

Kode Kelompok : N3O

Nama Kelompok : Walaoeh

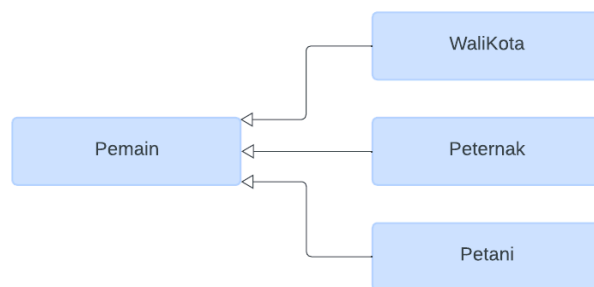
1. 10023608 / Tazkirah Amaliah
2. 13522042 / Amalia Putri
3. 13522048 / Angelica Kierra Ninta Gurning
4. 13522058 / Imanuel Sebastian Girsang
5. 13522080 / Julian Chandra Sutadi
6. 13522108 / Muhammad Neo Cicero Koda

Asisten Pembimbing : M. Syahrul Surya Putra

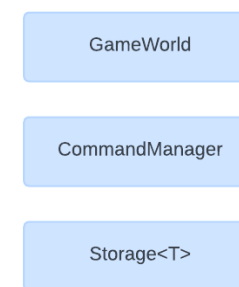
1. Diagram Kelas

Dalam merancang pemrograman berorientasi objek diperlukan suatu struktur program yang jelas, yakni macam-macam kelas serta hubungan antar kelas tersebut yang tergambar pada diagram kelas sebagai berikut.

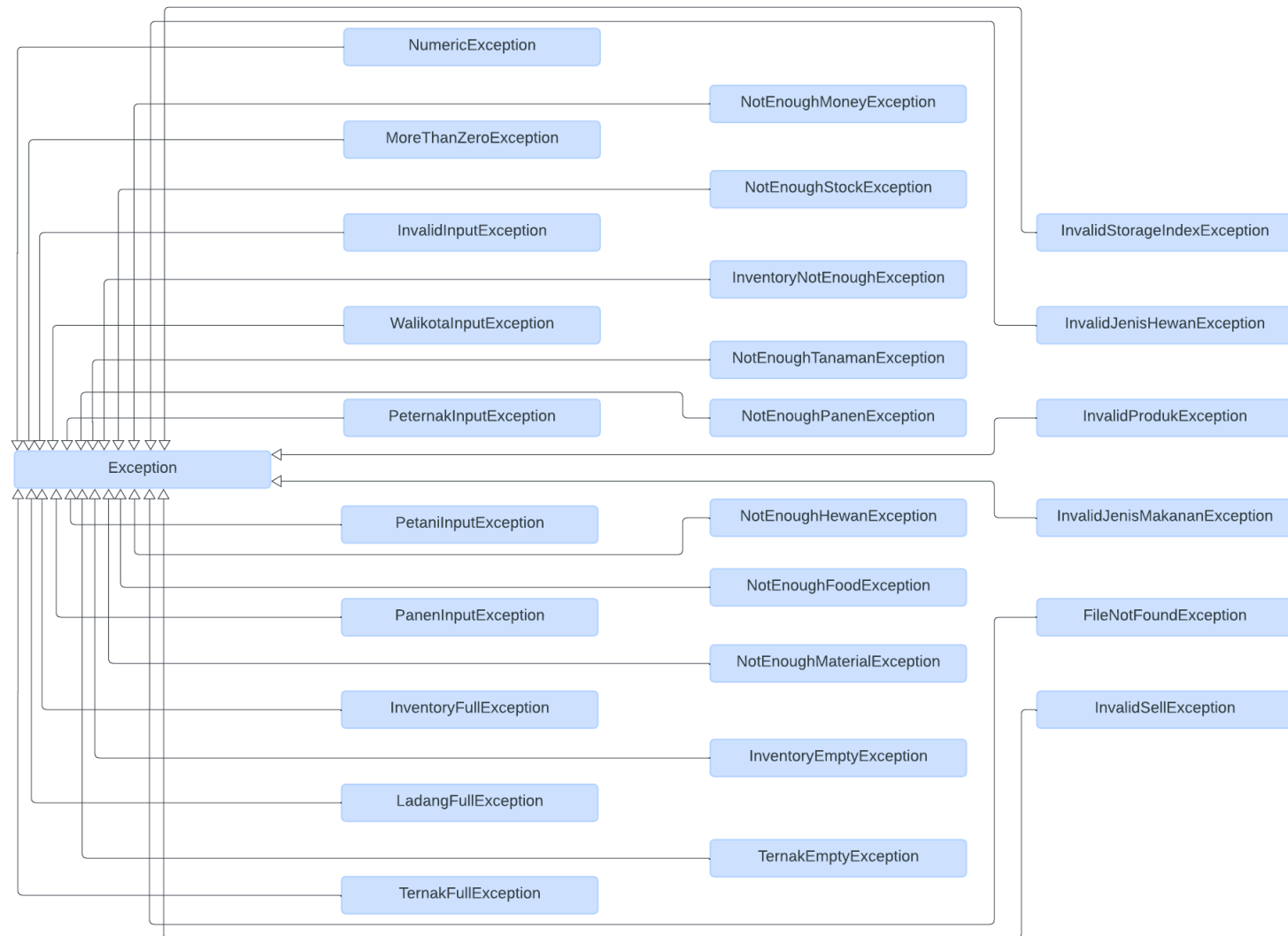
1.1. Hierarki Kelas



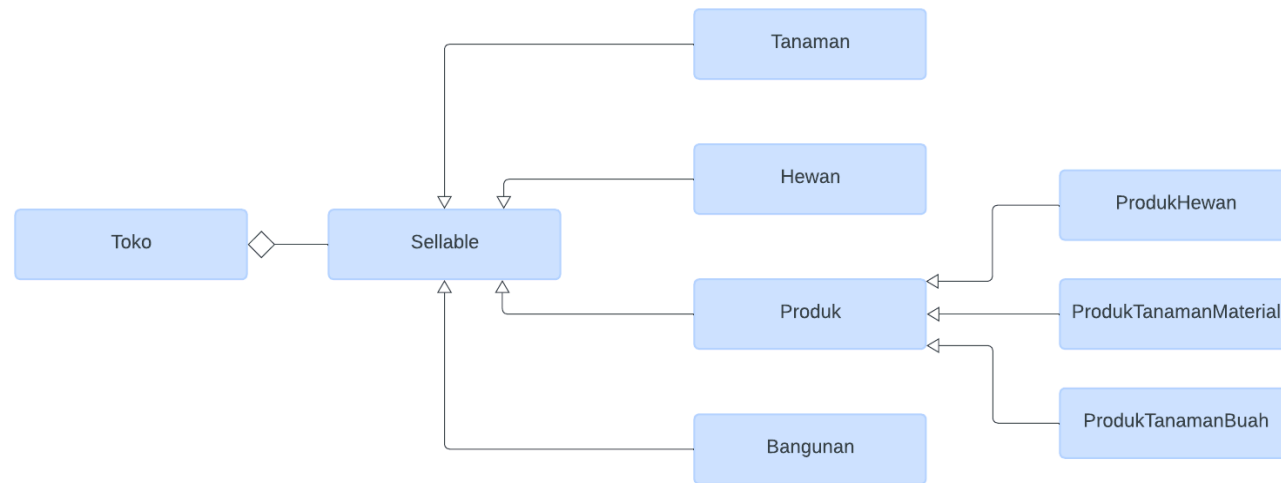
Gambar 1.1.1. Diagram Kelas: Pemain



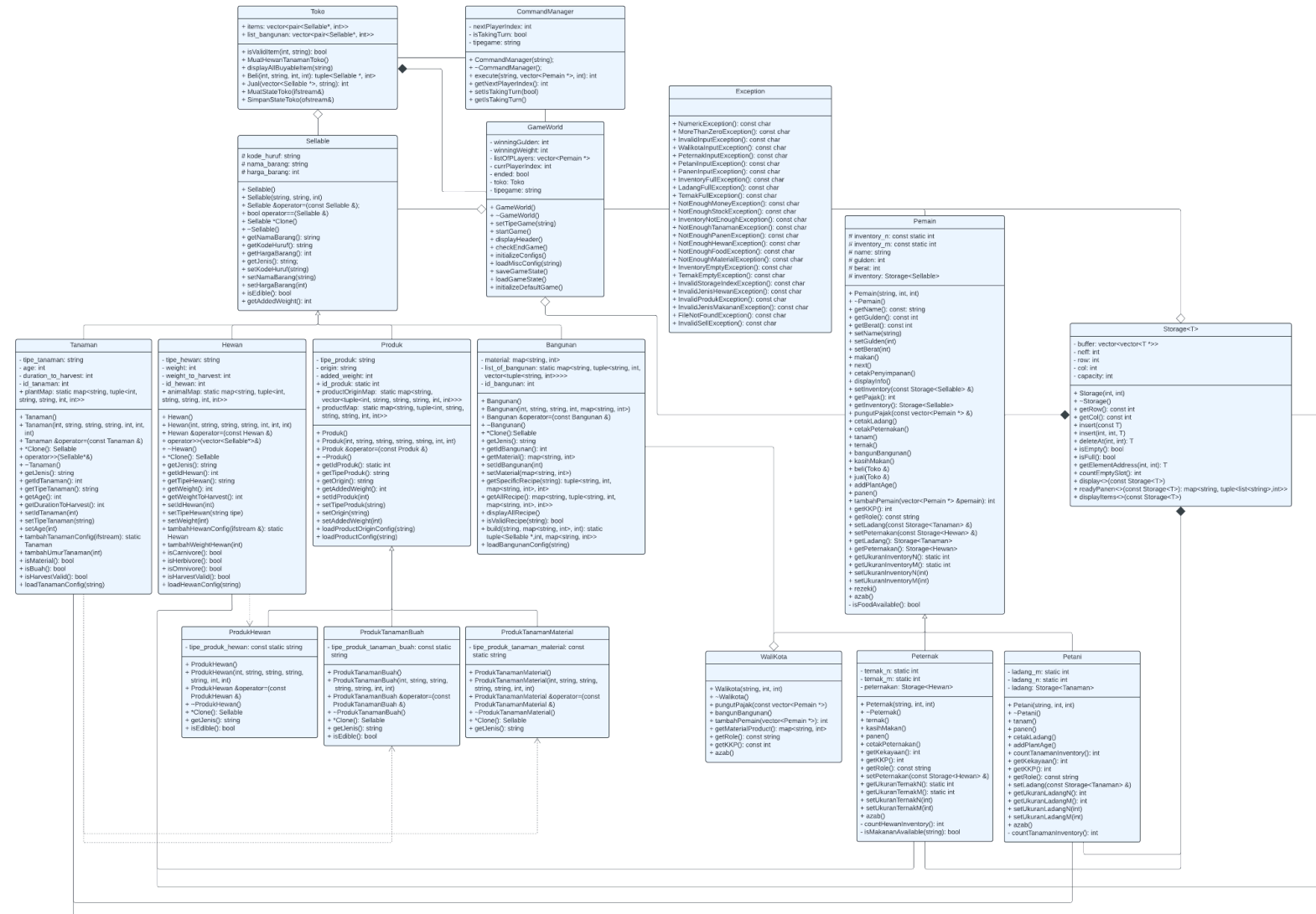
Gambar 1.1.2. Diagram Kelas: Main Manager



Gambar 1.1.3. Diagram Kelas: Exception

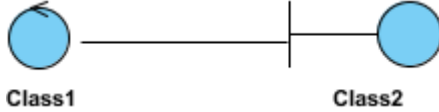
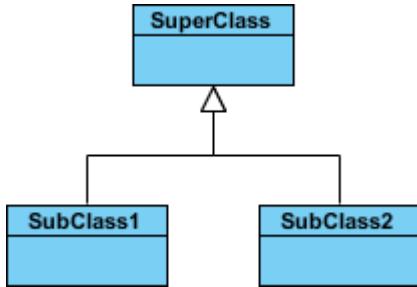
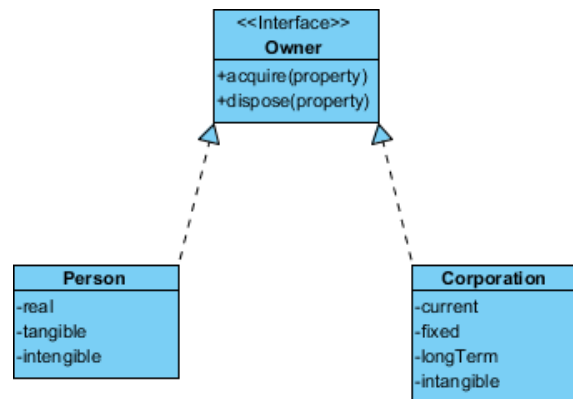


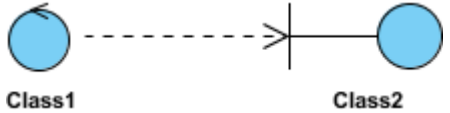
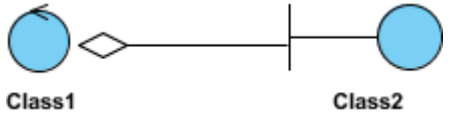
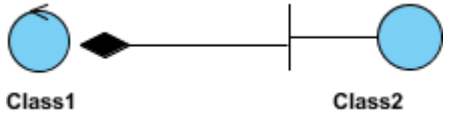
Gambar 1.1.4. Diagram Kelas: Toko dan Sellable



Gambar 1.1.5. Diagram Kelas

1.2. Legenda Diagram Kelas

Nama	Gambar	Keterangan
<i>Association</i>	 <p>Diagram showing an association between Class1 and Class2. A horizontal line connects two circles, with Class1 on the left and Class2 on the right.</p>	<ul style="list-style-type: none"> Sebuah tautan struktural antara dua kelas. Ada asosiasi antara Class1 dan Class2.
<i>Inheritance</i>	 <p>Diagram showing inheritance. A box labeled SuperClass is at the top. Below it are two boxes labeled SubClass1 and SubClass2. Arrows point from SubClass1 and SubClass2 to SuperClass.</p>	<ul style="list-style-type: none"> Mewakili hubungan "is a". Nama kelas abstrak ditampilkan dalam huruf miring. SubClass1 dan SubClass2 adalah spesialisasi dari SuperKelas.
<i>Realization</i>	 <p>Diagram showing realization. A box labeled <<Interface>> Owner is at the top. It contains two methods: +acquire(property) and +dispose(property). Below it are two boxes labeled Person and Corporation. Dashed arrows point from Person and Corporation to the Owner interface.</p>	<ul style="list-style-type: none"> Hubungan antara kelas blueprint dan objek yang berisi detail implementasi tingkatannya masing-masing. Objek ini dikatakan merealisasikan kelas blueprint tersebut atau dengan kata lain, ini menunjukkan hubungan antara antarmuka dan kelas pelaksana.

<i>Dependency</i>	 <p>Class1</p> <p>Class2</p>	<ul style="list-style-type: none"> • Sebuah jenis asosiasi khusus. • Ada antara dua kelas jika perubahan pada definisi salah satu kelas dapat menyebabkan perubahan pada kelas lainnya (tapi tidak sebaliknya). • Class1 bergantung pada Class2.
<i>Aggregation</i>	 <p>Class1</p> <p>Class2</p>	<ul style="list-style-type: none"> • Ini mewakili hubungan "<i>part of</i>". • Class2 adalah bagian dari Class1. • Banyak contoh (ditandai dengan *) dari Class2 dapat dihubungkan dengan Class1. • Objek dari Class1 dan Class2 memiliki masa hidup yang terpisah.
<i>Composition</i>	 <p>Class1</p> <p>Class2</p>	<ul style="list-style-type: none"> • Sebuah jenis agregasi khusus di mana bagian-bagian hancur ketika keseluruhan dihancurkan. • Objek dari Class2 hidup dan mati bersama dengan Class1. • Class2 tidak dapat berdiri sendiri.

Tabel 1.2.1. Legenda Diagram Kelas

(sumber: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/>)

1.3. Desain Kelas

1.3.1. GameWorld

Kelas GameWorld menampung banyak kelas dan mengatur jalannya permainan antara pemain, yakni walikota, peternak, dan petani dengan program serta aturan yang berlaku pada permainan.

1.3.2. **CommandManager**

Kelas CommandManager melakukan eksekusi program dari banyak *method* sesuai program yang ada, seperti “NEXT”, “CETAK_PENYIMPANAN”, “PUNGUT_PAJAK”, dan lain-lain. Selain itu, suatu masukan yang tidak valid akan ditangani oleh kelas Exception.

1.3.3. **Exception**

Kelas Exception secara implisit menerapkan konsep *inheritance* dan *polymorphism* karena setiap *subclass*, seperti InvalidInputException, WalikotalInputException, InventoryFullException, dan lainnya pada Exception merujuk pada suatu *superclass*, yakni suatu modul bernama exception.

1.3.4. **Pemain**

Kelas Pemain menerapkan konsep *inheritance* dan *polymorphism* karena setiap *subclass*, yakni WaliKota, Peternak, dan Petani merujuk pada *superclass*, yakni Pemain. Selain itu, kelas Pemain menerapkan konsep *abstract class*, yang mana memiliki banyak *method* virtual. Kelas ini juga memiliki ketergantungan dengan kelas lain di luar hierarki kelas Pemain, yakni dengan kelas Toko yang merupakan *superclass* dari beberapa kelas yang dibutuhkan oleh Pemain dalam program.

1.3.5. **Storage**

Kelas Storage menerapkan konsep *Generic class* dan STL (*Standard Template Library*) sehingga memudahkan suatu objek yang memiliki tipe yang berbeda-beda dapat diaplikasikan pada suatu program yang memiliki cara kerja yang sama. Konsep ini meningkatkan efisiensi program dan mengurangi hal yang repetitif.

1.3.6. **Toko**

Kelas Toko menerapkan konsep STL (*Standard Template Library*) yang *attribute*-nya menggunakan objek Sellable serta menyimpan nilai kuantitas tiap objek Sellable tersebut.

1.3.7. Sellable

Kelas Sellable menerapkan konsep *inheritance* dan *polymorphism* yang memiliki 4 *subclass*, yakni kelas Tanaman, Hewan, Produk, dan Bangunan. Pada kelas Produk juga terdapat 3 *subclass*, yakni ProdukHewan, ProdukTanamanBuah, dan ProdukTanamanMaterial. Semua komponen pada Sellable tersebut diimplementasikan untuk melakukan transaksi antara kelas Toko dan Pemain.

2. Penerapan Konsep OOP

2.1. Inheritance & Polymorphism

Konsep *inheritance* yang digunakan pada program Kelola Kerajaan diimplementasikan pada kelas Pemain dan Sellable sebagai *superclass* dan turunannya. Pada kelas Pemain (*superclass*) memiliki *subclass*: WaliKota, Peternak, dan Petani. Pada kelas Sellable (*superclass*) memiliki *subclass*: Tanaman, Hewan, Bangunan, dan Produk yang mana sekaligus menjadi *superclass* bagi ProdukHewan, ProdukTanamanBuah, dan ProdukTanamanMaterial sebagai *subclass*. Konsep *polymorphism* diimplementasikan pada semua *subclass* (*child*) tersebut agar dapat dikelola seperti *superclass* (*parent*) sehingga *superclass* dan *subclass* memiliki sifat yang sama. Tabel 2.1.1. Inheritance & Polymorphism

Kelas	Kode Program
-------	--------------

Superclass (parent): Pemain

Subclass (child):

1. WaliKota,
2. Peternak, dan
3. Petani

```

class Pemain
{
private:
    bool isFoodAvailable();

protected:
    static int inventory_n;
    static int inventory_m;
    string name;
    int gulden;
    int berat;
    Storage<Sellable> inventory;

public:
    Pemain(string name, int gulden, int berat);
    ~Pemain();

    /**
     * @brief Getter nama pemain
     *
     * @return string
     */
    string getName() const;

    /**
     * @brief Getter gulden pemain
     *
     * @return int
     */
    int getGulden() const;

    /**
     * @brief Getter berat pemain
     *
     * @return int
     */
    int getBerat() const;

    /**
     * @brief Setter untuk nama pemain
     *
     */
}

```

```
> libs > Pemain > Pemain.hpp > Pemain
class Pemain
{
    void setName(string name);
    /**
     * @brief Setter untuk Gulden pemain
     */
    void setGulden(int gulden);
    /**
     * @brief Setter untuk berat pemain
     */
    void setBerat(int berat);
    /**
     * @brief Makan untuk menambah berat pemain
     */
    void makan();
    /**
     * @brief Menyelesaikan turn pemain saat ini, memberikan turn kepada pemain selanjutnya
     */
    void next();
    /**
     * @brief Mencetak Inventory pemain saat ini
     */
    void cetakPenyimpanan();
    /**
     * @brief Mencetak Informasi penting dari pemain ke layar
     */
    void displayInfo();
    /**
     * @brief to copy inventory into this player inventory
     */
    void setInventory(const Storage<Sellable> &storage);
    /**
     * @brief Menghitung Kekayaan pemain
     */
}
```

```

class Pemain
{
    /**
     * @brief Menghitung Kekayaan pemain
     */
    virtual int getKekayaan();

    /**
     * @brief Mengembalikan pajak yang dibebankan kepada pemain
     *
     * @return int
     */
    int getPajak();

    /**
     * @brief Mengembalikan penyimpanan yang dimiliki pemain
     *
     * @return Storage<Sellable>
     */
    Storage<Sellable> getInventory();

    /**
     * @brief Pungut Pajak, Khusus untuk walikota
     *
     * @param pemain vector pemain yang akan dikenakan pajak
     */
    virtual void pungutPajak(const vector<Pemain *> &pemain);
    virtual void cetakLadang();
    virtual void cetakPernakan();
    virtual void tanam();
    virtual void ternak();

    /**
     * @brief Membangun Bangunan, Khusus untuk walikota
     *
     */
    virtual void bangunBangunan();
    virtual void kasihMakan();
    void beli(Toko & toko);
    void jual(Toko & toko);
    virtual void addPlantAge();
    virtual void panen();

    /**

```

```

122     virtual void panen();
123     /**
124      * @brief Menambah pemain atau peternak, Khusus untuk walikota
125      *
126      * @param pemain vector pemain yang akan dimanipulasi
127      * @return index pemain baru (untuk menentukan urutan main nantinya)
128      */
129     virtual int tambahPemain(vector<Pemain *> &pemain);
130     virtual int getKKP() = 0;
131     virtual string getRole() const = 0;
132
133     // Configurations needs
134
135     virtual void setLadang(const Storage<Tanaman> &storage);
136     virtual void setPeternakan(const Storage<Hewan> &storage);
137     virtual Storage<Tanaman> getLadang();
138     virtual Storage<Hewan> getPeternakan();
139
140     static int getUkuranInventoryN();
141     static int getUkuranInventoryM();
142     static void setUkuranInventoryN(int n);
143     static void setUkuranInventoryM(int m);
144
145     void rezeki();
146     virtual void azab() = 0;
147 };
148
149 #endif

```

```
class Petani : public Pemain
{
private:
    static int ladang_m;
    static int ladang_n;
    Storage<Tanaman> ladang;
    int countTanamanInventory();

public:
    Petani(string name, int gulden, int berat);
    ~Petani();
    void tanam();
    void panen();
    void cetakLadang();
    void addPlantAge();
    int getKekayaan();
    int getKKP();
    string getRole() const;

    void setLadang(const Storage<Tanaman> &storage);
    Storage<Tanaman> getLadang();

    static int getUkuranLadangN();
    static int getUkuranLadangM();
    static void setUkuranLadangN(int n);
    static void setUkuranLadangM(int m);

    void azab();
};

#endif
```

```
class Peternak : public Pemain
{
private:
    static int ternak_n;
    static int ternak_m;
    Storage<Hewan> peternakan;
    int countHewanInventory();
    bool isMakananAvailable(string tipeHewan);

public:
    Peternak(string name, int gulden, int berat);
    ~Peternak();
    void ternak();
    void kasihMakan();
    void panen();
    void cetakPeternakan();
    int getKekayaan();
    int getKKP();
    string getRole() const;

    void setPeternakan(const Storage<Hewan> &storage);
    Storage<Hewan> getPeternakan();
    static int getUkuranTernakN();
    static int getUkuranTernakM();
    static void setUkuranTernakN(int n);
    static void setUkuranTernakM(int m);

    void azab();
};

#endif
```

```

os > Pemain > Walikota > Walikota.hpp > ...
class Walikota : public Pemain
{
public:
    /**
     * @brief User Defined Constructor
     *
     * @param name
     * @param gulden
     * @param berat
     */
    Walikota(string name, int gulden, int berat);
    /**
     * @brief Destroy the Walikota object
     *
     */
    ~Walikota();
    /**
     * @brief tagihPajak, helper function to get the tax for a certain player
     *
     * @param pemain
     * @return int
     */
    void pungutPajak(const vector<Pemain *> &pemain) override;
    /**
     * @brief bangunBangunan, method to build a building
     *
     */
    void bangunBangunan() override;
    /**
     * @brief tambahPemain, method to add a player
     *
     * @param pemain
     * @return int, the new player index (needed to adjust whos gonna be playing next)
     */
    int tambahPemain(vector<Pemain *> &pemain) override;
    /**
     * @brief Get the material and its quantity from walikota inventory
     *
     * @return map<string, int>
     */

```

```
14 class Walikota : public Pemain
37     /**
38      * @brief bangunBangunan, method to build a building
39      *
40      */
41     void bangunBangunan() override;
42     /**
43      * @brief tambahPemain, method to add a player
44      *
45      * @param pemain
46      * @return int, the new player index (needed to adjust whos gonna be playing next)
47      */
48     int tambahPemain(vector<Pemain *> &pemain) override;
49     /**
50      * @brief Get the material and its quantity from walikota inventory
51      *
52      * @return map<string, int>
53      */
54     map<string, int> getMaterialProduct();
55     /**
56      * @brief Get the Role of this walikota (walikota)
57      *
58      * @return string
59      */
60
61     string getRole() const;
62
63     int getKKP() ;
64
65     void azab();
66 };
67
68 #endif
```


Contoh *polymorphism*: *Instance* dari sebuah petani, peternak, dan walikota dibuat, namun disimpan dalam `listOfPlayers` dengan dianggap sebagai pemain.

```
void GameWorld::initializeDefaultGame()
{
    Pemain *pemain1 = new Petani("Petani1", 50, 40);
    Pemain *pemain2 = new Peternak("Peternak1", 50, 40);
    Pemain *pemain3 = new Walikota("Walikota", 50, 40);

    listOfPlayers.push_back(pemain1);
    listOfPlayers.push_back(pemain2);
    listOfPlayers.push_back(pemain3);
}
```

Superclass (parent): Sellable

Subclass (child):

1. Tanaman,
2. Hewan,
3. Produk (*parent* dari:
4. ProdukHewan,
5. ProdukTanamanBuah,
6. ProdukTanamanMaterial), dan
7. Bangunan

```
class Sellable
{
protected:
    string kode_huruf; // 3 huruf
    string nama_barang;
    int harga_barang;

public:
    // ctor default
    Sellable();

    // ctor user defined
    Sellable(string kode, string nama, int harga);

    // operator overloading
    Sellable &operator=(const Sellable &other);
    bool operator==(Sellable &other);

    virtual Sellable *Clone() = 0;

    // dtor
    virtual ~Sellable();

    /* Methods */
    // getter
    string getNamaBarang();
    string getKodeHuruf();
    int getHargaBarang();
    virtual string getJenis() = 0;

    // setter
    void setKodeHuruf(string kode);
    void setNamaBarang(string nama);
    void setHargaBarang(int harga);

    // additional
    virtual bool isEdible();
    virtual int getAddedWeight();
};

#endif
```

```
/*<ID> <KODE_HURUF> <NAME> <TYPE> <DURATION_TO_HARVEST> <PRICE>*/  
class Tanaman : public Sellable  
{  
private:  
    string tipe_tanaman;  
    int id_tanaman = 1;  
    int age;  
    int duration_to_harvest;  
public:  
    static map<string, tuple<int, string, string, int, int>> plantMap;  
  
    // ctor default  
    Tanaman();  
  
    // ctor user defined  
    Tanaman(int id, string kode, string nama_tanaman, string tipe, int umur, int durasi_panen, int harga);  
  
    // operator overloading  
    Tanaman &operator=(const Tanaman &other);  
  
    Sellable *Clone();  
  
    // dtor  
    ~Tanaman();  
  
    /* Methods */  
    // getter  
    string getJenis();  
  
    int getIdTanaman();  
    string getTipeTanaman();  
    int getAge();  
    int getDurationToHarvest();  
  
    // setter  
    void setIdTanaman(int id);  
    void setTipeTanaman(string tipe);  
    void setAge(int age);
```

```
static Tanaman tambahTanamanConfig(ifstream &file);

void tambahUmurTanaman(int age);

// Konversi tanaman menjadi produk (buah) kalau weight sudah mencapai weight_to_harvest
void operator>>(Sellable*& produkTanamanBuahBaru);

// boolean
bool isMaterial();
bool isBuah();
bool isHarvestValid();

// load config to animalMap
static void loadTanamanConfig(string path);
};

#endif
```

```

3
4  /*<ID> <KODE_HURUF> <NAME> <TYPE> <WEIGHT_TO_HARVEST> <PRICE>*/
5  class Hewan : public Sellable
6  {
7  private:
8      int id_hewan = 1;
9      string tipe_hewan;
10     int weight;
11     int weight_to_harvest;
12
13 public:
14     // map with key =
15     static map<string, tuple<int, string, string, int, int>> animalMap;
16
17     // ctor default
18     Hewan();
19
20     // ctor user defined
21     Hewan(int id, string kode, string nama_hewan, string tipe, int weight, int weight_to_harvest, int harga);
22
23     // operator overloading
24     Hewan &operator=(const Hewan &other);
25
26     // dtor
27     ~Hewan();
28
29     Sellable *Clone();
30
31     /* Methods */
32     // getter
33     string getJenis();
34
35     int getIdHewan();
36     string getTipeHewan();
37     int getWeight();
38     int getWeightToHarvest();

```

```
49
50 // setter
51 void setIdHewan(int id);
52 void setTipeHewan(string tipe);
53 void setWeight(int weight);
54
55 static Hewan tambahHewan();
56 static Hewan tambahHewanConfig(istream &file);
57
58 void tambahWeightHewan(int weight);
59 // Konversi hewan menjadi produk (hewan) kalau weight sudah mencapai weight_to_harvest
60 void operator>>(vector<Sellable*>& vectorProdukHewan);
61
62 // boolean
63 bool isCarnivore();
64 bool isHerbivore();
65 bool isOmnivore();
66 bool isHarvestValid();
67
68 // load config to animalMap
69 static void loadHewanConfig(string path);
70 };
71
72 #endif
```

```

class Bangunan : public Sellable
{
private:
    int id_bangunan;
    map<string, int> material; // <string nama_material, int jumlah_material>
public:
    static map<string, tuple<string, int, map<string, int>, int>> list_of_bangunan;
    // map with key = nama_bangunan, value = tuple<kode_bangunan, harga_bangunan, vector<tuple<material, jumlah_material>>>

    // ctor default
    Bangunan();

    // ctor user defined
    Bangunan(int id, string kode, string nama_bangunan, int harga, map<string, int> material);

    // operator overloading
    Bangunan &operator=(const Bangunan &other);

    // dtor
    ~Bangunan();

    Sellable *Clone();

    /* Methods */
    // getter
    string getJenis();

    int getIdBangunan();

    map<string, int> getMaterial();

    // setter
    void setIdBangunan(int id);

    void setMaterial(map<string, int> material);
    tuple<string, int, map<string, int>, int> getSpecificRecipe(string name);

    map<string, tuple<string, int, map<string, int>, int>> getAllRecipe();

    static void displayAllRecipe();
    static bool isValidRecipe(string name);
    static tuple<Sellable *, int, map<string, int>> build(string name, map<string, int> material);
    // load config to list_of_bangunan
    static void loadBangunanConfig(string path);
};

#endif

```

```

/*<ID> <KODE_HURUF> <NAME> <TYPE> <ORIGIN> <ADDED_WEIGHT> <PRICE>*/
class Produk : public Sellable
{
protected:
    string tipe_produk;
    string origin;
    int added_weight;

public:
    static int id_produk;
    // key: nama barang origin; value: id, code, name, type, addedWeight, price
    static map<string, vector<tuple<int, string, string, string, int, int>>> productOriginMap;
    static map<string, tuple<int, string, string, string, int, int>> productMap;

    // ctor default
    Produk();

    // ctor user defined
    Produk(int id, string kode, string nama_produk, string tipe, string origin, int added_weight, int harga);

    // operator overloading
    Produk &operator=(const Produk &other);

    // dtor
    virtual ~Produk();

    /* Methods */
    // getter
    static int getIdProduk();
    string getTypeProduk();
    string getOrigin();
    int getAddedWeight();

    // setter
    void setIdProduk(int id);
    void setTypeProduk(string tipe);
    void setOrigin(string origin);
    void setAddedWeight(int added_weight);

    // load config to productMap
    static void loadProductOriginConfig(string path);
    static void loadProductConfig(string path);
};

#endif

```



```

/*<ID> <KODE_HURUF> <NAME> <TYPE> <ORIGIN> <ADDED_WEIGHT> <PRICE>*/
class ProdukHewan : public Produk
{
private:
    const static string tipe_produk_hewan;
public:
    // ctor default
    ProdukHewan();

    // ctor user defined
    ProdukHewan(int id, string kode, string nama_produk, string tipe, string origin, int added_weight, int harga);

    // operator overloading
    ProdukHewan &operator=(const ProdukHewan &other);

    // dtor
    ~ProdukHewan();

    // Clone
    Sellable *Clone();

    /* Methods */
    // getter
    string getJenis();

    /*
    1. Herbivora
    Cow -> 1 Cow Meat
    Sheep -> 1 Sheep Meat
    Horse -> 1 Horse Meat
    Rabbit -> 1 Rabbit Meat

    2. Karnivora
    Snake -> 1 Snake Meat

    3. Omnivora
    Chicken -> 1 Chicken Meat & 1 Chicken Egg
    Duck -> 1 Duck Meat & 1 Duck Egg
    */

    bool isEdible() override;
};

#endif

```

```

/*<ID> <KODE_HURUF> <NAME> <TYPE> <ORIGIN> <ADDED_WEIGHT> <PRICE>*/
class ProdukTanamanBuah : public Produk
{
private:
    const static string tipe_produk_tanaman_buah;
public:
    // ctor default
    ProdukTanamanBuah();

    // ctor user defined
    ProdukTanamanBuah(int id, string kode, string nama_produk, string tipe, string origin, int added_weight, int harga);

    // operator overloading
    ProdukTanamanBuah &operator=(const ProdukTanamanBuah &other);

    // dtor
    ~ProdukTanamanBuah();

    // Clone
    Sellable *Clone();

    /* Methods */
    // getter
    string getJenis();
    bool isEdible() override;
};

#endif

```

```

/*<ID> <KODE_HURUF> <NAME> <TYPE> <ORIGIN> <ADDED_WEIGHT> <PRICE>*/
class ProdukTanamanMaterial : public Produk
{
private:
    const static string tipe_produk_tanaman_material;
public:
    // ctor default
    ProdukTanamanMaterial();

    // ctor user defined
    ProdukTanamanMaterial(int id, string kode, string nama_produk, string tipe, string origin, int added_weight, int harga);

    // operator overloading
    ProdukTanamanMaterial &operator=(const ProdukTanamanMaterial &other);

    // dtor
    ~ProdukTanamanMaterial();

    // Clone
    Sellable *Clone();

    /* Methods */
    // getter
    string getJenis();
};

#endif

```

Contoh *Polymorphism*: Objek hasil instansiasi kelas bangunan dan produk diperlakukan sebagai *Sellable* untuk disimpan dalam daftar produk toko.

```
Sellable *item = nullptr;
Sellable *bangunan = nullptr;

if (Produk::productMap.find(itemName) != Produk::productMap.end())
{
    tuple<int, string, string, string, int, int> product_item_tuple

    if (get<3>(product_item_tuple) == "PRODUCT_MATERIAL_PLANT")
    {
        item = new ProdukTanamanMaterial(get<0>(product_item_tuple),
    }
    else if (get<3>(product_item_tuple) == "PRODUCT_FRUIT_PLANT")
    {
        item = new ProdukTanamanBuah(get<0>(product_item_tuple), get
    }
    else if (get<3>(product_item_tuple) == "PRODUCT_ANIMAL")
    {
        item = new ProdukHewan(get<0>(product_item_tuple), get<1>(pr
    }
}

if (Bangunan::list_of_bangunan.find(itemName) != Bangunan::list_of_b
{
    tuple<string, int, map<string, int>, int> bangunan_item_tuple =

    bangunan = new Bangunan(get<3>(bangunan_item_tuple), get<0>(bang
}

if (item != nullptr)
{
    items.push_back(make_pair(item, quantity));
}
```

Tabel 2.1.1. Method/Operator Overloading

2.2. Method/Operator Overloading

Konsep *Method/Operator Overloading* yang digunakan pada program Kelola Kerajaan diimplementasikan pada kelas

Kelas	Kode Program
Sellable (operator==)	<pre> 1 bool Sellable::operator==(Sellable &other) 2 { 3 return this->kode_huruf == other.getKodeHuruf() && this->nama_barang == other.getNamaBarang() && this->harga_barang == other.getHargaBarang(); 4 }</pre>
Storage (operator+)	<pre> /** * @brief Insert object to first empty slot * * @param obj Object to be inserted */ void operator+(T &obj) { bool found = false; int i = 0; while (i < row && !found) { int j = 0; while (j < col && !found) { if (buffer[i][j] == nullptr) { buffer[i][j] = &obj; found = true; } j++; } i++; } neff++; }</pre>

Tanaman (operator>>)	<pre> void Tanaman::operator>>(Sellable*0 produkTanamanBaru) { vector<tuple<int, string, string, string, int, int>> produk_buah_vektor = Produk::productOriginMap[getNamaBarang()]; if (isBuah()) { produkTanamanBaru = new ProdukTanamanBuah(get<0>(produk_buah_vektor[0]), get<1>(produk_buah_vektor[0]), get<2>(produk_buah_vektor[0]), get<3>(produk_buah_vektor[0]), getNamaBarang(), get<4>(produk_buah_vektor[0]), get<5>(produk_buah_vektor[0])); } else { produkTanamanBaru = new ProdukTanamanMaterial(get<0>(produk_buah_vektor[0]), get<1>(produk_buah_vektor[0]), get<2>(produk_buah_vektor[0]), get<3>(produk_buah_vektor[0]), getNamaBarang(), get<4>(produk_buah_vektor[0]), get<5>(produk_buah_vektor[0])); } return; } </pre>
Hewan (operator>>)	<pre> void Hewan::operator>>(vector<Sellable*>& vektorProdukHewan) { vector<tuple<int, string, string, string, int, int>> produkBaruVektor = Produk::productOriginMap[getNamaBarang()]; for (int i = 0; i < produkBaruVektor.size(); i++) { Sellable* produkHewanBaru = new ProdukHewan(get<0>(produkBaruVektor[i]), get<1>(produkBaruVektor[i]), get<2>(produkBaruVektor[i]), get<3>(produkBaruVektor[i]), getNamaBarang(), get<4>(produkBaruVektor[i]), get<5>(produkBaruVektor[i])); vektorProdukHewan.push_back(produkHewanBaru); } return; } </pre>

Tabel 2.2.1. Method/Operator Overloading

2.3. Template & Generic Classes

Konsep *Template & Generic Classes* yang digunakan pada program Kelola Kerajaan diimplementasikan pada kelas Storage. Konsep *Template & Generic Classes* digunakan untuk memudahkan fleksibilitas pada tipe data yang berbeda, namun memiliki implementasi *method* yang sama. Selain itu, kelas ini bertanggung jawab untuk menampilkan informasi penyimpanan dari Pemain terkait panen dan juga dari Sellable beserta turunannya.

Kelas	Kode Program
-------	--------------

Storage

```

1  template <class T>
2  class Storage;
3
4  template <class T>
5  void display(const Storage<T> &storage);
6
7  template <class T>
8  map<string, tuple<vector<string>,int>> readyPanen(const Storage<T> &storage);
9
10 template <class T>
11 void displayItems(const Storage<T> &storage);
12
13 template <>
14 void display<Sellable>(const Storage<Sellable> &storage);
15
16 template <>
17 void display<Hewan>(const Storage<Hewan> &storage);
18
19 template <>
20 void display<Tanaman>(const Storage<Tanaman> &storage);
21
22 template <>
23 map<string, tuple<vector<string>,int>> readyPanen<Hewan>(const Storage<Hewan> &storage);
24
25 template <>
26 map<string, tuple<vector<string>,int>> readyPanen<Tanaman>(const Storage<Tanaman> &storage);
27
28 template <>
29 void displayItems<Hewan>(const Storage<Hewan> &storage);
30
31 template <>
32 void displayItems<Tanaman>(const Storage<Tanaman> &storage);

```

Tabel 2.3.1. Template & Generic Classes

2.4. Exception

Konsep *Exception* yang digunakan pada program Kelola Kerajaan diimplementasikan untuk menangani *error* pada suatu kondisi atau syarat yang tidak dipenuhi sehingga jika ada *error* setelah diberikan masukan, program tidak akan langsung berhenti dan hal tersebut memberikan kesan *interface* yang baik kepada pengguna untuk kembali melanjutkan masukan yang seharusnya sehingga aturan pada permainan tetap berlangsung dengan baik hingga akhir permainan. Terdapat beberapa cuplikan dari sebagian implementasi kelas ini, yakni sebagai berikut.

```

1  const char *WalikotaInputException::what()
2  {
3      return "Input yang anda masukkan merupakan milik dari Walikota!\nSilahkan masukkan input yang sesuai dengan role anda!";
4  }
5  const char *PeternakInputException::what()
6  {
7      return "Input yang anda masukkan merupakan milik dari Peternak!\nSilahkan masukkan input yang sesuai dengan role anda!";
8  }
9  const char *PetaniInputException::what()
10 {
11     return "Input yang anda masukkan merupakan milik dari Petani!\nSilahkan masukkan input yang sesuai dengan role anda!";
12 }

```

Gambar 2.4.1. Cuplikan Kode Exception

Nama Exception	Penjelasan Kegunaan Exception
NumericException	Untuk memvalidasi masukan tipe data <i>integer</i>
MoreThanZeroException	Untuk memvalidasi masukan angka bernilai lebih dari 0
InvalidInputException	Untuk memvalidasi masukan perintah yang sesuai dengan aturan permainan

WalikotaInputException	Untuk memvalidasi masukan perintah selain dari walikota
PeternakInputException	Untuk memvalidasi masukan perintah selain dari peternak
PetaniInputException	Untuk memvalidasi masukan perintah selain dari petani
PanenInputException	Untuk memvalidasi masukan perintah selain dari petani atau peternak
InventoryFullException	Untuk memvalidasi masukan perintah saat <i>inventory</i> sudah penuh
LadangFullException	Untuk memvalidasi masukan perintah saat ladang sudah penuh
TernakFullException	Untuk memvalidasi masukan perintah saat peternakan sudah penuh
NotEnoughMoneyException	Untuk memvalidasi masukan perintah saat gulden (uang) tidak cukup
NotEnoughStockException	Untuk memvalidasi masukan perintah saat stok barang tidak cukup (ada atau memenuhi permintaan)
InventoryNotEnoughException	Untuk memvalidasi masukan perintah saat slot <i>inventory</i> tidak cukup
NotEnoughTanamanException	Untuk memvalidasi masukan perintah saat jumlah tanaman tidak cukup
NotEnoughPanenException	Untuk memvalidasi masukan perintah saat jumlah panen tidak cukup
NotEnoughHewanException	Untuk memvalidasi masukan perintah saat jumlah hewan tidak cukup
NotEnoughFoodException	Untuk memvalidasi masukan perintah saat jumlah makanan tidak cukup
NotEnoughMaterialException	Untuk memvalidasi masukan perintah saat jumlah material tidak cukup
InventoryEmptyException	Untuk memvalidasi masukan perintah saat <i>inventory</i> kosong
TernakEmptyException	Untuk memvalidasi masukan perintah saat peternakan kosong

InvalidStorageIndexException	Untuk memvalidasi masukan perintah saat indeks pada matriks
InvalidJenisHewanException	Untuk memvalidasi masukan perintah saat jenis hewan tidak sesuai
InvalidProdukException	Untuk memvalidasi masukan perintah selain dari jenis produk yang dapat dimakan
InvalidJenisMakananException	Untuk memvalidasi masukan perintah selain dari tipe makanan yang dapat dimakan (sesuai tipe hewan: herbivora, karnivora, dan omnivora)
FileNotFoundException	Untuk memvalidasi kondisi saat file yang ingin dibaca tidak ditemukan
InvalidSellException	Untuk memvalidasi kondisi saat bangunan ingin dijual (karena semuanya dapat dijual, kecuali bangunan)

Tabel 2.4.1. Exception

2.5. C++ Standard Template Library

Konsep STL (*Standard Template Library*) dari C++ yang digunakan pada program Kelola Kerajaan diimplementasikan pada banyak kelas, seperti kelas Bangunan yang menerapkan map dan tuple, kelas Toko yang menerapkan vector, tuple, dan pair (utility), serta pada kelas lainnya.

Penjelasan	Kode Program
Map: berisi kumpulan pasangan kunci dan nilai di mana suatu nilai tersebut hanya dapat diakses dari kuncinya.	<p>Contoh: Bangunan.cpp</p> <pre> 1 map<string, tuple<string, int, map<string, int>, int>> Bangunan::getAllRecipe() 2 { 3 return list_of_bangunan; 4 }</pre>

<p>Vector: dapat didefinisikan sebagai array dinamis atau array dengan beberapa fitur tambahan.</p>	<p>Contoh: ProdukHewan.cpp</p> <pre> 1 Sellable* ProdukHewan::tambahProdukHewanHerbivora(Hewan &hewan) 2 { 3 vector<tuple<int, string, string, string, int, int>> produk_herbivora_vector = Produk::productOriginMap[hewan.getNamaBarang()]; 4 5 ProdukHewan* produk_hewan_baru = new ProdukHewan(get<0>(produk_herbivora_vector[0]), 6 get<1>(produk_herbivora_vector[0]), 7 get<2>(produk_herbivora_vector[0]), 8 get<3>(produk_herbivora_vector[0]), hewan.getNamaBarang(), 9 get<4>(produk_herbivora_vector[0]), 10 get<5>(produk_herbivora_vector[0])); 11 delete &hewan; 12 return produk_hewan_baru; 13 }</pre>
<p>Tuple: objek yang dapat menampung sejumlah elemen. Elemen-elemennya dapat berupa tipe data yang berbeda. Elemen-elemen tuple diinisialisasi sebagai argumen dalam urutan di mana mereka akan diakses.</p>	<p>Contoh: Bangunan.cpp</p> <pre> 1 tuple<string, int, map<string, int>, int> Bangunan::getSpecificRecipe(string name) 2 { 3 return list_of_bangunan.find(name)->second; 4 }</pre>
<p>Pair: digunakan untuk menggabungkan dua nilai yang mungkin memiliki tipe data yang berbeda. Pair menyediakan cara untuk menyimpan dua objek heterogen sebagai satu unit. Ini biasanya digunakan jika kita ingin menyimpan tuple.</p>	<p>Contoh: Toko.cpp</p>

	<pre> 1 void Toko::displayAllBuyableItem(string role) 2 { 3 cout << "Selamat datang di toko!!" << endl; 4 5 cout << "Berikut merupakan hal yang dapat Anda Beli : " << endl; 6 vector<pair<Sellable *, int>> temp(items); 7 if (role != "Walikota") 8 { 9 // Concat bangunan into displayed 10 temp.insert(temp.end(), list_bangunan.begin(), list_bangunan.end()); 11 } 12 int i = 1; 13 for (const auto &item : temp) 14 { 15 16 cout << i << ". " << item.first->getNamaBarang() << " - " << item.first->getHargaBarang(); 17 18 if (item.second != -1) 19 { 20 cout << " (" << item.second << ")" << endl; 21 } 22 else 23 { 24 cout << endl; 25 } 26 i++; 27 } 28 } </pre>
--	---

Tabel 2.5.1. C++ Standard Template Library

2.6. Abstract Base Class

Konsep Abstract Base Class diterapkan pada Kelas Pemain dan Sellables. Kelas-kelas tersebut memiliki fungsi *pure virtual*. Pada kelas tersebut setidaknya memiliki satu method abstrak, yang artinya method tidak memiliki implementasi pada kelas *parent*. Pemain merupakan kelas *base* untuk turunan peternak, petani, dan walikota. Sellables merupakan kelas *base* untuk Hewan, Tanaman, Bangunan, dan Produk. *Pure Virtual Method* digunakan pada saat dibutuhkan abstraksi yang lebih lanjut untuk kelas

turunannya. Untuk contoh, pada methodh *virtual void azab()*, digunakan method yang *pure virtual* karena implementasi dari method tersebut berbeda dan bergantung pada kelas turunannya.

Penjelasan	Kode Program
<p>Method abstrak yang terdapat pada Sellable.hpp adalah</p> <ul style="list-style-type: none"> - virtual Sellable *Clone() - virtual string getJenis() 	<p>Contoh: Sellable.hpp</p> <pre> 9 class Sellable 10 { 11 protected: 12 string std::string Sellable::nama_barang 13 string nama_barang; 14 int harga_barang; 15 16 public: 17 // ctor default 18 Sellable(); 19 20 // ctor user defined 21 Sellable(string kode, string nama, int harga); 22 23 // operator overloading 24 Sellable &operator=(const Sellable &other); 25 bool operator==(Sellable &other); 26 27 virtual Sellable *Clone() = 0; 28 29 // dtor 30 virtual ~Sellable(); 31 32 /* Methods */ 33 // getter 34 string getNamaBarang(); 35 string getKodeHuruf(); 36 int getHargaBarang(); 37 virtual string getJenis() = 0; 38 39 // setter 40 void setKodeHuruf(string kode); 41 void setNamaBarang(string nama); 42 void setHargaBarang(int harga); 43 44 // additional 45 virtual bool isEdible(); 46 virtual int getAddedWeight(); 47 }; 48 49 #endif </pre>
Method abstrak yang terdapat pada	Contoh: Pemain.cpp

Pemain.hpp adalah

- virtual string getRole()
- virtual int getKKP()
- virtual void azab() {bonus}

```
class Pemain
{
    *
    * @param pemain vector pemain yang akan dikenakan pajak
    */
    virtual void pungutPajak(const vector<Pemain *> &pemain);
    virtual void cetakLadang();
    virtual void cetakPernakan();
    virtual void tanam();
    virtual void ternak();
    /**
    * @brief Membangun Bangunan, Khusus untuk walikota
    *
    */
    virtual void bangunBangunan();
    virtual void kasihMakan();
    void beli(Toko & toko);
    void jual(Toko & toko);
    virtual void addPlantAge();
    virtual void panen();
    /**
    * @brief Menambah pemain atau peternak, Khusus untuk walikota
    *
    * @param pemain vector pemain yang akan dimanipulasi
    * @return index pemain baru (untuk menentukan urutan main nantinya)
    */
    virtual int tambahPemain(vector<Pemain *> &pemain);
    virtual int getKKP() = 0;
    virtual string getRole() const = 0;

    // Configurations needs

    virtual void setLadang(const Storage<Tanaman> &storage);
    virtual void setPernakan(const Storage<Hewan> &storage);
    virtual Storage<Tanaman> getLadang();
    virtual Storage<Hewan> getPernakan();

    static int getUkuranInventoryN();
    static int getUkuranInventoryM();
    static void setUkuranInventoryN(int n);
    static void setUkuranInventoryM(int m);

    void rezeki();
    virtual void azab() = 0;
};

#endif
```

Tabel 2.6.1. Abstract Base Class

2.7. Aggregation

Konsep *Aggregation* yang digunakan pada program Kelola Kerajaan diimplementasikan pada beberapa kelas, contohnya Toko dan Sellable serta GameWorld dan Pemain. Ketiga pasang kelas tersebut memiliki *attribute* yang merupakan objek dari kelas lain. Implementasi konsep ini terlihat jelas pada cuplikan kode di bawah ini. Konsep ini digunakan untuk memudahkan program dalam menyimpan lebih dari 1 informasi terkait barang-barang yang ingin diperjual-belikan dan juga ketiga jenis pemain yang ada. Selain itu, pemain selain walikota dapat ditambahkan sehingga memerlukan suatu list yang kontigu, yakni vektor.

Penjelasan	Kode Program
Toko dan Sellable: Dalam kelas Toko, terdapat vektor pasangan dari Sellable dan kuantitasnya	<pre> 1 class Toko 2 { 3 public: 4 vector<pair<Sellable *, int>> items; 5 vector<pair<Sellable *, int>> list_bangunan; </pre>
GameWorld dan Pemain: Dalam kelas GameWorld, terdapat vektor yang elemennya berisi pemain-pemain	<pre> 1 class GameWorld 2 { 3 private: 4 vector<Pemain *> listOfPlayers; </pre>

Tabel 2.7.1. Aggregation

2.8. Composition

Konsep *Composition* yang digunakan pada program Kelola Kerajaan diimplementasikan pada kelas `Storage` yang berhubungan dengan seluruh kelas `Pemain` beserta turunannya kecuali `Walikota`, yakni `Peternak` dan `Petani`. Konsep ini diperlukan untuk menampilkan isi dari ladang milik `Petani` dan peternakan milik `Peternak` yang terenkapsulasi dalam kelas `Storage` sehingga pada kelas `Pemain`, `Peternak`, dan `Petani` akan menggunakan *template* kembali dari kelas `Storage`. Konsep ini termasuk *composition* karena `Storage` memerlukan objek dari `Hewan`, `Tanaman`, dan `Sellable` yang terkait juga dengan `Pemain`, `Peternak`, dan `Petani` sedangkan kelas-kelas tersebut juga memerlukan kelas `Storage` untuk mengimplementasikan *method* yang diperlukan. Jika `Storage` dihapus, kelas `Pemain`, `Peternak`, dan `Petani` tidak dapat diimplementasikan dengan baik sesuai program yang diinginkan.

Penjelasan	Kode Program
Storage dan Pemain	<pre> 1 template <> 2 void display<Sellable>(const Storage<Sellable> &storage) 3 { 4 cout << endl; 5 // =====[Penyimpanan]===== 6 cout << " "; 7 int numOfEq = (1 + 6 * storage.col - 15) / 2; // 15 is len([Penyimpanan]) 8 for (int i = 0; i < numOfEq; i++) 9 { 10 cout << "="; 11 } 12 cout << "[Penyimpanan]"; 13 for (int i = 0; i < numOfEq; i++) 14 { 15 cout << "="; 16 } 17 cout << endl; 18 19 // Print the letters for each column 20 cout << " "; 21 for (char c = 'A'; c < 'A' + storage.col; ++c) 22 { 23 cout << " " << c << " "; 24 } 25 cout << " " << endl; 26 27 for (int i = 0; i < storage.col; ++i) 28 { 29 if (i == 0) 30 { 31 cout << " +"; 32 } 33 cout << "-----+"; 34 } 35 cout << endl; </pre>

```
36 for (int i = 0; i < storage.row; ++i)
37 {
38     cout << " " << intToStringWithLeadingZero(i + 1) << " |";
39     for (int j = 0; j < storage.col; ++j)
40     {
41         string keluaran = "";
42         if (storage.buffer[i][j] != nullptr)
43         {
44             keluaran = (*storage.buffer[i][j]).getKodeHuruf();
45         }
46         if (keluaran == "")
47         {
48             keluaran = " ";
49         }
50         cout << " " << keluaran << " |";
51     }
52     cout << endl;
53     cout << "  +";
54     for (int j = 0; j < storage.col; ++j)
55     {
56         cout << "-----";
57     }
58     cout << endl;
59 }
60 }
```


Storage dan Peternak

```
1  template <>
2  void displayItems<Hewan>(const Storage<Hewan> &storage)
3  {
4      set<map<string, string>> Items;
5      for (const auto &innerVector : storage.buffer)
6      {
7          for (Hewan *value : innerVector)
8          {
9              map<string, string> itemMap;
10             if (value != nullptr)
11             {
12                 itemMap[value->getKodeHuruf()] = value->getNamaBarang();
13                 Items.insert(itemMap);
14             }
15         }
16     }
17     for (const auto &item : Items)
18     {
19         for (const auto &pair : item)
20         {
21             std::cout << " - " << pair.first << ": " << pair.second << endl;
22         }
23     }
24 }
```

Storage dan Petani

```

1  template <>
2  void displayItems<Tanaman>(const Storage<Tanaman> &storage)
3  {
4      set<map<string, string>> Items;
5      for (const auto &innerVector : storage.buffer)
6      {
7          for (Tanaman *value : innerVector)
8          {
9              if (value != nullptr)
10             {
11                 map<string, string> itemMap;
12                 if (value != nullptr)
13                 {
14                     itemMap[value->getKodeHuruf()] = value->getNamaBarang();
15                     Items.insert(itemMap);
16                 }
17             }
18         }
19     }
20     for (const auto &item : Items)
21     {
22         for (const auto &pair : item)
23         {
24             std::cout << " - " << pair.first << ": " << pair.second << endl;
25         }
26     }
27 }

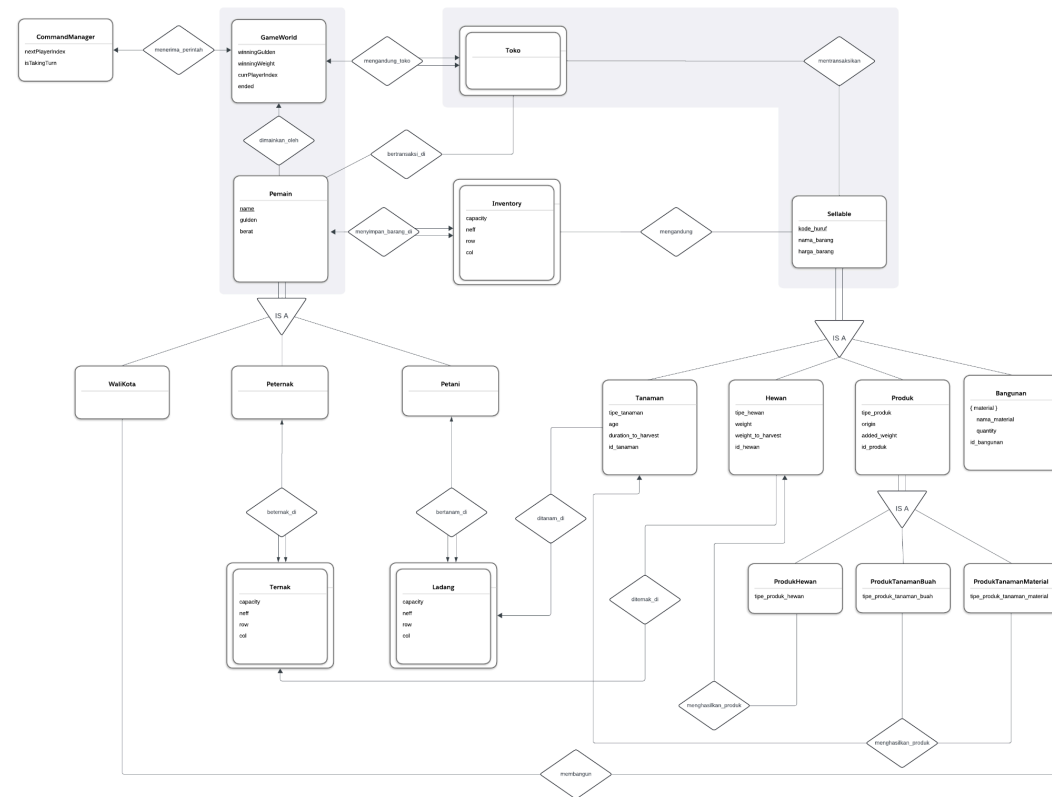
```

Tabel 2.8.1. Composition

3. Bonus Yang dikerjakan

3.1. Bonus yang diusulkan oleh spek

3.1.1. Diagram Sistem



Tabel 3.1.1.1. Entity Relationship Diagram

3.1.2. Kreativitas

3.1.2.1. Azab dan Rezeki

Game akan memiliki mode untuk bermain menggunakan bonus yang diimplementasikan. Pada setiap pemanggilan NEXT atau saat pergantian pemain, program akan mengeluarkan angka acak dari skala 1-13 untuk tiap pemain. Apabila angka tersebut merupakan angka 4, 9, 13 pemain akan mendapatkan “azab”, namun jika angka tersebut merupakan angka 7, pemain akan mendapatkan rezeki. Pemilihan angka 4, 9, dan 13 berdasarkan mayoritas kepercayaan bahwa angka-angka tersebut merupakan angka tidak beruntung. Sebaliknya angka 7 dipilih sebagai penentu rezeki karena mayoritas kepercayaan angka 7 dianggap sebagai angka yang beruntung. Jika angka selain 4, 7, 9, 13 yang didapatkan maka pemain tidak akan mendapatkan apa-apa.

Azab yang didapatkan untuk tiap jenis peran akan berbeda-beda. Untuk peran peternak, program akan menghapus hewan pertama yang ditemukan di peternakan peternak tersebut, namun apabila peternak tersebut tidak memiliki hewan apapun, tidak akan terjadi apa-apa. Sama halnya untuk peran petani, program akan menghapus tanaman pertama yang ditemukan di ladang petani tersebut, namun apabila petani tersebut tidak memiliki tanaman apapun, tidak akan terjadi apa-apa. Khusus, untuk walikota, *gulden*nya akan dikurangi sebanyak $\frac{1}{3}$ total gulden yang dimiliki, namun apabila *gulden* sudah 0, tidak akan terjadi apa-apa.

Rezeki yang diterima untuk tiap jenis peran sama. Gulden pemain akan bertambah sebanyak 10% apabila mendapatkan rezeki. Karena penentuan terkena azab/rezeki dilakukan secara acak, pada saat satu kali pemanggilan *NEXT*, jumlah pemain yang mendapatkan azab/rezeki tidak menentu.

Penjelasan	Kode Program
Rezeki	<pre> void Pemain::rezeki() { cout << CYAN << "Wah, kamu berhasil lari dari amarah dewa siwa!!" << endl; << "Dewa siwa sedang baik hati nich (~v~)(~v~)(~v~)" << RESET << endl; int tambahan = (int)(0.1 * gulden); cout << name << " mendapatkan tambahan uang sebanyak " << CYAN << tambahan << RESET << " gulden \$\$\$" << endl; gulden += tambahan; } </pre>

Azab untuk petani

```

0 void Petani::azab()
1 {
2     if (!ladang.isEmpty())
3     {
4         string namaTanaman;
5         bool found = false;
6         for (int i = 0; i < ladang.getRow(); i++)
7         {
8             for (int j = 0; j < ladang.getCol(); j++)
9             {
10                 Tanaman *item = ladang.getElementAt(i, j);
11                 if (item != nullptr)
12                 {
13                     namaTanaman = item->getNamaBarang();
14                     ladang.deleteAt(i, j);
15                     delete item;
16                     found = true;
17                     break;
18                 }
19             }
20             if (found)
21                 break;
22         }
23         cout << endl;
24         cout << BOLD << "NAKAL KAMU " << CYAN << name << RESET << MAGENTA << endl;
25         cout << "Wakaw dewa siwa marah!!! Tanaman " << BOLD << namaTanaman << RESET << MAGENTA << " kena serangan hama!!" << RESET << endl;
26         cout << MAGENTA << "^(^v^v) Bye-Bye~ " << namaTanaman << RESET << endl;
27     }
28     else
29     {
30         cout << endl;
31         cout << YELLOW << "Tadinya " << BOLD << name << RESET << YELLOW << " membuat dewa siwa marah!!!,nnamun karena " << name << " tidak memiliki tanaman lagi dewa siwa" << endl;
32     }
33 }

```

Azab untuk peternak

```

0 void Peternak::azab()
1 {
2     if (!peternakan.isEmpty())
3     {
4         string namaHewan;
5         bool found = false;
6         for (int i = 0; i < peternakan.getRow(); i++)
7         {
8             for (int j = 0; j < peternakan.getCol(); j++)
9             {
10                 Hewan *item = peternakan.getElementAt(i, j);
11                 if (item != nullptr)
12                 {
13                     namaHewan = item->getNamaBarang();
14                     peternakan.deleteAt(i, j);
15                     delete item;
16                     found = true;
17                     break;
18                 }
19             }
20             if (found)
21                 break;
22         }
23         cout << endl;
24         cout << BOLD << "NAKAL KAMU " << CYAN << name << RESET << MAGENTA << endl;
25         cout << "Wakaw dewa siwa marah!!! Hewan " << BOLD << namaHewan << RESET << MAGENTA << " kaburr!!" << endl;
26         cout << MAGENTA << "^(^v^v) Bye-Bye~ " << namaHewan << RESET << endl;
27     }
28     else
29     {
30         cout << endl;
31         cout << YELLOW << "Tadinya " << BOLD << name << RESET << YELLOW << " membuat dewa siwa marah!!!,nnamun karena " << name << " tidak memiliki hewan lagi dewa siwa" << endl;
32     }
33 }

```

<h1>Azab untuk walikota</h1>	<pre> 1 void Walikota:azab() 2 { 3 if (gulden != 0) 4 { 5 this->gulden = (int) (gulden * 0.67); 6 cout << endl; 7 cout << BOLD << "NAKAL KAMU " << CYAN << name << RESET << MAGENTA << endl; 8 cout << "Wakwaw dewa siwa marah!!! Kamu terciduk KPK (> " << " << endl; 9 cout << "Karena koneksi sebagai walikota banyak, guldenmu hanya berkurang" << BOLD << " 1/3 " << RESET << MAGENTA << "dari total keseluruhan" << RESET << endl; 10 } 11 else 12 { 13 cout << endl; 14 cout << YELLOW << "Tadinya " << BOLD << name << RESET << YELLOW << " membuat dewa siwa marah!!,\nnamun karena " << name << " tidak memiliki gulden lagi dewa siwa 15 } 16 } </pre>
<h1>Contoh berjalannya fungsi azab/rezeki</h1>	<pre> > NEXT (☹️☹️☹️) Dewa Siwa Sedang Marah! Azab akan diberikan kepada pemain yang sial !! Sepertinya para pemain sedang beruntung! Kemarahan Dewa Siwa tidak berdampak kepada siapapun ! Tidak ada pemain yang mendapatkan azab > NEXT (☹️☹️☹️) Dewa Siwa Sedang Marah! Azab akan diberikan kepada pemain yang sial !! Tadinya Jongrang membuat dewa siwa marah!!, namun karena Jongrang tidak memiliki hewan lagi dewa siwa kasian o(^ ☹️ ^)☹️ Hehehe... Jongrang tidak terkena apa apa NAKAL KAMU Roro Wakwaw dewa siwa marah!!! Tanaman ORANGE_TREE kena serangan hama!! ☹️ (☹️☹️☹️) Bye~Bye~ ORANGE_TREE (☹️☹️☹️) Dewa Siwa Sedang Marah! Azab akan diberikan kepada pemain yang sial !! Wah, kamu berhasil lari dari amarah dewa siwa!! Dewa siwa sedang baik hati nich (☹️☹️☹️)~(☹️☹️☹️)~ Budi mendapatkan tambahan uang sebanyak 100 gulden \$\$\$ Wah, kamu berhasil lari dari amarah dewa siwa!! Dewa siwa sedang baik hati nich (☹️☹️☹️)~(☹️☹️☹️)~ Jongrang mendapatkan tambahan uang sebanyak 10 gulden \$\$\$ Sepertinya para pemain sedang beruntung! Kemarahan Dewa Siwa tidak berdampak kepada siapapun ! Tidak ada pemain yang mendapatkan azab </pre>

Tabel 3.1.2.1.1. Azab dan Rezeki

3.1.2.2. Ascii Art dan Output Bewarna

Ascii Art dan Output Berwarna diimplementasikan untuk meningkatkan visibilitas dari keluaran program.

[illegible]

Output bewarna

```

> NEXT

(♯_▯▯▯) Dewa Siwa Sedang Marah! Azab akan diberikan kepada pemain yang sial !!

Sepertinya para pemain sedang beruntung!
Kemarahan Dewa Siwa tidak berdampak kepada siapapun ! Tidak ada pemain yang mendapatkan azab

Sekarang giliran Budi untuk melakukan aksi.

> CETAK_PETERNAKAN
=====
      A      B      C      D      E      F      G      H      I
+-----+-----+-----+-----+-----+-----+-----+-----+
01 |  CHK  |      |  SHP  |      |      |      |      |      |
+-----+-----+-----+-----+-----+-----+-----+-----+
02 |      |      |      |      |  DCK  |      |      |      |
+-----+-----+-----+-----+-----+-----+-----+-----+
03 |      |      |      |      |      |      |      |      |
+-----+-----+-----+-----+-----+-----+-----+-----+
04 |      |      |      |      |      |      |      |      |
+-----+-----+-----+-----+-----+-----+-----+-----+
05 |      |      |      |      |      |      |      |      |
+-----+-----+-----+-----+-----+-----+-----+-----+
06 |      |      |      |      |      |      |      |      |
+-----+-----+-----+-----+-----+-----+-----+-----+
07 |      |      |      |      |      |      |      |      |
+-----+-----+-----+-----+-----+-----+-----+-----+
08 |      |      |      |      |      |      |      |      |
+-----+-----+-----+-----+-----+-----+-----+-----+
- CHK: CHICKEN
- DCK: DUCK
- SHP: SHEEP

> NEXT

(♯_▯▯▯) Dewa Siwa Sedang Marah! Azab akan diberikan kepada pemain yang sial !!

Tadinya Jongrang membuat dewa siwa marah!!,
namun karena Jongrang tidak memiliki hewan lagi dewa siwa kasian o( ^▯▯ ^ )> Hehehe...
Jongrang tidak terkena apa apa

```

Tabel 3.1.2.2.1. Ascii Art dan Output Bewarna

4. Pembagian Tugas

Modul (dalam poin spek)	Implementer	Tester
GameWorld	13522058 13522108	13522042 13522048 13522058 13522080 13522108
CommandManager	13522058 13522108	13522042 13522048 13522058 13522080 13522108
Exception	13522058	13522042 13522048 13522058 13522080 13522108
Toko	13522058 13522108	13522042 13522048 13522058 13522080 13522108
Sellable	13522042	13522042 13522048 13522058 13522080 13522108

Tanaman	10023608 13522042	10023608 13522042 13522048 13522058 13522080 13522108
Hewan	13522042	13522042 13522048 13522058 13522080 13522108
Produk	13522042 13522108	13522042 13522048 13522058 13522080 13522108
ProdukHewan	13522042	13522042 13522048 13522058 13522080 13522108
ProdukTanamanBuah	13522042	13522042 13522048 13522058 13522080 13522108
ProdukTanamanMaterial	13522042	13522042 13522048 13522058

		13522080 13522108
Bangunan	13522042 13522058	13522042 13522048 13522058 13522080 13522108
Storage	13522048 13522080	13522042 13522048 13522058 13522080 13522108
Pemain	13522048 13522080 13522058	13522042 13522048 13522058 13522080 13522108
WaliKota	13522058	13522042 13522048 13522058 13522080 13522108
Peternak	13522048	13522042 13522048 13522058 13522080 13522108
Petani	13522080	13522042

		13522048 13522058 13522080 13522108
Laporan	13522042, 13522048, 13522058, 13522080, 13522108	

5. Lampiran

Link Github: <https://github.com/ImmanuelSG/N30-Walaoeh>

Form Asistensi Tugas Besar
IF2210/Pemrograman Berorientasi Objek
Sem. 4 2023/2024

No. Kelompok/Kelas : N30/K02
 Nama Kelompok : Walaoch
 Anggota Kelompok (Nama/NIM) : 1. Tazkirah Amaliah / 10023608
 2. Amalia Putri / 13522042
 3. Angelica Kierra Ninta Gurning / 13522048
 4. Imanuel Sebastian Girsang / 13522058
 5. Julian Chandra Sutadi / 13522080
 6. Muhammad Neo Cicero Koda / 13522108
 Asisten Pembimbing : M Syahrul Surya putra / 13520161

- Kib tambah player baru, untuk sesuai lekstografi

Tanggal : Kamis, 4 April 2024	Catatan Asistensi:
Tempat : Labpro	
Kehadiran Anggota Kelompok:	
1. Tazkirah Amaliah 10023608	- Ladang, inventory dll ↳ dibebaskan cara implementasi kosongnya
2. Amalia Putri 13522042	- Makefile ↳ coba explore cara pak makefile, backup manual walaupun ga diamanin
3. Angelica Kierra Ninta Gurning 13522048	- Product pisan ↳ plant & hewan ↳ yang bisa dimakan dan untuk dibangun → bisa edible/not edible
4. Imanuel Sebastian Girsang 13522058	- Pikin cara parse buat config - Product cukup di game manager sebenarnya → konfigurasi → sebaiknya simpen di game manager [buat nyimpen data statusnya]
5. Julian Chandra Sutadi 13522080	→ Tipe material plant all gak bakal diubah, yang diubah palingan duration to harvest all. (simpen bentuk vektor?) → Pisan Product → buat edible
6. Muhammad Neo Cicero Koda 13522108	→ Load config dulu, load sama yang udah di save state → Commands buat class connect ke game world
M Syahrul Surya Putra / 13520161	Tanda Tangan Asisten:

- Asisten → bonus kreativitas / bonus
 Gmn klo ada command manager?
 ↳ game world udah ada player trus cek akses → compare struct
~~misal klo ke harvest tau orgnya gmn klo ke harvest dia apa tanpa cek agaknya~~
 Game exception hubungin ke game world / command manager
 Prosedural → calling function by function
 OOP → cmn nembak method objek, klo harvest, harvest doang, klo method
 butuh method lain udah rengahak ke prosedural
 kreasi → dinilai komplektnya
 Ada yg validasi dan ada yang throw exception