtual void displayMessage() const = 0;
};
```

2. FeedTypeInvalidException

```
class FeedTypeInvalidException : public GameException { // Inheritance dari GameException
private:
    string animalType;
    string productType;

public:
    /**
     * Menciptakan GameException dengan pesan
     * "Jenis Product tidak dapat dimakan"
     * - disimpan pula jenis hewan dan product yang akan dimakan
     * @param animalType_ jenis hewan yang diberi makan
     * @param productType_ jenis product pakan
     */
    FeedTypeInvalidException(const string &, const string &);

    void displayMessage() const;
};
```

3. IndexInvalidException

```
class IndexInvalidException : public GameException { // Inheritance GameException

public:
    /**
     * Menciptakan GameException dengan pesan
     * "Index Invalid"
     */
    IndexInvalidException();

    void displayMessage() const;
};
```

4. ResourceInsufficientException

```
class ResourceInsufficientException : public GameException { // Inheritance GameException
private:
    vector<pair<string, int>> insufficientMaterial;

public:
    /**
     * Menciptakan Exception ketika walikota tidak bisa membangun bangunan
     * karena sumber daya tidak cukup
     * @param
     */
    ResourceInsufficientException(const vector<pair<string, int>> &insufficientMaterial_);

    void displayMessage() const;
};
```

5. InputInvalidException

```
class InputInvalidException : public GameException { // Inheritance GameException
public:
```

```
    /**
     * Exception saat input invalid
     * @param
     */
    InputInvalidException();

    void displayMessage() const;
};
```

6. CommandInvalidException

```
class CommandInvalidException : public GameException { // Inheritance GameException
public:
    /**
     * Exception saat command invalid atau tidak tersedia
     * @param
     */
    CommandInvalidException();

    void displayMessage() const;
};
```

7. CustomException

```
class CustomException : public GameException { // Inheritance GameException
public:
    /**
     * Menciptakan GameException yang disesuaikan dengan
     * kesalahan tertentu
     * Hanya dapat dilakukan pada pesan sebuah kalimat saja
     */
```

```
    CustomException(const string &message_);
    ~CustomException();
    void displayMessage() const;
};
```

Inheritance dan Polymorphism ini digunakan agar kelas yang dibuat tidak perlu mendefinisikan atribut yang terbilang sama berkali kali dan hanya menggunakan atribut yang dimiliki oleh kelas lain saja sehingga hanya menambah beberapa atribut saja.

2.2. Method/Operator Overloading

2.2.1. Method Overloading

Method overloading digunakan agar method dengan nama yang sama dapat menerima argumen dengan tipe yang berbeda.

Method overloading dilakukan pada beberapa method seperti berikut

1. addItem dari Class MatrixContainer
   - addItem(int,int,Class)
   - addItem(string, Class)
   - addItem(Class)
2. getItem dari Class MatrixContainer
   - getItem(int,int)
   - getItem(string)
3. getMakanan dari Class MatrixContainer
   - getMakanan(string)
   - getMakanan(int,int)
4. removeItem dari Class MatrixContainer
   - removeItem(int,int)
   - removeItem(string)
5. isCellEmpty dari Class MatrixContainer

- isCellEmpty(int,int)
- isCellEmpth(string)

6. getGoods dari Class Shop
   - getGoods(string)
   - getGoods(int)
7. setGoods dari Class Shop
   - setGoods(string)
   - setGoods(int)
8. getGameObject dari Class Shop
   - getGameObject(string)
   - getGameObject(int)
9. getStock dari Class Shop
   - getStock(int)
   - getStock(string)

2.2.2. Operator Overloading

Untuk mempermudah penggunaan program dari kelas lain, biasanya pada kelas tersebut dibuat operator overloading untuk mempermudah pembacaan program dan tidak perlu menulis nama program yang ada di kelas tersebut. Operator overloading yang digunakan adalah +, +=, ++, ==, [].

1. Operator+

   1.1. Penggunaan pada shop

```
// Implementasi Operator+
// Operator+ pada Shop akan menambahkan stock barang di toko jika reference objek yang
// dipassing memiliki tipe yang sama dengan yang ada di toko dan merupakan objek yang
```

```
// finite (bukan animal ataupun plant)

void Shop::operator+(const GameObject &obj)
{
    for (int i = 0; i < size; i++)
    {
        // Avoid add stock for unlimited object
        Plant *plant = dynamic_cast<Plant *>(getGameObject(i));
        Animal *animal = dynamic_cast<Animal *>(getGameObject(i));

        // add for limited object
        if (plant == NULL && animal == NULL && (*getGameObject(i)) == obj)
        {
            content[i].second = getStock(i) + 1;
        }
    }
}
```

```
// Penggunaan pada program

// Bagian dari cek beli
for (int i = 0; i < (int)slotS.size(); i++)
{
    // cek bangunan
    if (dynamic_cast<Building *>(inventory.getItem(slotS[i])) != NULL)
    {
        throw CustomException("Tidak bisa menjual bangunan");
    }

    // tambah stock jika barang yang dibeli finite
    toko + (*inventory.getItem(slotS[i])); // Dengan Operator+, penambahan operasi
                                           penambahan stock untuk barang yang sejenis
                                           menjadi lebih ringkas

    // setGulden
    uangTambah += inventory.getItem(slotS[i])->getPrice();
    setGulden(getGulden() + inventory.getItem(slotS[i])->getPrice());
```

```
    vectorTemp.push_back({inventory.getItem(slotS[i]), slotS[i]});

    // remove dari inventory
    inventory.removeItem(slotS[i]);
}
```

1.2. Penggunaan pada Matrix

```
/** * Operator+ untuk menambahkan item ke cell kosong pertama yang ditemukan */

void operator+(const T &item)
{
    addItem(item);
}

void addItem(int r, int c, const T &item)
    {
        // handle out of idx
        if (r >= rowSize || c >= colSize || r < 0 || c < 0)
        {
            throw IndexInvalidException();
        }
        else
        {
            if (isCellEmpty(r, c))
            {
                buffer[r][c] = item;
            }
            else
            {
                throw CustomException("Cell/petak sudah terisi");
            }
        }
    }
```

```
// Penggunaan pada saat petani memanen
Plant *plant = lahan.getItem(cell);
inventory + (new Product(plant->harvest())); // Product hasil panen yang baru
                                                diinstansiasi akan
                                                dimasukkan ke cell inventory yang kosong
lahan.removeItem(cell);
delete plant;

// Penggunaan pada saat peternak memanen
vector<Product> products = animal->harvest();
for (auto product : products)
{
    inventory + (new Product(product));       // Product hasil panen yang baru
                                                 diinstansiasi akan dimasukkan ke cell
                                                 inventory kosong yang pertama ditemukan
    cout << product.getKodeHuruf() << endl;
}


// Penggunaan pada saat walikota membangun bangunan
BuildingConfig config = buildings.at(input);
Building *bangunan = new Building(config.getKodeHuruf(), config.getNama(),
   config.getPrice(), config.getRecipe());

inventory + bangunan; // inventory akan dimasukkan atau ditambahkan dengan objek bangunan
```

2. Operator+=

```
// Implementasi Operator+=
void Carnivore::operator+=(const Product &food)
{
    this->feed(food);
```

```
}
```

```
// Penggunaan pada program (Dikutip dari Peternak.cpp)
void Peternak::kasihMakan()
{

    .
    .
    .
    while (product == NULL)
    {
        try
        {
            cout << "Slot: ";
            cin >> invCoordinate;
            cin.ignore();
            cout << '\n';

            product = dynamic_cast<Product *>(inventory.getItem(invCoordinate));
        }
        catch (const GameException &e)
        {
            e.displayMessage();
        }
    }

    (*animal) += (*product); // Objek *animal ditambahi dengan variabel objek *product.
                             Dengan menggunakan Operator overloading, pada animal, akan
                             dieksekusi fungsi feed yang menambah berat badan dari
                             objek tersebut.

    .
    .
    .

}
```

3. Operator++

```
// Implementasi Operator++
void Plant::operator++(int)
{
    grow();
}
```

```
// Penggunaan pada program (Dikutip dari GameEngine.cpp)
void Petani::tambahUmurTanaman()
{
    for (int i = 0; i < lahan.getRow(); i++)
    {
        for (int j = 0; j < lahan.getCol(); j++)
        {
            if (!lahan.isCellEmpty(i, j))
            {
                (*lahan.getItem(i, j))++; // Pada lahan, salah satu item yang dengan kelas
                                          Plant diambil lalu menggunakan operator++
                                          yang merupakan fungsi grow. Fungsi grow
                                          bertugas menambah atribut duration pada plant
                                          dengan 1
            }
        }
    }
}
```

4. Operator==

Operator== digunakan untuk membandingkan 2 objek sejenis. Jika kedua objek sama, akan mengembalikan *true*.

```
// Implementasi Operator==
bool GameObject::operator==(const GameObject& other)
{
    return kodeHuruf == other.kodeHuruf && name == other.name && price == other.price;
}
```

```
// Penggunaan pada program (Dikutip dari Shop.cpp)
```

```
void Shop::addGoods(pair<GameObject *, int> goods)
{
    for (int i = 0; i < (int)content.size(); i++)
    {
        if (content[i].first == goods.first) // Kedua objek dengan kelas GameObject
                                             dibandingkan seluruh atributnya. Fungsi
                                             operator== mengembalikan true jika semua
                                             atribut sama dengan atribut objek lain dan
                                             mengembalikan false jika tidak
        {
            setStock(goods.first->getName(), getStock(goods.first->getName()) +
goods.second);
            break;
        }
    }
    size++;
    content.push_back(goods);
}
```

5. Operator []

Operator[] digunakan untuk mendapatkan GameObject dalam shop yang terletak pada index tertentu. Misalnya, Shop[i] akan mengembalikan GameObject yang terletak di *shop* pada index ke-*i*.

```
// Implementasi Operator []
GameObject *Shop::operator[](int idx) const
{
    return getGameObject(idx);
}
```

```
// Penggunaan pada program (Dikutip dari Player.cpp)
void Player::cekBeli(Shop &toko)
{
    // Minta masukan dari user
```

```
    int masukan;

    while (cout << "Barang yang ingin dibeli : " && !(cin >> masukan))
    {
        cin.clear();                                          // clear bad input flag
        cin.ignore(numeric_limits<streamsize>::max(), '\n'); // discard input
        cout << "Input invalid (bukan integer), mohon masukkan input kembali.\n";
    }
    cin.ignore();

    cout << endl;

    // validasi masukan
    if (masukan < 1 || masukan > toko.getSize())
    {
        throw CustomException("Input tidak valid.");
    }

    // validasi penyimpanan
    int quantity;

    while (cout << "Kuantitas : " && !(cin >> quantity))
    {
        cin.clear();                                          // clear bad input flag
        cin.ignore(numeric_limits<streamsize>::max(), '\n'); // discard input
        cout << "Input invalid (bukan integer), mohon masukkan input kembali.\n";
    }
    cin.ignore();

    cout << endl;

    if (quantity <= 0 || quantity > inventory.emptySpace())
    {
        throw CustomException("Kuantitas tidak valid atau melebihi kapasitas.");
    }

    // validasi stock di toko
    if (toko.getStock(masukan - 1) != -1 && quantity > toko.getStock(masukan - 1))
```

```
    {
        throw CustomException("Stok barang yang Anda pilih tidak cukup.");
    }

    // validasi gulden
    if (quantity * toko[masukan - 1]->getPrice() > getGulden())
    {
        throw CustomException("Gulden Anda tidak cukup.");
    }

    // Berikan IO dan kurangi uang
    setGulden(getGulden() - quantity * toko[masukan - 1]->getPrice());
    cout << "Selamat Anda berhasil membeli " << quantity << " " << toko[masukan -
1]->getName() << ".\n";
    cout << "Uang Anda tersisa " << getGulden() << endl;

    // IO pemilihan slot
    cout << "Pilih slot untuk menyimpan barang yang Anda beli!" << endl;
    printInventory();

    // Pilih slot
    bool isDone = false;
    while (!isDone)
    {
        string inSlot;
        cout << "Petak slot: ";
        getline(cin, inSlot);
        vector<string> slotS = Util::parserSlots(inSlot);
        int cnt = 0;
        try
        {
            // validasi jumlah slot
            if ((int)slotS.size() != quantity)
            {
                throw CustomException("Jumlah Slot tidak valid");
            }

            // eksekusi
```

```
            for (int i = 0; i < (int)slotS.size(); i++)
            {
                GameObject *item = Util::callCCtor(toko[masukan - 1]);
                inventory.addItem(slotS[i], item);
                cnt++;
            }

            // kurangi stock jika barang finite
            if (dynamic_cast<Plant *>(toko[masukan - 1]) == NULL && dynamic_cast<Animal
*>(toko[masukan - 1]) == NULL)
            {
                toko.setStock(toko[masukan - 1]->getName(), toko.getStock(masukan - 1) -
quantity);
            }
            isDone = true;
        }
        catch (const GameException &e)
        {
            // batal beli
            for (int i = 0; i < cnt; i++)
            {
                inventory.removeItem(slotS[i]);
            }
            e.displayMessage();
        }
    }
}
```

2.3. Template & Generic Classes

Terkadang ada beberapa objek yang definisi dan fungsinya sama tetapi terdapat satu atau lebih yang atributnya berbeda. Agar tidak perlu membuat kelas yang berbeda, C++ menyediakan fitur template yaitu fitur yang bisa menerima semua jenis kelas. Pada program ini, Template & Generic Classes digunakan pada MatrixContainer. Peternakan, Ladang, dan Penyimpanan merupakan hal yang sama namun dengan isi yang berbeda. Disinilah template dan generic bisa digunakan untuk menyatukan ketiga hal tadi menjadi satu yaitu MatrixContainer. Kami juga menggunakan template yang disediakan oleh STL C++ seperti vector<Class>, map<Class1,Class2>, pair<Class1,Class2>.

2.3.1. MatrixContainer<Class>

Class MatrixContainer adalah sebuah *generic class* yang menyimpan berbagai GameObject* dalam matriks.

```
// MatrixContainer.hpp
#ifndef _MATRIXCONTAINER_HPP_
#define _MATRIXCONTAINER_HPP_

using namespace std;
#include <iostream>
#include <string>
#include <sstream>
#include <iomanip>
#include <map>
#include <vector>
#include "../../Util/header/Util.hpp"
#include "../../GameObject/header/Animal.hpp"
#include "../../GameObject/header/Plant.hpp"
#include "pcolor.hpp"

template <class T> // Pendefinisian dari Template (Class T)
class MatrixContainer
{
private:
    int rowSize;
    int colSize;
    vector<vector<T>> buffer; // T merujuk pada kelas template

public:
    MatrixContainer()
    { // DEFINISI }
    MatrixContainer(int r, int c)
    { // DEFINISI }
    ~MatrixContainer()
    { // DEFINISI }
    int getRow() const
    { // DEFINISI }
    int getCol() const
```

```
{ // DEFINISI }
void addItem(int r, int c, const T &item) // Parameter bisa kelas apa saja asal bisa
                                            disesuaikan dengan apa yang diinginkan program
{ // DEFINISI }
void addItem(string koordinat, T item) // Parameter bisa kelas apa saja asal bisa
                                         disesuaikan dengan apa yang diinginkan program
{ // DEFINISI }

void addItem(const T &item) // Parameter bisa kelas apa saja asal bisa
                              disesuaikan dengan apa yang diinginkan program
{ // DEFINISI }

T getItem(int r, int c) const // Parameter mengembalikan nilai T yaitu kelas template
{ // DEFINISI }

T getItem(string koordinat) const // Parameter mengembalikan nilai T yaitu kelas template
{ // DEFINISI }

T getMakanan(int r, int c) // Parameter mengembalikan nilai T yaitu kelas template
{ // DEFINISI }

T getMakanan(string koordinat) // Parameter mengembalikan nilai T yaitu kelas template
{ // DEFINISI }

void removeItem(int r, int c)
{ // DEFINISI }

void removeItem(string koordinat)
{ // DEFINISI }

bool isCellEmpty(int r, int c) const
{ // DEFINISI}


bool isCellEmpty(string koordinat)
{ // DEFINISI }

bool isTypeEmpty(T obj) // Parameter mengembalikan nilai T yaitu kelas template
```

```
    { // DEFINISI }

    bool isFoodEmpty()
    { // DEFINISI }
.
.
.
.
.

#endif
```

```
// MatrixContainer.cpp
#include "../header/MatrixContainer.hpp"

template <>
void MatrixContainer<GameObject *>::printElmt(int r, int c) { // Program ekslusif untuk
                                                               MatrixContainer bertipe
                                                               GameObject*
    cout << buffer[r][c]->getKodeHuruf();
}

template <>
void MatrixContainer<Plant *>::printElmt(int r, int c) { // Program ekslusif untuk
                                                          MatrixContainer bertipe
                                                          Plant*
    if (buffer[r][c]->isReadyToHarvest()){
        print_greens(buffer[r][c]->getKodeHuruf());
    }
    else{
        print_reds(buffer[r][c]->getKodeHuruf());
    }
}

template <>
void MatrixContainer<Animal *>::printElmt(int r, int c) { // Program ekslusif untuk
```

```
                                                                    MatrixContainer bertipe
                                                                    Animal*
    if (buffer[r][c]->isReadyToHarvest()){
        print_greens(buffer[r][c]->getKodeHuruf());
    }
    else{
        print_reds(buffer[r][c]->getKodeHuruf());
    }
}
```

2.3.2. Generic Pada STL

Terdapat beberapa STL yang disediakan C++ yang menggunakan template class juga. Contohnya adalah sebagai berikut.

1. Vector

```
// Shop.hpp
#ifndef _SHOP_HPP_
#define _SHOP_HPP_

#include <iostream>
#include <vector> // Modul ini di-include agar vector bisa digunakan
#include <map>
using namespace std;

#include "../../GameObject/header/Building.hpp"
#include "../../GameObject/header/Plant.hpp"
#include "../../GameObject/header/Animal.hpp"

class Shop
{
```

```
private:
    int size;
    vector<pair<GameObject *, int>> content; // Vector menggunakan generic kelas dan yang
                                                didalamnya adalah kelas dari konten vector

public:
    /**
     * Default Constructor
     */
    Shop();
.
.
.
#endif
```

2. Pair

```
// BuildingConfig.hpp
#ifndef BUILDINGCONFIG_HPP
#define BUILDINGCONFIG_HPP

#include <iostream>
#include <vector>
#include <map> // Karena map adalah list berisi pair, pair sudah tersedia di modul ini
#include "Config.hpp"
using namespace std;

class BuildingConfig : public Config {
.
.
.
        pair<string,int> getRecipeComp(int) const; // pair memiliki dua value dengan kelas
                                                      yang berbeda yaitu string dan int

.
.
```

```
.
#endif
```

3. Map

```
#ifndef GAMECONTEXT_HPP
#define GAMECONTEXT_HPP

#include "AnimalConfig.hpp"
#include "BuildingConfig.hpp"
#include "PlantConfig.hpp"
#include "ProductConfig.hpp"
#include <iostream>
#include <map>
using namespace std;

class GameContext
{
private:
    map<string, AnimalConfig> animals; // Map adalah list pair (key & value). Key bertipe
                                       string dan value bertipe AnimalConfig
    map<string, BuildingConfig> buildings;
    map<string, PlantConfig> plants;
    map<string, ProductConfig> products;
    .
    .
    .

public:
.
.
.
    map<string, AnimalConfig> getAnimals() const; // Fungsi ini mengembalikan value
                                                     dari kelas map
.
.
```

```
.
#endif
```

2.4. Exception

Untuk menangani permasalahan error pada program, kami menggunakan fitur C++ yaitu throw dan try and catch lalu membuat kelas exception beserta turunannya. Hal ini dilakukan untuk menghindari penggunaan if-else statement dan langsung menangani exception yang ada pada program. Berikut adalah cuplikan kode dari penggunaan exception.

1. GameException (Parent Class)

```
// Pembuatan class (Dikutip dari GameException.hpp)
class GameException { // Parent Class
protected:
    string message;

public:
    GameException();
    GameException(const string &);
    virtual ~GameException();
    void setMessage(const string &);
    virtual void displayMessage() const = 0;
};
```

```
// Implementasi try and catch (Dikutip dari GameEngine.cpp)
try
        {
            string input;
            cout << "Apakah Anda ingin memuat state? (y/n): ";
            cin >> input;
            cin.ignore();
            cout << endl;

            if (input == "y" || input == "Y")
```

```
            {
                stateSetup();
            }
            else if (input == "n" || input == "N")
            {
                defaultSetup();
            }
            else
            {
                throw InputInvalidException();
            }
            isValid = true;
        }
        catch (const GameException &e) // GameException ditangkap agar bisa dihandle dan tidak membiarkan
                                       // syntax throw menutup program secara mendadak
        {
            e.displayMessage();
        }
    }
```

2. FeedTypeInvalidException

```
// Pembuatan class (Dikutip dari GameException.hpp)
class FeedTypeInvalidException : public GameException { // Inheritance dari GameException
private:
    string animalType;
    string productType;

public:
    /**
     * Menciptakan GameException dengan pesan
     * "Jenis Product tidak dapat dimakan"
     * - disimpan pula jenis hewan dan product yang akan dimakan
     * @param animalType_ jenis hewan yang diberi makan
     * @param productType_ jenis product pakan
```

```
     */
    FeedTypeInvalidException(const string &, const string &);

    void displayMessage() const;
};
```

```
// Implementasi Throw (Dikutip dari Animal.cpp)
void Herbivore::feed(const Product &food) {
    if (food.getType() != "PRODUCT_FRUIT_PLANT") {
        throw FeedTypeInvalidException(type, food.getType()); // Menghandle tipe yang salah dan memberi
                                                                perintah ke program untuk melempar pesan
                                                                error

    } else {
        weight += food.getAddedWeight();
    }
}
```

3. IndexInvalidException

```
// Pembuatan class (Dikutip dari GameException.cpp)
class IndexInvalidException : public GameException { // Inheritance GameException

public:
    /**
     * Menciptakan GameException dengan pesan
     * "Index Invalid"
     */
    IndexInvalidException();

    void displayMessage() const;
};
```

```
// Implementasi Throw (Dikutip dari MatrixContainer.hpp)
void addItem(int r, int c, const T &item)
    {
```

```
        // handle out of idx
        if (r >= rowSize || c >= colSize || r < 0 || c < 0)
        {
            throw IndexInvalidException(); // Jika index dari input tidak sesuai dengan ukuran dari matrix
                                              penyimpanan, ladang, atau peternakan, program melempar error
                                              index yang dimasukkan ialah invalid
        }
        else
        {
            if (isCellEmpty(r, c))
            {
                buffer[r][c] = item;
            }
            else
            {
                throw CustomException("Cell/petak sudah terisi");
            }
        }
    }
```

4. ResourceInsufficientException

```
// Pembuatan class (Dikutip dari GameException.hpp)
class ResourceInsufficientException : public GameException { // Inheritance GameException
private:
    vector<pair<string, int>> insufficientMaterial;

public:
    /**
     * Menciptakan Exception ketika walikota tidak bisa membangun bangunan
     * karena sumber daya tidak cukup
     * @param
     */
    ResourceInsufficientException(const vector<pair<string, int>> &insufficientMaterial_);
```

```
    void displayMessage() const;
};
```

```
// Implementasi Throw
void Walikota::cekRecipe(const map<string, BuildingConfig> &buildings, map<string, int> &materials)
{
    .
    .
    .

    if (insufficientMaterial.size() != 0)
    {
        throw ResourceInsufficientException(insufficientMaterial); // Jika salah satu jenis material yang
                                                                   dibutuhkan tidak cukup (List
                                                                   insufficientMaterial mempunyai panjang
                                                                   0), program akan melempar exception
                                                                   sumber daya tidak cukup.
    }

    .
    .
    .
}
```

5. InputInvalidException

```
// Pembuatan class (Dikutip dari GameException.hpp)
class InputInvalidException : public GameException { // Inheritance GameException
public:
    /**
     * Exception saat input invalid
     * @param
     */
    InputInvalidException();

    void displayMessage() const;
```

```
};
```

```
// Implementasi Throw (Dikutip dari GameEngine.cpp)
try
        {
            string input;
            cout << "Apakah Anda ingin memuat state? (y/n): ";
            cin >> input;
            cin.ignore();
            cout << endl;

            if (input == "y" || input == "Y")
            {
                stateSetup();
            }
            else if (input == "n" || input == "N")
            {
                defaultSetup();
            }
            else
            {
                throw InputInvalidException(); // Input selain Y or N (Boleh huruf kecil) akan membuat
                                                  program melempar error kesalahan memasukkan input
            }
            isValid = true;
        }
        catch (const GameException &e)
        {
            e.displayMessage();
        }
    }
```

6. CommandInvalidException

```
// Pembuatan class (Dikutip dari GameException.hpp)
```

```
class CommandInvalidException : public GameException { // Inheritance GameException
public:
    /**
     * Exception saat command invalid atau tidak tersedia
     * @param
     */
    CommandInvalidException();

    void displayMessage() const;
};
```

```
// Implementasi Throw (Dikutip dari GameEngine.cpp)
int GameEngine::checkCommand(string input)
{
    for (int i = 0; i < (int)commands.size(); i++)
    {
        if (input == commands[i])
        {
            return i;
        }
    }
    throw CommandInvalidException(); //
}
```

7. CustomException

```
// Pembuatan class (Dikutip dari GameException.hpp)
class CustomException : public GameException { // Inheritance GameException
public:
    /**
     * Menciptakan GameException yang disesuaikan dengan
     * kesalahan tertentu
     * Hanya dapat dilakukan pada pesan sebuah kalimat saja
     */
    CustomException(const string &message_);
```

```
    ~CustomException();
    void displayMessage() const;
};
```

```
// Implementasi Throw (Dikutip dari FileController.cpp)
vector<vector<string>> FileController::readFile(string path)
{
    vector<vector<string>> elements;

    fstream file;
    file.open(path, ios::in);
    if (file.is_open())
    {
        string line;
        while (getline(file, line))
        {
            vector<string> this_line;
            this_line = split(line, ' ');
            elements.push_back(this_line);
        }
        file.close();
    }
    else
    {
        throw CustomException("Lokasi berkas tidak valid T__T"); // Jika path untuk mencari file tidak
                                                                 ditemukan, program melempar exception
                                                                 dengan pesan "Lokasi berkas tidak valid
                                                                 T__T".
    }
    return elements;
}
```

2.5. C++ Standard Template Library

C++ menyediakan Standard Template Library agar pengguna tidak perlu lagi mengimplementasi kelas kelas seperti list dinamis, stack, queue, set, map, dll.

1. Vector (#include <vector>)

```
// Inklusi vektor (Dikutip dari BuildingConfig.hpp)
#ifndef BUILDINGCONFIG_HPP
#define BUILDINGCONFIG_HPP

#include <iostream>
#include <vector> // Modul vector diinklusi
#include <map>
#include "Config.hpp"
using namespace std;

class BuildingConfig : public Config {
    private:
        vector<pair<string,int>> recipe; // Instansiasi atribut bertipe vector

    public:
        vector<pair<string,int>> getRecipe() const; // Fungsi ini mengembalikan value vector
        void setRecipe(const vector<pair<string,int>>); // Prosedur ini memerlukan masukkan yang bertipe
                                                        vector
};

#endif
```

```
// Penggunaan vektor (Dikutip dari GameContext.cpp)
void GameContext::readConfig()
{
    vector<vector<string>> animal_list;
    vector<vector<string>> plant_list;
    vector<vector<string>> building_list;
    vector<vector<string>> product_list;
    vector<vector<string>> misc;
```

```
    // Instansiasi vector yang berisi vector juga (2-Dimensional vector)

    .
    .
    .

    // Assign animal to game context
    for (int i = 0; i < (int)animal_list.size(); i++)
    {
        temp_id = stoi(animal_list.at(i).at(0));
        temp_kode_huruf = animal_list.at(i).at(1);
        temp_nama = animal_list.at(i).at(2);
        temp_type = animal_list.at(i).at(3);
        temp_weight_to_harvest = stoi(animal_list.at(i).at(4));
        temp_price = stoi(animal_list.at(i).at(5));
        temp_animal.setAll(temp_id, temp_kode_huruf, temp_nama, temp_price, temp_type, temp_weight_to_harvest);
        this->animals.insert({temp_nama, temp_animal});
        // Penggunaan metode bawaan .at(int) untuk mengakses elemen vector
    }

    .
    .
    .
    .

}
```

2. Map & Pair (#include <map>)

```
// Inklusi map dan pair (Dikutip dari Shop.hpp)
#ifndef _SHOP_HPP_
#define _SHOP_HPP_

#include <iostream>
#include <vector>
#include <map> // Inklusi modul map (pair juga diinklusi di modul ini)
```

```
using namespace std;

#include "../../GameObject/header/Building.hpp"
#include "../../GameObject/header/Plant.hpp"
#include "../../GameObject/header/Animal.hpp"

class Shop
{
private:
    // ATTRIBUTE

public:
    // METHOD
};
#endif
```

```
// Instansiasi, fungsi, dan masukan dari map (Dikutip dari GameContext.hpp)
#ifndef GAMECONTEXT_HPP
#define GAMECONTEXT_HPP

#include "AnimalConfig.hpp"
#include "BuildingConfig.hpp"
#include "PlantConfig.hpp"
#include "ProductConfig.hpp"
#include <iostream>
#include <map>
using namespace std;

class GameContext
{
private:
    map<string, AnimalConfig> animals; // Atribut ini mempunyai tipe map
    map<string, BuildingConfig> buildings; // Atribut ini mempunyai tipe map
    map<string, PlantConfig> plants; // Atribut ini mempunyai tipe map
    map<string, ProductConfig> products; // Atribut ini mempunyai tipe map
    .
    .
    .
```

```
public:

    .
    .
    .

    map<string, AnimalConfig> getAnimals() const; // Fungsi ini mengembalikan value map
    void setAnimals(map<string, AnimalConfig>); // Prosedur ini membutuhkan variable map sebagai masukan

    .
    .
    .

};
#endif
```

```
// Instansiasi pair (Dikutip dari GameContext.cpp)
void GameContext::readConfig()
{
    .
    .
    .

    pair<string, int> temp_comp; // Instansiasi pair dari string dan integer

    .
    .
    .
}
```

2.6. Konsep OOP lain

Ada beberapa konsep OOP lain yang kami gunakan selain kelima konsep yang diatas. Contohnya adalah composition, abstract base class, dan static class. Composition digunakan untuk menghubungkan kelas entity ke kelas controller (Contohnya dari kelas GameContext ke kelas GameEngine), Abstract Base Class digunakan agar fungsi parent tidak bisa diinstansiasi sehingga objek harus merupakan salah satu dari derived Abstract Classnya, dan Static Class digunakan agar semua kelas bisa menggunakan alat-alat pembantu tanpa harus menginstansiasi objek dari class Util. Berikut adalah cuplikan program programnya.

1. Composition

```
// Letak penggunaan Composition (Dikutip dari GameEngine.hpp)
#ifndef _GAME_ENGINE_HPP_
#define _GAME_ENGINE_HPP_

// INCLUDE MODULES

class GameEngine
{
private:
    map<string, Player *> players;
    vector<string> playerNames;
    GameContext gameConfig;
    Shop shop;
    int currentTurn;
    static const vector<string> commands;
    // Pada atribut, GameEngine menggunakan kelas GameContext, Shop, dan Player yang merupakan kelas dari
       file lain. Ini menunjukkan bahwa ada asosiasi yang berupa composition antara GameEngine dengan
       ketiga objek yang disebutkan

    .
    .
    .
```

```
public:
    // METHODS
};

#endif
```

2. Abstract Base Class

```
// Letak penggunaan virtual untuk mendefinisikan Abstract Base Class
#ifndef _ANIMAL_HPP_
#define _ANIMAL_HPP_

#include "GameObject.hpp"
#include "Product.hpp"
#include <vector>
#include "../../GameException/header/GameException.hpp"

/** Abstract Class for Animal */
class Animal : public GameObject
{
protected:
    // ATRIBUT

public:

    .
    .
    .
    .

    virtual void feed(const Product &food) = 0;
    virtual void operator+=(const Product &food) = 0;
    // Ini adalah fungsi pure virtual yang membuat kelas ini menjadi kelas abstrak

    .
    .
```

```
    .
    .

};

// DEFINISI DARI KELAS INHERITANCE ANIMAL

#endif
```

3. Static Class

```
// Util.hpp
#ifndef _UTIL_HPP_
#define _UTIL_HPP_

#include <string>
#include <vector>
#include <iostream>
#include <chrono>
#include <thread>
#include <algorithm>
#include "../../GameObject/header/Plant.hpp"
#include "../../GameObject/header/Animal.hpp"
#include "../../GameObject/header/Building.hpp"
#include "../../GameObject/header/Product.hpp"
using namespace std;

class Util
{
public:
    /**
     * Meng-convert string koordinat menjadi pair<row,col>
     * Indexing string dimulai dari 1
     * @param koordinat
     * @return Item pada cell tersebut : T
     */
```

```
    static pair<int, int> strToRowCol(string koordinat);

    /**
     * @param in : input koordinat, separated by comma
     * @return vector of koordinat
     */
    static vector<string> parserSlots(string in);

    /**
     * @return persen pajak berdasarkan kekayaan kena pajak
     */
    static int persenPajak(int kkp);

    /**
     * @param in int masukan user
     * @return hasil convert dari angka huruf : 1 -> a
     * handle hingga 26 * 26 + 26
     */
    static string angkaToHuruf(int in);

    /**
     * obj is not Null
     * @param obj
     * @return item as one of the child of gameObject
     */
    static GameObject *callCCtor(GameObject *obj);

    /**
     * @return true jika urutan int naik. Jika sama, return true jika urutan pair string turun
     */
    static bool customComparator(const pair<int, pair<string, string>> &a, const pair<int, pair<string,
string>> &b);

    static void clearScreen();

    static void waitScreen(int ms);

    static void displayStartingScreen();
```

```
    static bool containSpace(string s);

    /**
     * Meng-convert row col menjadi string coordinate
     * Indexing string dimulai dari 1
     * @param row: int
     * @param col: int
     * @return coordinate: string
     */
    static string rowColToStr(int row, int col);
};

#endif
```

```
// Util.cpp
#include "../header/Util.hpp"

pair<int, int> Util::strToRowCol(string koordinat)
{
    // Cari nilai kolom
    int i = 0;
    int col = 0;
    if (koordinat[i] < 'A' || koordinat[i] > 'Z')
    {
        throw InputInvalidException();
    }
    else
    {
        while (koordinat[i] >= 'A' && koordinat[i] <= 'Z')
        {
            col *= 26;
            col += koordinat[i] - 'A' + 1;
            i++;
        }
    }

    // Cari nilai baris
```

```
    int row = 0;
    for (int j = i; j < (int)koordinat.length(); j++)
    {
        // Cek
        if (koordinat[j] >= '0' && koordinat[j] <= '9')
        {
            row *= 10;
            row += koordinat[j] - '0';
        }
        else
        {
            throw InputInvalidException();
        }
    }
    row--;
    col--;
    return make_pair(row, col);
}

vector<string> Util::parserSlots(string in)
{
    vector<string> hasil;
    string temp = "";
    for (int i = 0; i < (int)in.length(); i++)
    {
        //
        if (in.at(i) == ',')
        {
            hasil.push_back(temp);
            temp = "";
        }
        else if (in.at(i) != ' ')
        {
            temp.push_back(in.at(i));
        }
    }
    hasil.push_back(temp);
    return hasil;
```

```
}
int Util::persenPajak(int kkp)
{
    if (kkp <= 6)
    {
        return 5;
    }
    else if (kkp <= 25)
    {
        return 15;
    }
    else if (kkp <= 50)
    {
        return 25;
    }
    else if (kkp <= 500)
    {
        return 30;
    }
    else
    {
        return 35;
    }
}

string Util::angkaToHuruf(int in)
{
    string result = "";
    while (in != -1)
    {
        int sisa = in % 26;
        result += (char)('A' + sisa);
        in /= 26;
        in--;
    }
    reverse(result.begin(), result.end());
    return result;
}
```

```
GameObject *Util::callCCtor(GameObject *obj)
{
    if (dynamic_cast<Plant *>(obj) != NULL)
    {
        Plant *item = new Plant(*dynamic_cast<Plant *>(obj));
        return item;
    }
    else if (dynamic_cast<Animal *>(obj) != NULL)
    {
        if (dynamic_cast<Carnivore *>(obj) != NULL)
        {
            Carnivore *item = new Carnivore(*dynamic_cast<Carnivore *>(obj));
            return item;
        }
        else if (dynamic_cast<Herbivore *>(obj) != NULL)
        {
            Herbivore *item = new Herbivore(*dynamic_cast<Herbivore *>(obj));
            return item;
        }
        else if (dynamic_cast<Omnivore *>(obj) != NULL)
        {
            Omnivore *item = new Omnivore(*dynamic_cast<Omnivore *>(obj));
            return item;
        }
    }
    else if (dynamic_cast<Product *>(obj) != NULL)
    {
        Product *item = new Product(*dynamic_cast<Product *>(obj));
        return item;
    }
    else // if (dynamic_cast<Building *>(obj) != NULL)
    {
        Building *item = new Building(*dynamic_cast<Building *>(obj));
        return item;
    }

    return NULL;
```

```
}

bool Util::customComparator(const pair<int, pair<string, string>> &a, const pair<int, pair<string, string>> &b)
{
    // Priority int dengan urutan menurun
    if (a.first != b.first)
    {
        return a.first > b.first; // urutan menurun
    }

    // Jika int sama, bandingkan string secara naik
    return a.second.first < b.second.first;
}

void Util::clearScreen()
{
    system("clear");
}

void Util::waitScreen(int ms)
{
    this_thread::sleep_for(chrono::milliseconds(ms));
}

void Util::displayStartingScreen()
{
    clearScreen();
    cout << "Initialization begin..." << endl;
    cout << "▒▒▒▒▒▒▒▒▒▒ 10%" << endl;
    waitScreen(300);
    clearScreen();

    cout << "Initialization begin..." << endl;
    cout << "▒▒▒▒▒▒▒▒▒▒ 20%" << endl;
    waitScreen(300);
    clearScreen();

    cout << "Initialization begin..." << endl;
```

```
    cout << "███▒▒▒▒▒▒▒ 30%" << endl;
    waitScreen(300);
    clearScreen();

    cout << "Initialization begin..." << endl;
    cout << "████▒▒▒▒▒▒ 40%" << endl;
    waitScreen(300);
    clearScreen();

    cout << "Initialization begin..." << endl;
    cout << "█████▒▒▒▒▒ 50%" << endl;
    waitScreen(300);
    clearScreen();

    cout << "Initialization begin..." << endl;
    cout << "██████▒▒▒▒ 60%" << endl;
    waitScreen(300);
    clearScreen();

    cout << "Initialization begin..." << endl;
    cout << "███████▒▒▒ 70%" << endl;
    waitScreen(300);
    clearScreen();

    cout << "Initialization begin..." << endl;
    cout << "████████▒▒ 80%" << endl;
    waitScreen(300);
    clearScreen();

    cout << "Initialization begin..." << endl;
    cout << "█████████▒ 90%" << endl;
    waitScreen(300);
    clearScreen();

    cout << "Initialization begin..." << endl;
    cout << "██████████ 100%" << endl;
    waitScreen(300);
    clearScreen();
```

```
    cout <<
"=============================================================================================================";
    cout << '\n';
    std::cout << "  _    _       _                    ______                                  __     ___   ___
__     \n"
                " | |  | |     | |                |  ____|                                 \\ \\   / / \\ | \\
\\        / /\\    \n"
                " | |__| | __ _| | __ _           | |__   _ __ ___  _   _ _   _            \\ \\_/ /|  \\| |\\
\\  /\\  / /  \\   \n"
                " |  __  |/ _` | |/ _` |          |  __| | '_ ` _ \\| | | | | | |             \\   / | . ` | \\
\\/  \\/ / /\\ \\  \n"
                " | |  | | (_| | | (_| |          | |____| | | | | | |_| | |_| |              | |  | |\\  |  \\
/\\  / ____ \\ \n"
                " |_|  |_|\\__,_|_|\\__,_|          |______|_| |_| |_|\\__, |\\__,_|              |_|  |_| \\_|
\\/  \\/_/    \\_\\\n"
                "                                 ______                __/ |        ______
\n"
                "                               |______|              |___/        |______|
\n";
    cout <<
"=============================================================================================================";
    cout << "\n\n";
}

bool Util::containSpace(string s)
{
    for (int i = 0; i < (int)s.size(); i++)
    {
        if (s[i] == ' ')
            return true;
    }
    return false;
}

string Util::rowColToStr(int row, int col)
{
    return angkaToHuruf(col) + to_string(row + 1);
```

```
}
```

3. Bonus Yang dikerjakan

Pada tugas ini, kami membuat bonus kreasi mandiri berupa sebuah perintah baru, yaitu INFO. Perintah ini akan menampilkan data pemain saat ini, seperti penyimpanan (*inventory*), jumlah gulden, jumlah gulden tersisa untuk menang, berat pemain, berat pemain tersisa untuk menang, ladang (jika pemain saat ini memiliki *role* petani), dan peternakan (jika pemain saat ini memiliki *role* peternak).

3.1. Bonus yang diusulkan oleh spek

3.1.1. GUI

Tidak dikerjakan

3.1.2. Pola Desain

Tidak dikerjakan

3.2. Bonus Kreasi Mandiri

3.2.1. Info

Implementasi

```
// void Info (Dikutip dari GameEngine.cpp)
void GameEngine::info()
{
    // cetak
```

```
    players[playerNames[currentTurn]]->printInventory();
    cout << "Gulden: " << players[playerNames[currentTurn]]->getGulden() << endl;
    cout << "Banyak gulden untuk menang: " << max(gameConfig.getGuldenWin() -
players[playerNames[currentTurn]]->getGulden(), 0) << endl;
    cout << "Weight: " << players[playerNames[currentTurn]]->getWeight() << endl;
    cout << "Berat untuk menang: " << max(gameConfig.getWeightWin() -
players[playerNames[currentTurn]]->getWeight(), 0) << endl;

    if (players[playerNames[currentTurn]]->getType() == "PETANI")
    {
        cout << endl;
        dynamic_cast <Petani *>(players[playerNames[currentTurn]])->printLahan();
    }
    if (players[playerNames[currentTurn]]->getType() == "PETERNAK")
    {
        cout << endl;
        dynamic_cast <Peternak *>(players[playerNames[currentTurn]])->printPeternakan();
    }
}
```

4. Pembagian Tugas

Modul (dalam poin spek)	Implementer	Tester
Next	10023637 13522054 13522068 13522098 13522110 13522118	13522054 13522068 13522118
Cetak Penyimpanan	10023637 13522054 13522068	13522054 13522068 13522118

	13522098 13522110 13522118	
Pungut Pajak	10023637 13522054 13522068 13522098 13522110 13522118	13522054 13522068 13522118
Cetak Ladang dan Cetak Peternakan	10023637 13522054 13522068 13522098 13522110 13522118	13522054 13522068 13522118
Tanam	10023637 13522054 13522068 13522098 13522110 13522118	13522054 13522068 13522118
Ternak	10023637 13522054 13522068 13522098 13522110 13522118	13522054 13522068 13522118
Bangun Bangunan	10023637 13522054 13522068	13522054 13522118

	13522098 13522110 13522118	
Makan	10023637 13522054 13522068 13522098 13522110 13522118	13522054 13522068 13522118
Memberi Pangan	10023637 13522054 13522068 13522098 13522110 13522118	13522054 13522068 13522098 13522118
Membeli	10023637 13522054 13522068 13522098 13522110 13522118	13522054 13522068 13522118
Menjual	10023637 13522054 13522068 13522098 13522110 13522118	13522054 13522068 13522118
Memanen	10023637 13522054 13522068	13522054 13522068 13522098

	13522098 13522110 13522118	
Muat	13522068 13522098 13522110	13522054 13522068 13522098
Simpan	13522068 13522098 13522110	13522054 13522068 13522098 13522118
Tambah Pemain	10023637 13522054 13522068 13522098 13522110 13522118	13522054 13522068 13522118