Kode Kelompok : BRO
Nama Kelompok : Nama Kelompok
1. 13522014 / Raden Rafly Hanggaraksa B
2. 13522057 / Moh Fairuz Alauddin Yahya
3. 13522066 / Nyoman Ganadipa Narayana
4. 13522084 / Dhafin Fawwaz Ikramullah
5. 13522095 / Rayhan Fadhlan Azka
Asisten Pembimbing : Vincent Prasetiya Atmadja

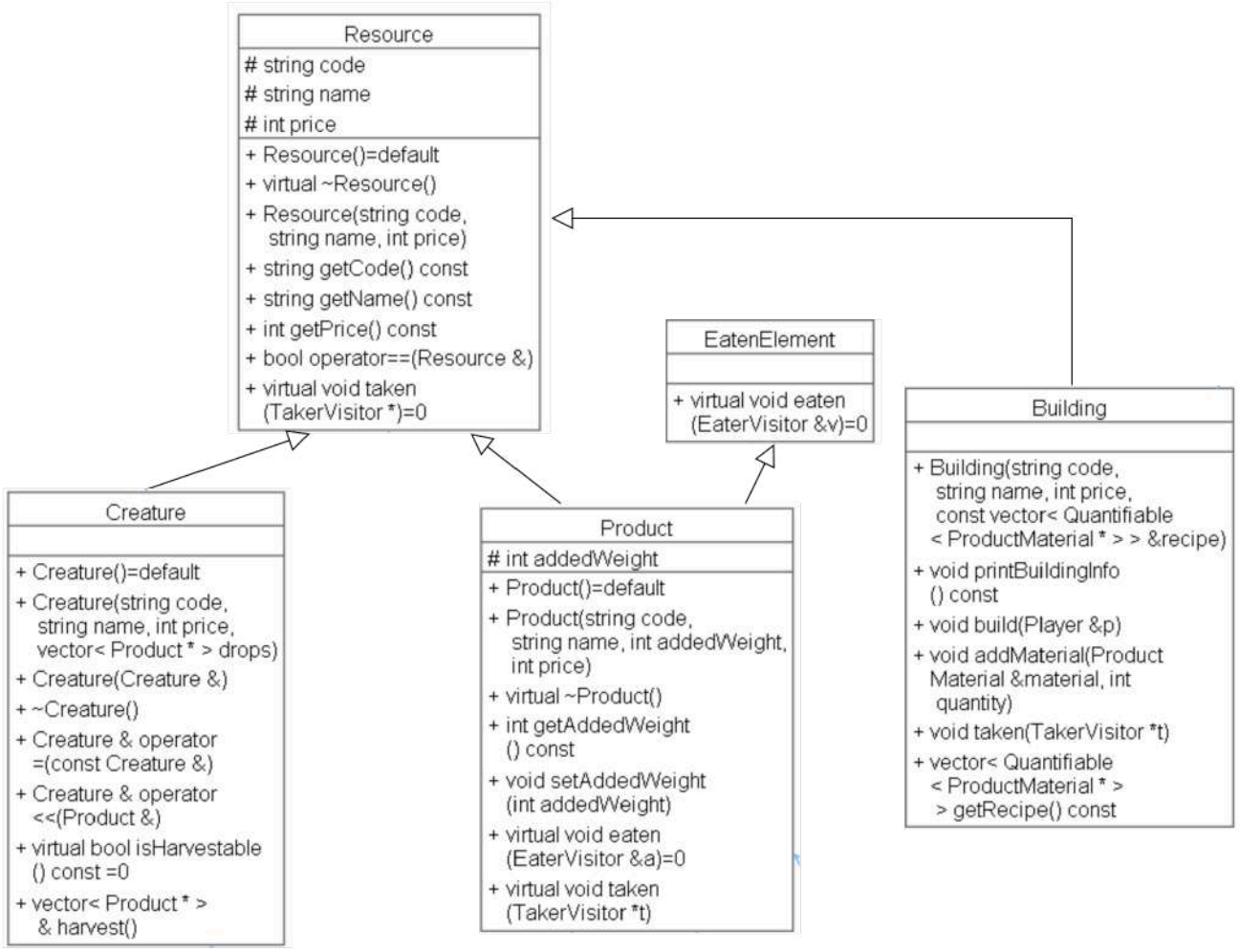
1. Diagram Kelas

Untuk kepentingan agar tulisan dan garis dapat terlihat jelas kami membagi penggambaran diagram menjadi 2 tipe disini, yaitu relasi inheritance dan dependence. Diagram lengkap dapat dilihat pada lampiran. Penggunaan relasi inheritance dipilih karena adanya kebutuhan untuk sharing informasi atribut antar kelas dengan beberapa perbedaan pada masing-masing kelas sesuai kebutuhan. Dengan menggunakan relasi inheritance, kita dapat memanfaatkan fitur sharing informasi yang ada pada kelas induk, memberikan fleksibilitas untuk menambahkan atribut atau perilaku yang spesifik pada kelas turunan secara terpisah. Kelebihan dari pendekatan ini adalah meningkatkan reusabilitas kode, mengurangi duplikasi, dan mempercepat pengembangan sistem dengan menyediakan hierarki yang jelas dan mudah dipahami.

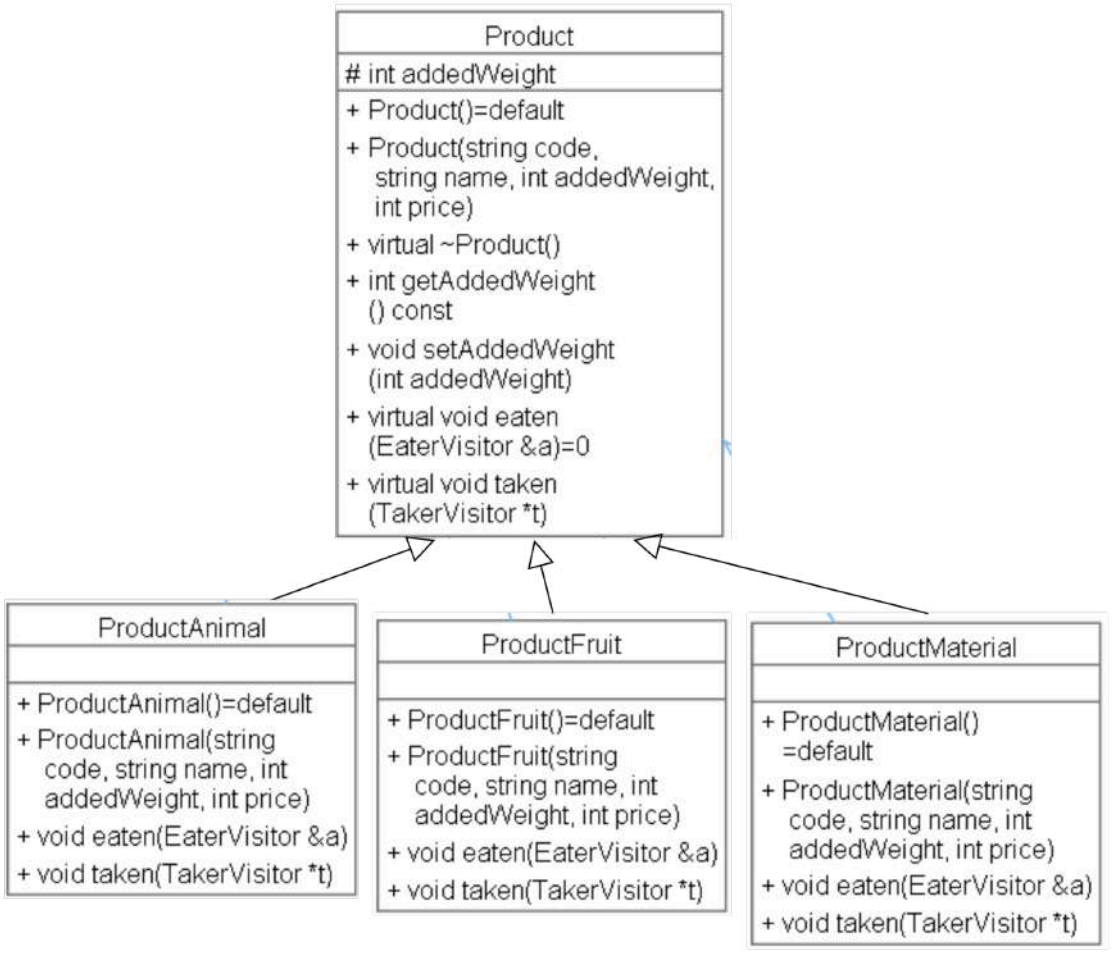
Penggunaan relasi dependence dipilih dalam desain kelas ketika objek atau kelas satu bergantung pada kelas lain, namun kedua kelas tersebut memiliki kepentingan yang berbeda dan tidak memungkinkan untuk diwariskan atau digabungkan. Relasi dependence membantu dalam memisahkan fungsi-fungsi yang berbeda ke dalam kelas-kelas yang terpisah, meningkatkan decoupling dan modularity pada sistem. Dengan pendekatan ini, pengembangan dan pemeliharaan kode menjadi lebih efisien karena setiap kelas bertanggung jawab atas tugas-tugas yang terpisah.

Kendala yang sempat dialami dalam pemilihan desain ini adalah dalam menentukan tingkat kedalaman pewarisan class, khususnya apakah suatu class harus diturunkan hingga ke child object terkecil atau tidak. Hal ini memicu pertanyaan tentang kapan sebaiknya memisahkan class dan kapan sebaiknya tidak.

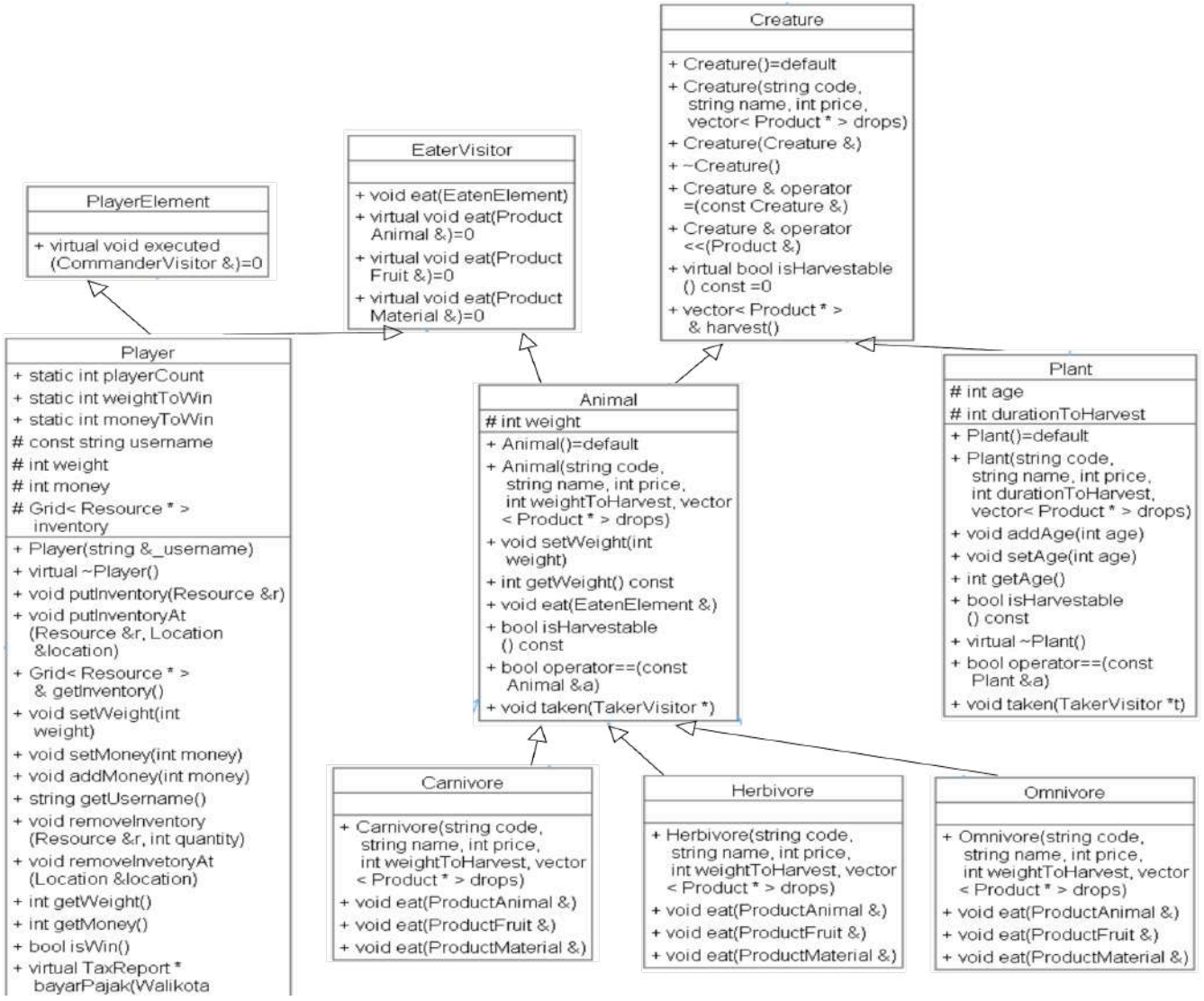
1.1. Relasi Inheritance



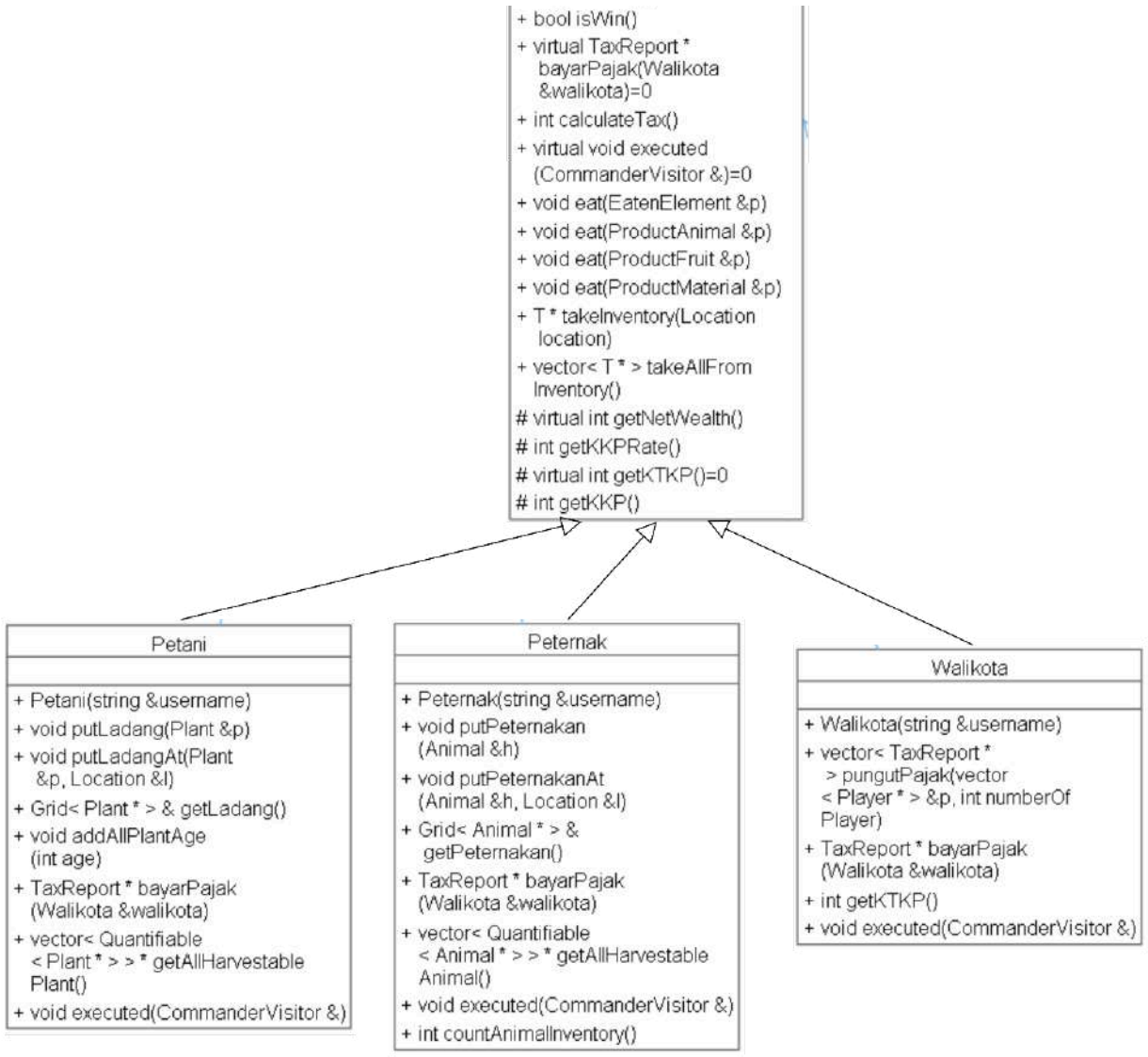
Gambar 1.1.1 Diagram Class Relasi Inheritance Resource dan EatenElement Beserta Childnya



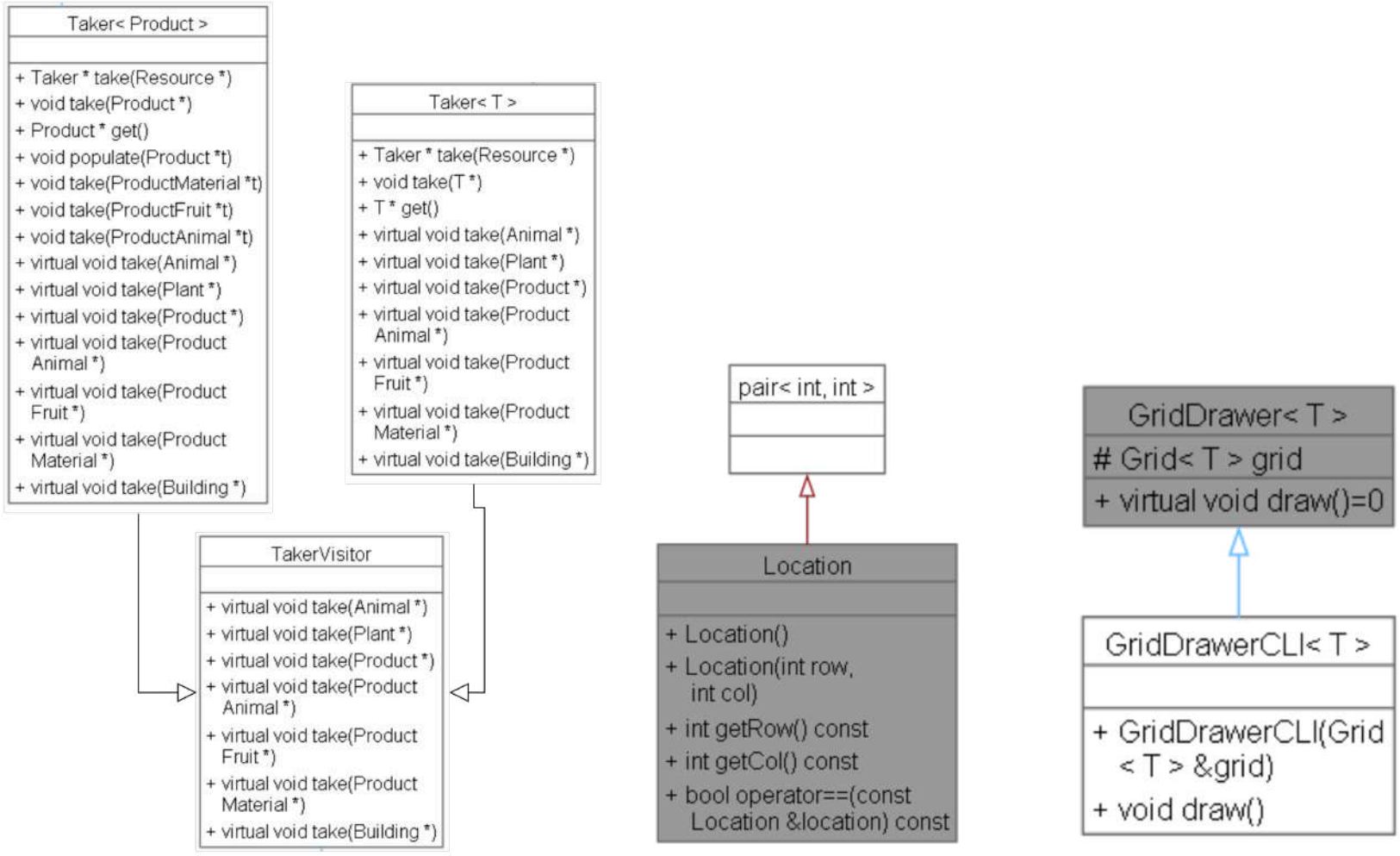
Gambar 1.1.2 Diagram Class Relasi Inheritance Product dan Childnya



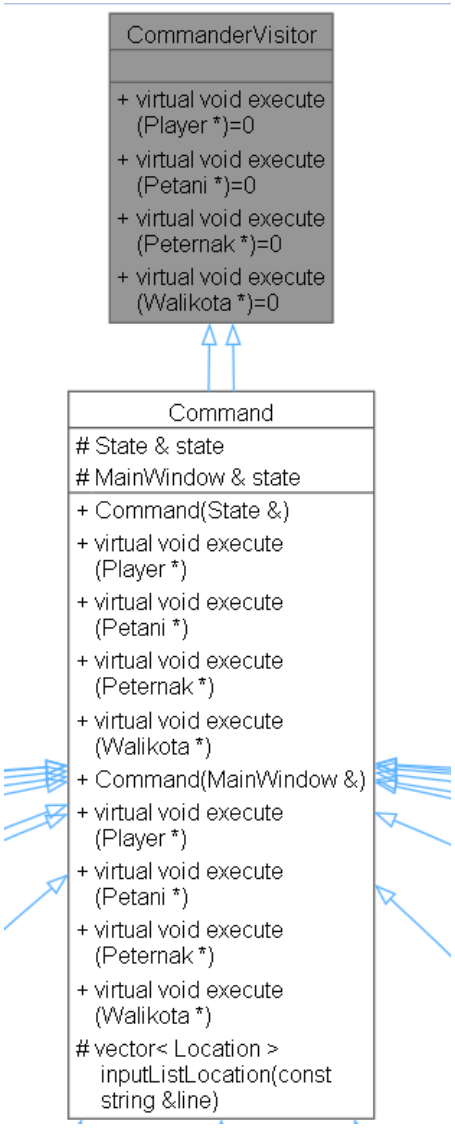
Gambar 1.1.3 Diagram Class Relasi Inheritance Creature, EaterVisitor, Animal, PlayerElement, beserta Childnya


+ bool isWin()
+ virtual TaxReport * bayarPajak(Walikota &walikota)=0
+ int calculateTax()
+ virtual void executed (CommanderVisitor &)=0
+ void eat(EatenElement &p)
+ void eat(ProductAnimal &p)
+ void eat(ProductFruit &p)
+ void eat(ProductMaterial &p)
+ T * takeInventory(Location location)
+ vector< T * > takeAllFrom Inventory()
# virtual int getNetWealth()
# int getKKPRate()
# virtual int getKTKP()=0
# int getKKP()
Petani
+ Petani(string &username)
+ void putLadang(Plant &p)
+ void putLadangAt(Plant &p, Location &l)
+ Grid< Plant * > & getLadang()
+ void addAllPlantAge (int age)
+ TaxReport * bayarPajak (Walikota &walikota)
+ vector< Quantifiable < Plant * > > * getAllHarvestable Plant()
+ void executed(CommanderVisitor &)
Peternak
+ Peternak(string &username)
+ void putPeternakan (Animal &h)
+ void putPeternakanAt (Animal &h, Location &l)
+ Grid< Animal * > & getPeternakan()
+ TaxReport * bayarPajak (Walikota &walikota)
+ vector< Quantifiable < Animal * > > * getAllHarvestable Animal()
+ void executed(CommanderVisitor &)
+ int countAnimalInventory()
Walikota
+ Walikota(string &username)
+ vector< TaxReport * > pungutPajak(vector < Player * > &p, int numberOf Player)
+ TaxReport * bayarPajak (Walikota &walikota)
+ int getKTKP()
+ void executed(CommanderVisitor &)

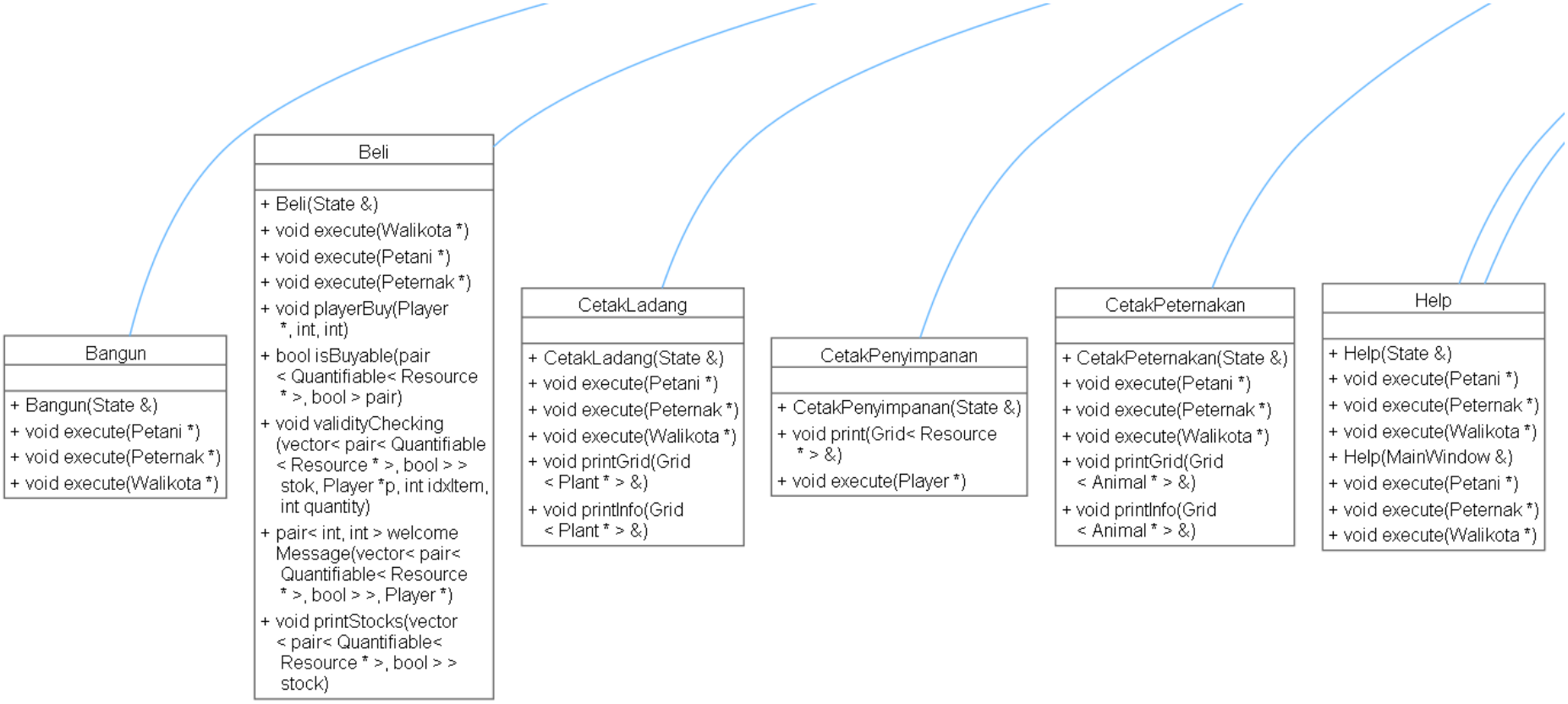
Gambar 1.1.4 Lanjutan Diagram Class Relasi Inheritance Player dan Childnya



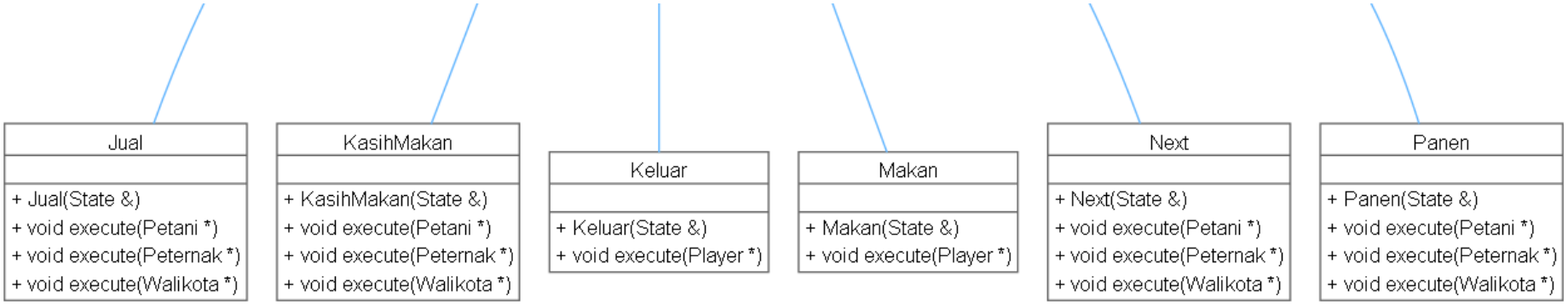
Gambar 1.1.5 Diagram Class Relasi Inheritance TakerVisitor, Location, GridDrawer dan Childnya



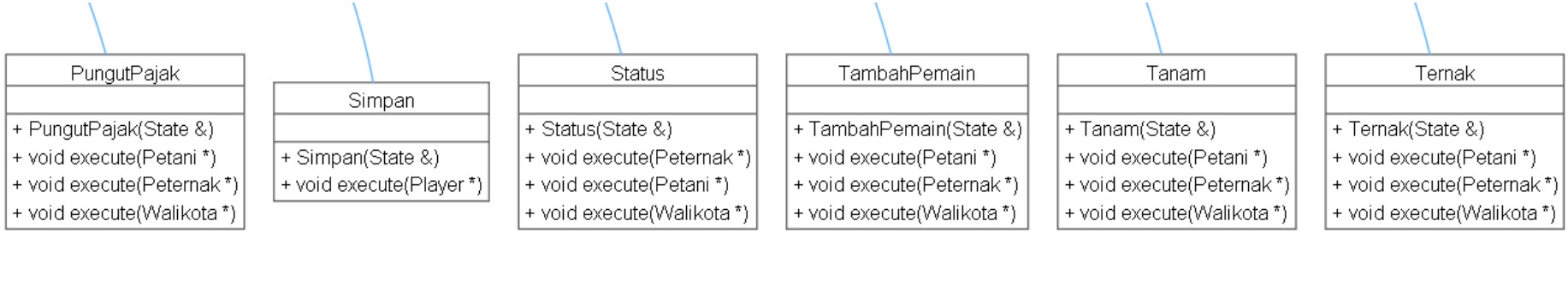
Gambar 1.1.6 Diagram Class Relasi Inheritance CommanderVisitor dan Childnya



Gambar 1.1.7 Diagram Class Relasi Inheritance Child dari Kelas Command

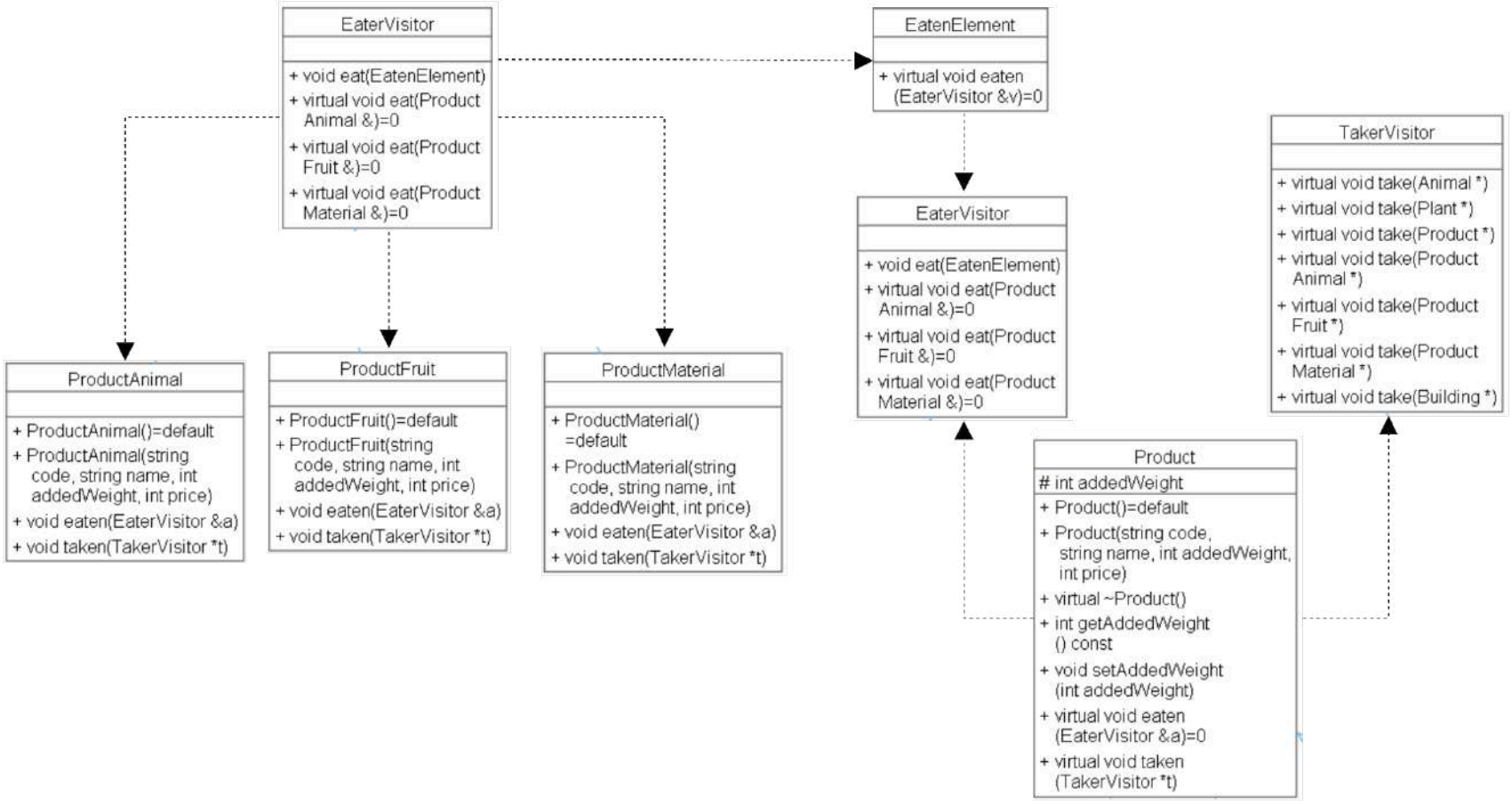


Gambar 1.1.8 Lanjutan 1 Diagram Class Relasi Inheritance Child dari Kelas Command

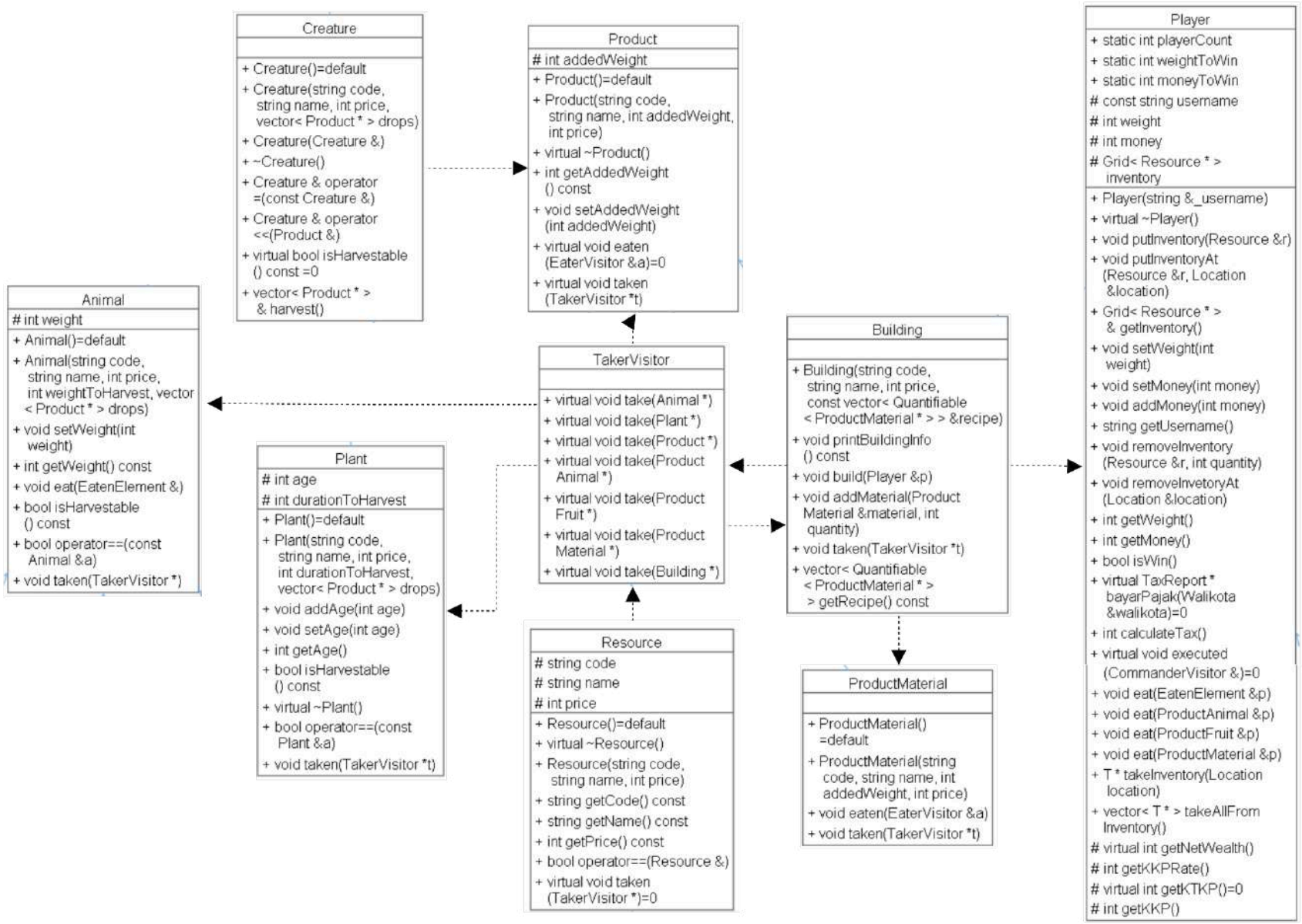


Gambar 1.1.9 Lanjutan 2 Diagram Class Relasi Inheritance Child dari Kelas Command

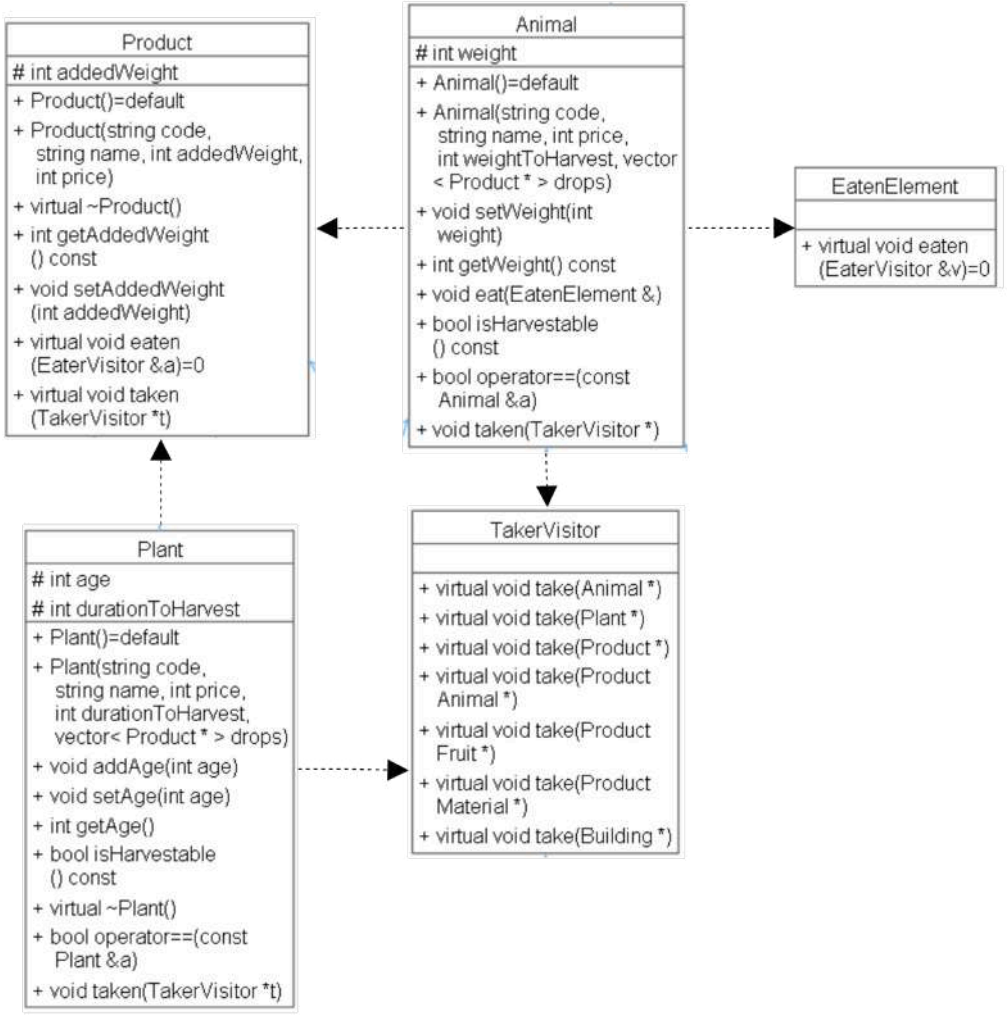
1.2. Relasi Dependence



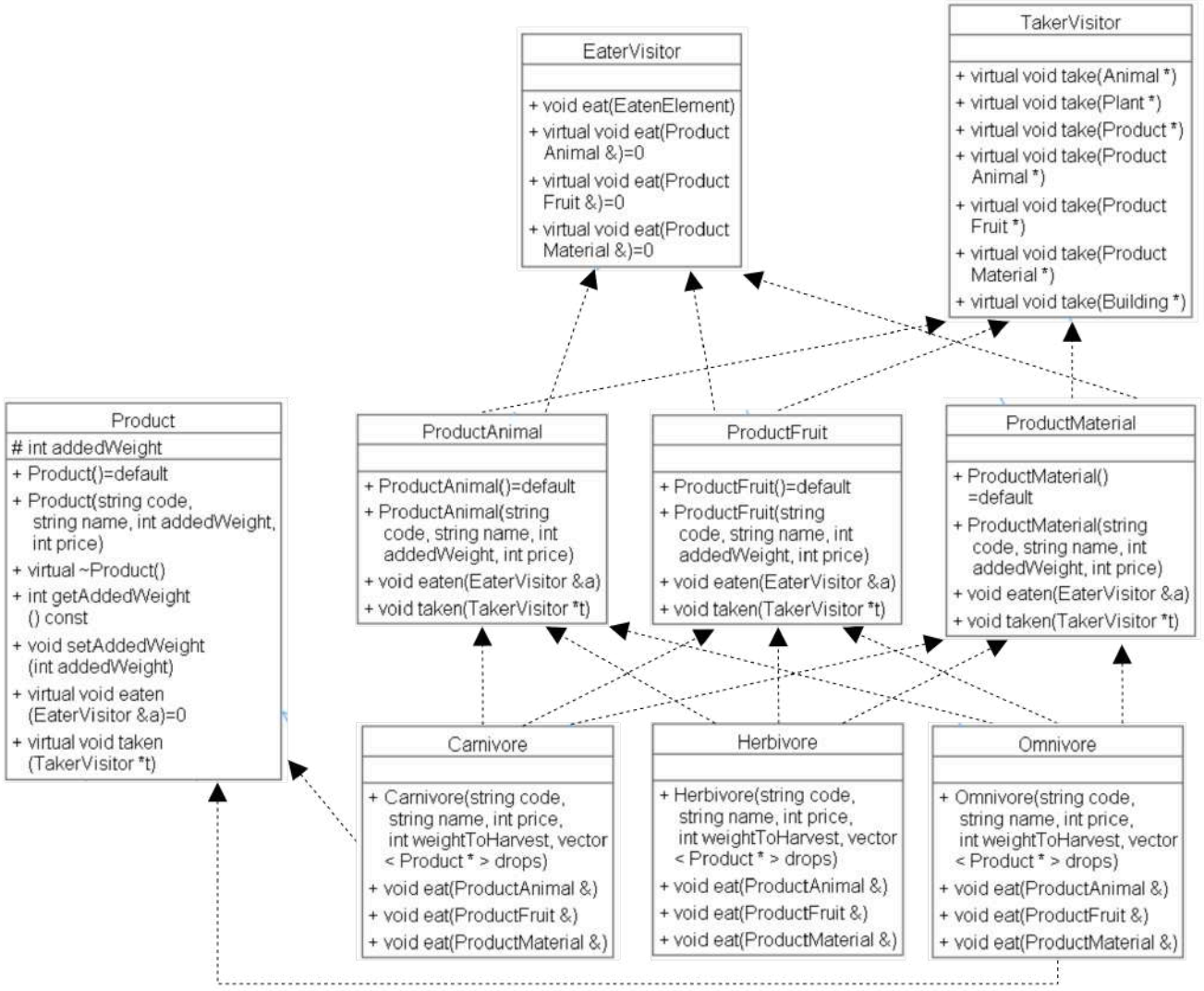
Gambar 1.2.1 Diagram Class Relasi Dependence Kelas EatVisitor, EaternElement, dan Product


Creature
+ Creature()=default
+ Creature(string code, string name, int price, vector< Product * > drops)
+ Creature(Creature &)
+ ~Creature()
+ Creature & operator =(const Creature &)
+ Creature & operator <<(Product &)
+ virtual bool isHarvestable () const =0
+ vector< Product * > & harvest()
Product
int addedWeight
+ Product()=default
+ Product(string code, string name, int addedWeight, int price)
+ virtual ~Product()
+ int getAddedWeight () const
+ void setAddedWeight (int addedWeight)
+ virtual void eaten (EaterVisitor &a)=0
+ virtual void taken (TakerVisitor *t)
Player
+ static int playerCount
+ static int weightToWin
+ static int moneyToWin
# const string username
# int weight
# int money
# Grid< Resource * > inventory
+ Player(string &_username)
+ virtual ~Player()
+ void putInventory(Resource &r)
+ void putInventoryAt (Resource &r, Location &location)
+ Grid< Resource * > & getInventory()
+ void setWeight(int weight)
+ void setMoney(int money)
+ void addMoney(int money)
+ string getUsername()
+ void removeInventory (Resource &r, int quantity)
+ void removeInvetoryAt (Location &location)
+ int getWeight()
+ int getMoney()
+ bool isWin()
+ virtual TaxReport * bayarPajak(Walikota &walikota)=0
+ int calculateTax()
+ virtual void executed (CommanderVisitor &)=0
+ void eat(EatenElement &p)
+ void eat(ProductAnimal &p)
+ void eat(ProductFruit &p)
+ void eat(ProductMaterial &p)
+ T * takeInventory(Location location)
+ vector< T * > takeAllFrom Inventory()
# virtual int getNetWealth()
# int getKKPRate()
# virtual int getKTKP()=0
# int getKKP()
Animal
int weight
+ Animal()=default
+ Animal(string code, string name, int price, int weightToHarvest, vector < Product * > drops)
+ void setWeight(int weight)
+ int getWeight() const
+ void eat(EatenElement &)
+ bool isHarvestable () const
+ bool operator==(const Animal &a)
+ void taken(TakerVisitor *)
TakerVisitor
+ virtual void take(Animal *)
+ virtual void take(Plant *)
+ virtual void take(Product *)
+ virtual void take(Product Animal *)
+ virtual void take(Product Fruit *)
+ virtual void take(Product Material *)
+ virtual void take(Building *)
Building
+ Building(string code, string name, int price, const vector< Quantifiable < ProductMaterial * > > &recipe)
+ void printBuildingInfo () const
+ void build(Player &p)
+ void addMaterial(Product Material &material, int quantity)
+ void taken(TakerVisitor *t)
+ vector< Quantifiable < ProductMaterial * > > getRecipe() const
Plant
int age
int durationToHarvest
+ Plant()=default
+ Plant(string code, string name, int price, int durationToHarvest, vector< Product * > drops)
+ void addAge(int age)
+ void setAge(int age)
+ int getAge()
+ bool isHarvestable () const
+ virtual ~Plant()
+ bool operator==(const Plant &a)
+ void taken(TakerVisitor *t)
Resource
string code
string name
int price
+ Resource()=default
+ virtual ~Resource()
+ Resource(string code, string name, int price)
+ string getCode() const
+ string getName() const
+ int getPrice() const
+ bool operator==(Resource &)
+ virtual void taken (TakerVisitor *)=0
ProductMaterial
+ ProductMaterial() =default
+ ProductMaterial(string code, string name, int addedWeight, int price)
+ void eaten(EaterVisitor &a)
+ void taken(TakerVisitor *t)

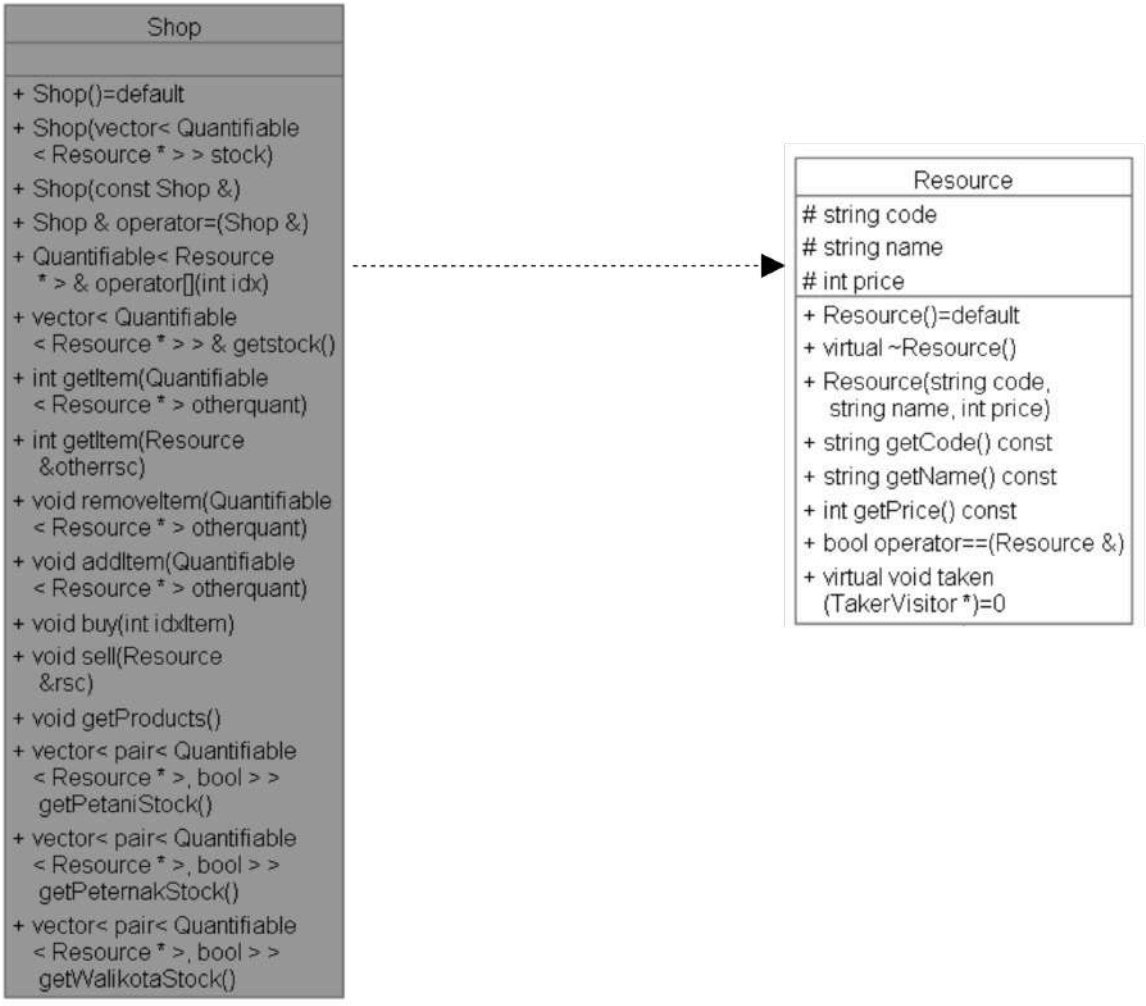
Gambar 1.2.2 Diagram Class Relasi Dependence Kelas Creature, TakerVisitor, Resource, dan Building



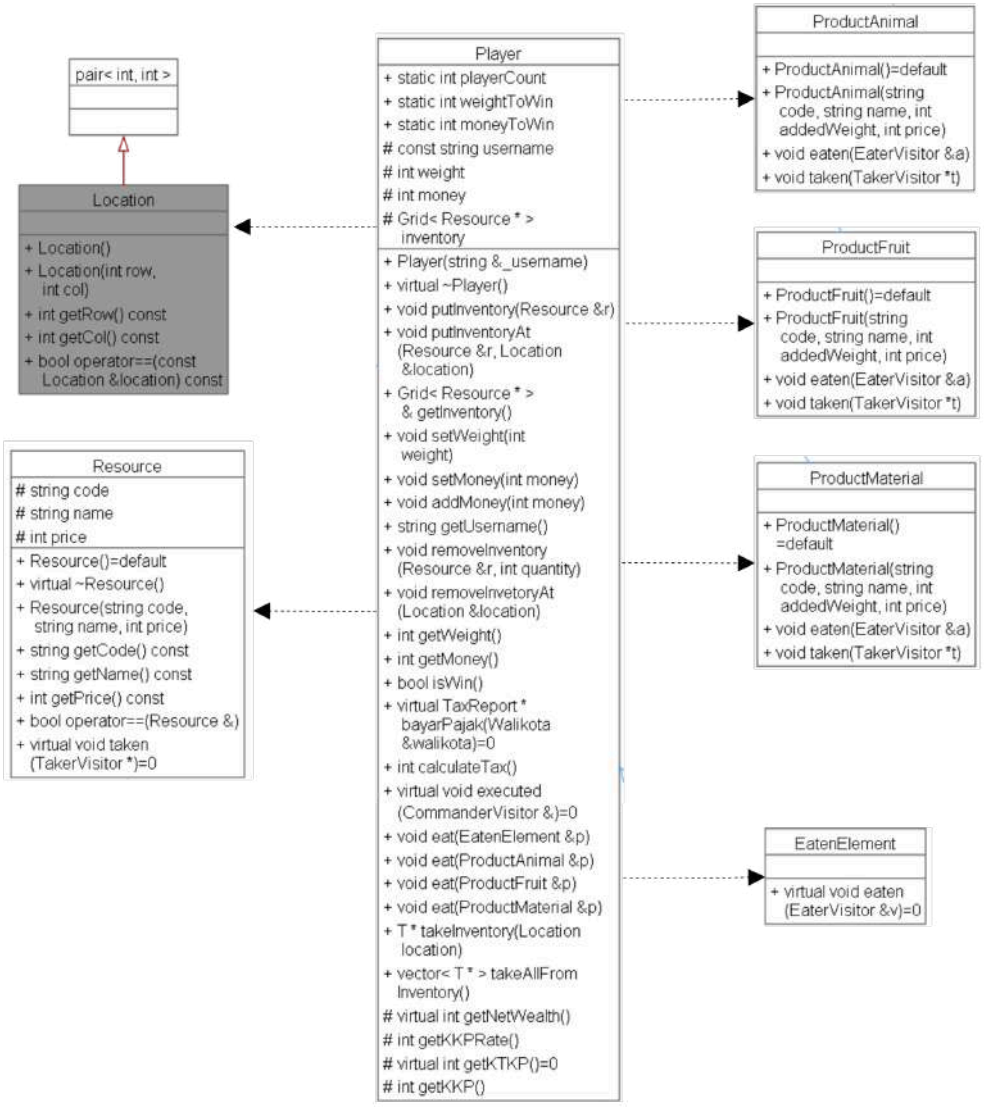
Gambar 1.2.3 Diagram Class Relasi Dependence Kelas Plant dan Animal



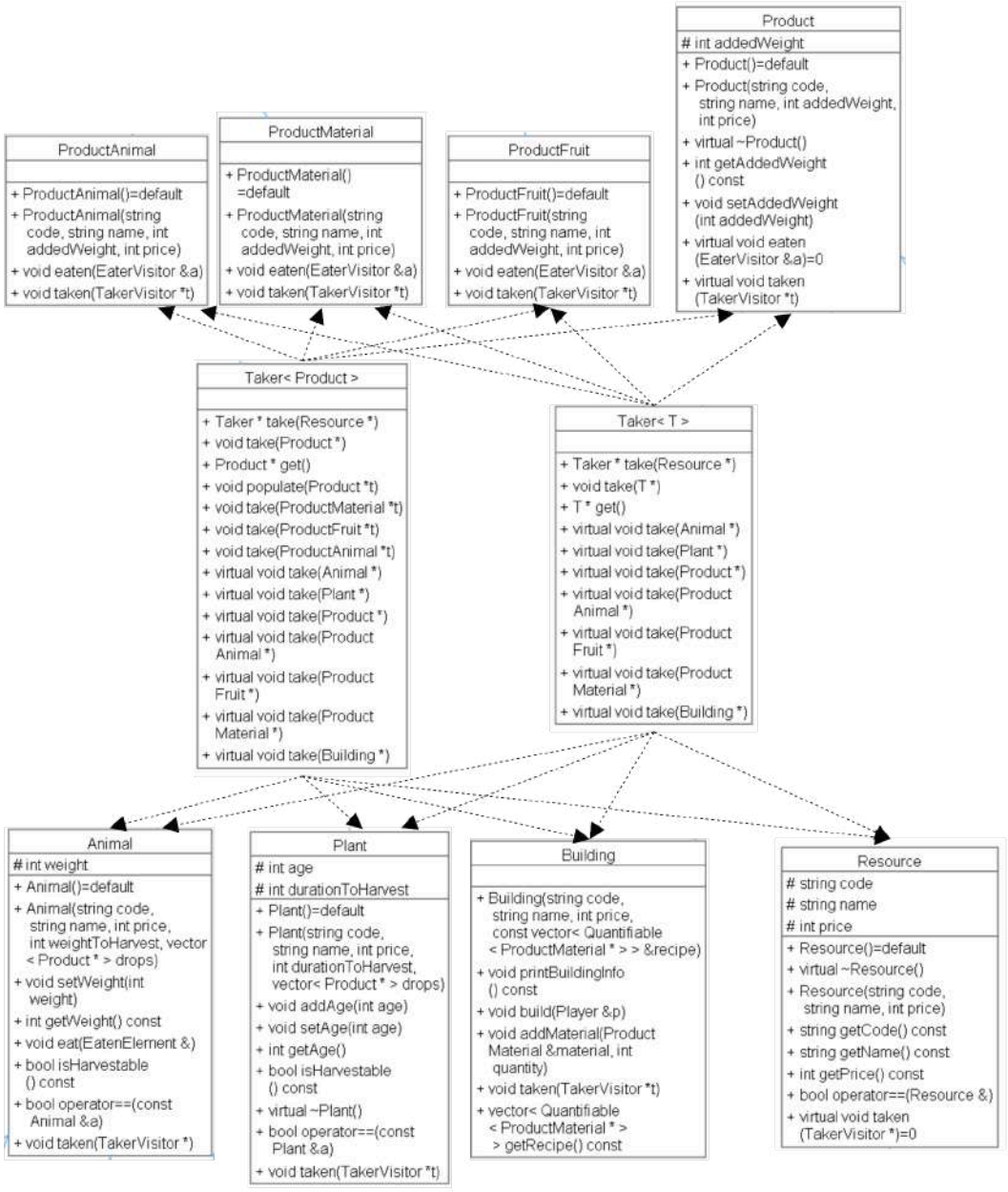
Gambar 1.2.4 Diagram Class Relasi Dependence Kelas ProductAnimal, ProductFruit, ProductMaterial dan Childnya.



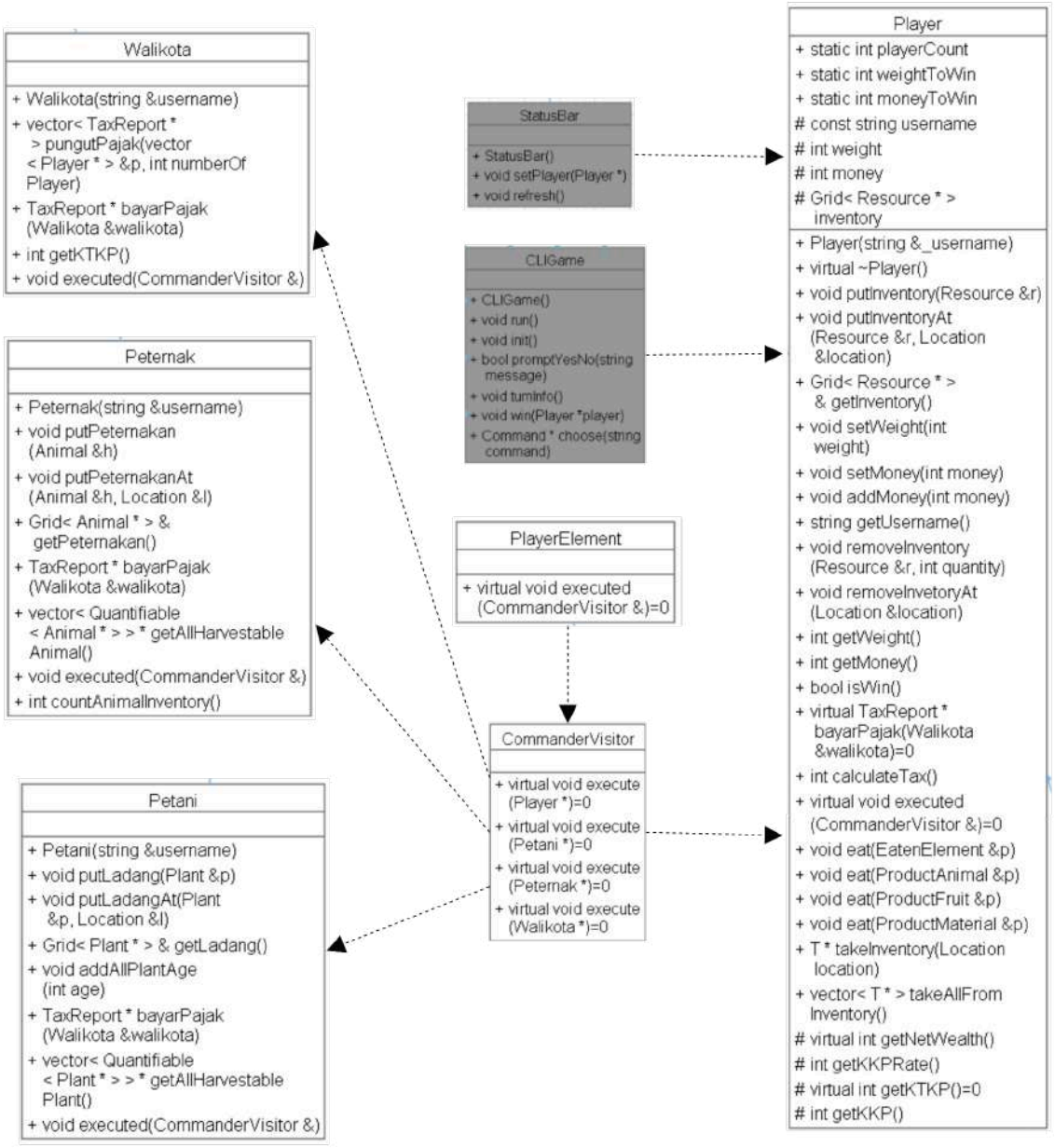
Gambar 1.2.5 Diagram Class Relasi Dependence Kelas Shop



Gambar 1.2.6 Diagram Class Relasi Dependence Kelas Player


Product
int addedWeight
+ Product()=default
+ Product(string code, string name, int addedWeight, int price)
+ virtual ~Product()
+ int getAddedWeight () const
+ void setAddedWeight (int addedWeight)
+ virtual void eaten (EaterVisitor &a)=0
+ virtual void taken (TakerVisitor *t)
ProductAnimal
+ ProductAnimal()=default
+ ProductAnimal(string code, string name, int addedWeight, int price)
+ void eaten(EaterVisitor &a)
+ void taken(TakerVisitor *t)
ProductMaterial
+ ProductMaterial() =default
+ ProductMaterial(string code, string name, int addedWeight, int price)
+ void eaten(EaterVisitor &a)
+ void taken(TakerVisitor *t)
ProductFruit
+ ProductFruit()=default
+ ProductFruit(string code, string name, int addedWeight, int price)
+ void eaten(EaterVisitor &a)
+ void taken(TakerVisitor *t)
Taker< Product >
+ Taker * take(Resource *)
+ void take(Product *)
+ Product * get()
+ void populate(Product *t)
+ void take(ProductMaterial *t)
+ void take(ProductFruit *t)
+ void take(ProductAnimal *t)
+ virtual void take(Animal *)
+ virtual void take(Plant *)
+ virtual void take(Product *)
+ virtual void take(Product Animal *)
+ virtual void take(Product Fruit *)
+ virtual void take(Product Material *)
+ virtual void take(Building *)
Taker< T >
+ Taker * take(Resource *)
+ void take(T *)
+ T * get()
+ virtual void take(Animal *)
+ virtual void take(Plant *)
+ virtual void take(Product *)
+ virtual void take(Product Animal *)
+ virtual void take(Product Fruit *)
+ virtual void take(Product Material *)
+ virtual void take(Building *)
Animal
int weight
+ Animal()=default
+ Animal(string code, string name, int price, int weightToHarvest, vector < Product * > drops)
+ void setWeight(int weight)
+ int getWeight() const
+ void eat(EatenElement &)
+ bool isHarvestable () const
+ bool operator==(const Animal &a)
+ void taken(TakerVisitor *)
Plant
int age
int durationToHarvest
+ Plant()=default
+ Plant(string code, string name, int price, int durationToHarvest, vector< Product * > drops)
+ void addAge(int age)
+ void setAge(int age)
+ int getAge()
+ bool isHarvestable () const
+ virtual ~Plant()
+ bool operator==(const Plant &a)
+ void taken(TakerVisitor *t)
Building
+ Building(string code, string name, int price, const vector< Quantifiable < ProductMaterial * > > &recipe)
+ void printBuildingInfo () const
+ void build(Player &p)
+ void addMaterial(Product Material &material, int quantity)
+ void taken(TakerVisitor *t)
+ vector< Quantifiable < ProductMaterial * > > getRecipe() const
Resource
string code
string name
int price
+ Resource()=default
+ virtual ~Resource()
+ Resource(string code, string name, int price)
+ string getCode() const
+ string getName() const
+ int getPrice() const
+ bool operator==(Resource &)
+ virtual void taken (TakerVisitor *)=0

Gambar 1.2.7 Diagram Class Relasi Dependence Kelas Taker<Product> dan Taker Generic


Walikota
+ Walikota(string &username)
+ vector< TaxReport * > pungutPajak(vector < Player * > &p, int numberOf Player)
+ TaxReport * bayarPajak (Walikota &walikota)
+ int getKTKP()
+ void executed(CommanderVisitor &)
StatusBar
+ StatusBar()
+ void setPlayer(Player *)
+ void refresh()
Player
+ static int playerCount
+ static int weightToWin
+ static int moneyToWin
# const string username
# int weight
# int money
# Grid< Resource * > inventory
+ Player(string &_username)
+ virtual ~Player()
+ void putInventory(Resource &r)
+ void putInventoryAt (Resource &r, Location &location)
+ Grid< Resource * > & getInventory()
+ void setWeight(int weight)
+ void setMoney(int money)
+ void addMoney(int money)
+ string getUsername()
+ void removeInventory (Resource &r, int quantity)
+ void removeInvetoryAt (Location &location)
+ int getWeight()
+ int getMoney()
+ bool isWin()
+ virtual TaxReport * bayarPajak(Walikota &walikota)=0
+ int calculateTax()
+ virtual void executed (CommanderVisitor &)=0
+ void eat(EatenElement &p)
+ void eat(ProductAnimal &p)
+ void eat(ProductFruit &p)
+ void eat(ProductMaterial &p)
+ T * takeInventory(Location location)
+ vector< T * > takeAllFrom Inventory()
# virtual int getNetWealth()
# int getKKPRate()
# virtual int getKTKP()=0
# int getKKP()
CLIGame
+ CLIGame()
+ void run()
+ void init()
+ bool promptYesNo(string message)
+ void tuminfo()
+ void win(Player *player)
+ Command * choose(string command)
Peternak
+ Peternak(string &username)
+ void putPeternakan (Animal &h)
+ void putPeternakanAt (Animal &h, Location &l)
+ Grid< Animal * > & getPeternakan()
+ TaxReport * bayarPajak (Walikota &walikota)
+ vector< Quantifiable < Animal * > > * getAllHarvestable Animal()
+ void executed(CommanderVisitor &)
+ int countAnimalInventory()
PlayerElement
+ virtual void executed (CommanderVisitor &)=0
CommanderVisitor
+ virtual void execute (Player *)=0
+ virtual void execute (Petani *)=0
+ virtual void execute (Peternak *)=0
+ virtual void execute (Walikota *)=0
Petani
+ Petani(string &username)
+ void putLadang(Plant &p)
+ void putLadangAt(Plant &p, Location &l)
+ Grid< Plant * > & getLadang()
+ void addAllPlantAge (int age)
+ TaxReport * bayarPajak (Walikota &walikota)
+ vector< Quantifiable < Plant * > > * getAllHarvestable Plant()
+ void executed(CommanderVisitor &)

Gambar 1.2.8 Diagram Class Relasi Dependence Kelas CommanderVisitor, PlayerElement, CLIGame, dan StatusBar.

2. Penerapan Konsep OOP

2.1. Inheritance & Polymorphism

Alasan umum digunakannya konsep inheritance adalah untuk menghindari duplikasi kode, memudahkan pemeliharaan dengan hanya perlu memperbarui di kelas turunan, meningkatkan readibility code, dan mendukung pengembangan tambahan pada kelas turunan tanpa mengubah kelas lainnya secara langsung.

2.1.1. Kelas Resource - Kelas Building, Creature, Product

Kelas Creature, Building, dan Product diturunkan dari kelas Resource karena setiap kelas tersebut memiliki sharing information, yang diwakili oleh kelas Resource. Resource adalah kelas induk yang menyediakan sharing informasi berupa code, name, dan price. Sedangkan pada kelas Building memiliki tambahan atribut berupa recipe, begitu juga pada kelas Product dengan atribut tambahan addedWeight, dan kelas Creature memiliki atribut tambahan drops. Atribut tambahan ini digunakan karena setiap kelas turunan memiliki *responsibility* yang berbeda.

Konsep Polymorphism juga digunakan untuk mendefinisikan method taken yang memiliki perilaku berbeda tiap kelas, seperti pada Building yang hanya mengizinkan dapat di take oleh role tertentu.

```
   class Resource
{
protected:
    string code;
    string name;
    int price;
   ...
   };
```

```
class Building : public Resource
{
private:
    vector<Quantifiable<ProductMaterial*>> recipe;
public:
    void taken(TakerVisitor* t);

};
```

```
class Creature: public Resource {
    private:
        vector<Product*> drops;
    public:
        Creature& operator<<(Product&);
        friend ostream& operator<<(ostream&, const Creature&);
        ...
};
```

```
class Product : public Resource, public EatenElement
{
    protected:
        int addedWeight;
        virtual void taken(TakerVisitor* t);
...
};
```

2.1.2. Kelas EatenElement - Kelas Product

Kelas Product merupakan turunan dari kelas EatenElement yang merupakan abstract class yang mendefinisikan perilaku saat suatu elemen dimakan. Dalam konteks inheritance ini, penggunaan inheritance digunakan sebagai alternatif solusi untuk menghindari penggunaan dynamic casting, yang dapat menjadi kurang efisien dan sulit di-*maintain* dalam beberapa kasus.

Dalam kasus ini, digunakan *Visitor Pattern* untuk menangani situasi saat hewan ingin memakan produk tetapi tidak mengetahui jenis produk yang akan dimakan. Dengan menerapkan *Visitor Pattern*, kelas EatenElement memiliki method virtual yang disebut eaten, yang akan diimplementasikan oleh kelas turunannya seperti kelas Product. Pada class EaterVisitor, terdapat beberapa metode eat yang masing-masing menerima objek dengan jenis yang berbeda (seperti ProductAnimal, ProductFruit, dan ProductMaterial). Metode-metode ini dideklarasikan sebagai virtual pure yang digunakan untuk mengunjungi elemen yang dimakan dan melakukan tindakan yang sesuai berdasarkan jenis produk tersebut, tanpa perlu menggunakan dynamic casting yang kompleks dan rawan kesalahan.

```
   class EatenElement
{
    public:
        virtual void eaten(EaterVisitor &v) = 0;
};
```

```
   class Product : public Resource, public EatenElement
{
    protected:
        int addedWeight;
...
}
```

```
    class EaterVisitor
{
    public:
        void eat(EatenElement);
        virtual void eat(ProductAnimal&) = 0;
        virtual void eat(ProductFruit&) = 0;
        virtual void eat(ProductMaterial&) = 0;
};
```

2.1.3. Kelas Product - Kelas ProductAnimal, ProductFruit, ProductMaterial

Kelas ProductAnimal, ProductFruit, dan ProductMaterial merupakan turunan dari kelas Product, yang mungkin memiliki atribut dan metode yang sama atau serupa di antara ketiganya. Penggunaan inheritance dalam konteks ini mengarah pada untuk memisahkan responsibility dan memenuhi kebutuhan dari kelas yang menggunakan objek tersebut. Misalnya Building ingin menggunakan recipe untuk membangun sebuah bangunan dimana bahannya adalah berasal dari ProductMaterial saja alias sebagai alternatif type handling.

```
class Product : public Resource, public EatenElement
{
    protected:
        int addedWeight;
...
}
```

```
class ProductFruit:public Product{
    public:
        ProductFruit() = default;
```

```
        ProductFruit(string code, string name, int addedWeight, int price);
        void eaten(EaterVisitor &a);
        void taken(TakerVisitor* t);
};
```

```
class ProductAnimal:public Product{
    public:
        ProductAnimal() = default;
        ProductAnimal(string code, string name, int addedWeight, int price);
        void eaten(EaterVisitor &a);
        void taken(TakerVisitor* t);
};
```

```
class ProductMaterial:public Product{
    public:
        ProductMaterial() = default;
        ProductMaterial(string code, string name, int addedWeight, int price);
        void eaten(EaterVisitor &a);
        void taken(TakerVisitor* t);
};
```

2.1.4. Kelas Creature - Kelas Animal, Plant

Kelas Animal dan Plant merupakan turunan dari kelas Creature, yang bertujuan tidak hanya untuk berbagi informasi atribut yang sama, tetapi juga untuk menangani tanggung jawab yang sangat berbeda antara keduanya. Perbedaan dalam responsibilitas ini tercermin dalam atribut tambahan yang dimiliki oleh masing-masing kelas. Kelas Plant memiliki atribut

tambahan age dan durationToHarvest, sementara attribut tambahan pada Kelas Animal adalah weight dan weightToHarvest.

```
class Creature: public Resource {
    private:
        vector<Product*> drops;
...
}
```

```
class Plant:public Creature
{
protected:
    int age;
    int durationToHarvest;
...
}
```

```
class Animal: public Creature, public EaterVisitor
{
private:
    int weightToHarvest;
protected:
    int weight;
...
}
```

2.1.5. Kelas EaterVisitor - Kelas Player, Animal

Kelas Player dan Animal merupakan turunan dari kelas EaterVisitor yang mendefinisikan perilaku memakan sesuatu. Dalam konteks inheritance ini, penggunaan inheritance digunakan sebagai alternatif solusi untuk menghindari penggunaan dynamic casting, yang dapat menjadi kurang efisien dan sulit di-maintain dalam beberapa kasus.

Dalam kasus ini, digunakan Visitor Pattern untuk menangani situasi saat hewan ingin memakan produk tetapi tidak mengetahui jenis produk yang akan dimakan. Dengan menerapkan Visitor Pattern, kelas EatenVisitor memiliki method virtual yang disebut eat, yang akan diimplementasikan oleh kelas turunannya seperti kelas Player. Pada class EaterVisitor, terdapat beberapa metode eat yang masing-masing menerima objek dengan jenis yang berbeda (seperti ProductAnimal, ProductFruit, dan ProductMaterial). Metode-metode ini dideklarasikan sebagai virtual pure yang digunakan untuk mengunjungi elemen yang dimakan dan melakukan tindakan yang sesuai berdasarkan jenis produk tersebut, tanpa perlu menggunakan dynamic casting yang kompleks dan rawan kesalahan.

```
class EaterVisitor
{
    public:
        void eat(EatenElement);
        virtual void eat(ProductAnimal&) = 0;
        virtual void eat(ProductFruit&) = 0;
        virtual void eat(ProductMaterial&) = 0;
};
```

```
class Player : public PlayerElement , public EaterVisitor // visitor pattern
{
protected:
    const string username;
    int weight;
```

```
    int money;
    Grid<Resource *> inventory;

    virtual int getNetWealth();
    int getKKPRate();
    virtual int getKTKP() = 0;
    int getKKP();
public:
    static int playerCount;
    static int weightToWin;
    static int moneyToWin;

    void eat(EatenElement &p);

    void eat(ProductAnimal &p);
    void eat(ProductFruit &p);
    void eat(ProductMaterial &p);
...
};
```

```
class Animal: public Creature, public EaterVisitor
{
private:
    int weightToHarvest;
protected:
    int weight;
...
};
```

2.1.6. Kelas Animal - Kelas Carnivore, Herbivore, Omnivore

Kelas Carnivore, Herbivore, dan Omnivore merupakan turunan dari kelas Animal dalam konteks ini, yang mengaplikasikan prinsip inheritance untuk memisahkan perilaku berdasarkan jenis makanan yang dikonsumsi oleh masing-masing kelas. Dengan inheritance, struktur hierarki kelas mengorganisir hewan-hewan ini berdasarkan kebutuhan makanannya, memastikan kelas-kelas turunan memiliki perilaku makan yang sesuai dengan sifatnya.Sebagai contoh, Carnivore hanya mampu memakan ProductAnimal, Herbivore hanya dapat mengonsumsi ProductFruit, sementara Omnivore memiliki kemampuan untuk mengonsumsi kedua jenis produk tersebut.

```
    class Animal: public Creature, public EaterVisitor
  {
  private:
      int weightToHarvest;
  protected:
      int weight;
  ...
  };
```

```
class Herbivore: public Animal {
    public:
        Herbivore(string code, string name, int price, int weightToHarvest, vector<Product*>
drops);
        void eat(ProductAnimal&);
        void eat(ProductFruit&);
        void eat(ProductMaterial&);
};
```

```
class Carnivore: public Animal {
    public:
        Carnivore(string code, string name, int price, int weightToHarvest, vector<Product*>
drops);
        void eat(ProductAnimal&);
        void eat(ProductFruit&);
        void eat(ProductMaterial&);
};
```

```
class Omnivore: public Animal {
    public:
        Omnivore(string code, string name, int price, int weightToHarvest, vector<Product*>
drops);
        void eat(ProductAnimal&);
        void eat(ProductFruit&);
        void eat(ProductMaterial&);
};
```

2.1.7. Kelas Player - Kelas Petani, Peternak, Walikota

Kelas Petani, Peternak, dan Walikota merupakan turunan dari kelas Player. Dalam konteks ini, tujuan inheritance adalah tujuan umum yaitu karena Petani, Peternak dan Walikota memiliki attribut tambahan yang berbeda sehingga juga merepresentasikan perilaku yang berbeda. Dimana petani memiliki ladang, peternak memiliki peternakan, sementara walikota sama sekali tidak memiliki keduanya. Konsep inheritance juga digunakan pada perhitungan pajak untuk masing masing tipe player yaitu pada perhitungan kekayaan tidak kena pajak dan perhitungan kekayaan neto. Konsep polymorphism juga diterapkan untuk memastikan Command yang diizinkan oleh masing-masing role player tersebut, sehingga tiap kelas hanya diperbolehkan mengakses command tertentu sesuai role yang dimiliki pada method executed.

```
class Player : public PlayerElement , public EaterVisitor // visitor pattern
{
protected:
    const string username;
    int weight;
    int money;
    Grid<Resource *> inventory;

    virtual int getNetWealth();
    int getKKPRate();
    virtual int getKTKP() = 0;
    int getKKP();
public:
    static int playerCount;
    static int weightToWin;
    static int moneyToWin;
    void eat(EatenElement &p);

    void eat(ProductAnimal &p);
    void eat(ProductFruit &p);
    void eat(ProductMaterial &p);
    virtual void executed(CommanderVisitor&) = 0;

...
};
```

```
class Petani : public Player
```

```
{
private:
    Grid<Plant*> ladang;
    int getNetWealth();
    int getKTKP();
    static vector <Petani*> listPetani;
public:
    void executed(CommanderVisitor&);
...
};
```

```
class Peternak : public Player
{
private:
    Grid<Animal*> peternakan;
    int getNetWealth();
    int getKTKP();
public:
    void executed(CommanderVisitor&);
...
}
```

```
class Walikota : public Player
{
private:
public:
    void executed(CommanderVisitor&);
```

```
...
};
```

2.1.8. Kelas CommanderVisitor - Kelas Command - Kelas Bangun, Beli, CetakLadang, CetakPenyimpanan, CetakPeternakan, Help, Jual, KasihMakan, Keluar, Makan, Next, Panen, PungutPajak, Simpan, Status, TambahPemain, Tanam, Ternak

Kelas Bangun, Beli, CetakLadang, CetakPenyimpanan, CetakPeternakan, Help, Jual, KasihMakan, Keluar, Makan, Next, Panen, PungutPajak, Simpan, Status, TambahPemain, Tanam, dan Ternak merupakan turunan dari Kelas Command,dan kelas Command sendiri merupakan turunan dari kelas Commander Visitor. Dalam konteks ini, tujuan konsep inheritance adalah karena semua perilaku antar child command sangat berbeda dan tentunya memiliki method dan atribut yang berbeda pula. Konsep polymophism juga digunakan disini untuk memastikan suatu command melalui method execute dapat diakses dan dijalankan berdasarkan role tertentu sesuai dengan fungsionalitasnya.

```
class Command : public CommanderVisitor {
    protected:
        State& state;
        vector<Location> inputListLocation(const string &line);
    public:
        Command(State&);
        virtual void execute(Player*);
        virtual void execute(Petani*);
        virtual void execute(Peternak*);
        virtual void execute(Walikota*);
```

```
};
```

2.2. Method/Operator Overloading

2.2.1. Method Overloading

Kebanyakan method overloading pada konsep kita akan digunakan pada visitor pattern, seperti pada CommanderVisitor, EaterVisitor, dan TakerVisitor, beserta turunan dari parent class ini. Pattern ini digunakan sebagai alternatif solusi untuk menghindari penggunaan dynamic casting, yang dapat menjadi kurang efisien dan sulit di-maintain dala beberapa kasus dan sebagai handling terhadap kasus jenis produk dan command yang berbeda.

Dalam struktur ini, method-method yang dideklarasikan sebagai virtual pure pada setiap visitor class digunakan untuk mengunjungi elemen yang dimakan atau dieksekusi dan melakukan tindakan yang sesuai berdasarkan jenis produk, command, atau peran tersebut. Sebagai contoh, pada CommanderVisitor, terdapat method `*execute*` yang di-overload untuk Player, Petani, Peternak, dan Walikota. Ini memungkinkan setiap turunan dari CommanderVisitor untuk melakukan tindakan eksekusi yang sesuai tergantung pada jenis pemain yang dihadapi.

Sementara itu, pada TakerVisitor, terdapat method `*take*` yang di-overload untuk berbagai jenis elemen seperti Animal, Plant, Product, dan sebagainya. Hal ini memungkinkan TakerVisitor untuk mengambil tindakan yang tepat terhadap jenis elemen yang sedang ditangani.

Terakhir, pada EaterVisitor, terdapat method `*eat*` yang di-overload untuk berbagai jenis produk seperti ProductAnimal, ProductFruit, dan ProductMaterial. Ini memungkinkan EaterVisitor untuk mengkonsumsi produk-produk tersebut dengan tindakan yang sesuai berdasarkan jenis produk yang ditemui.

Dengan mengimplementasikan pattern ini dan menggunakan method overloading, dapat memisahkan tanggung jawab dan mengelola aksi yang tepat berdasarkan jenis entitas atau peran yang dihadapi, tanpa harus terlibat dalam dynamic casting yang kompleks dan rawan kesalahan.

```
class CommanderVisitor
{
```

```
    public:
        virtual void execute(Player*) = 0;
        virtual void execute(Petani*) = 0;
        virtual void execute(Peternak*) = 0;
        virtual void execute(Walikota*) = 0;      .
};
```

```
class TakerVisitor
{
    public:
        virtual void take(Animal*);
        virtual void take(Plant*);
        virtual void take(Product*);
        virtual void take(ProductAnimal*);
        virtual void take(ProductFruit*);
        virtual void take(ProductMaterial*);
        virtual void take(Building*);
};
```

```
class EaterVisitor
{
    public:
        void eat(EatenElement);
        virtual void eat(ProductAnimal&) = 0;
        virtual void eat(ProductFruit&) = 0;
        virtual void eat(ProductMaterial&) = 0;
};
```

2.2.2. Operator Overloading

Tujuan penggunaan operator overloading adalah untuk memperluas fungsionalitas dan fleksibilitas bawaan dari C++. Dengan meng-overload operator, dapat menentukan perilaku khusus yang terjadi ketika operator tersebut digunakan dengan objek dari kelas yang telah didefinisikan, adapun operator overloading yang kami gunakan diantaranya

- Overloading Operator == pada Animal, Resource dan Plant untuk membandingkan kesamaan antara dua objek kelas tersebut.
- Overloading Operator << pada Animal, Plant, Creature, ResourceFactory, dan Shop untuk mencetak informasi objek ke output stream agar dengan mudah menampilkan informasi yang relevan dari objek ke layar atau file untuk keperluan debugging atau tampilan.
- Overloading Operator [] pada Grid dan Shop untuk akses elemen pada lokasi tertentu dalam grid, seperti nilai pada koordinat yang ditentukan. Sementara itu, pada kelas Shop, operator [] digunakan untuk mengakses elemen pada indeks tertentu, seperti item yang tersedia di toko pada indeks tertentu.
- Overloading Operator >> pada Location dan Plant untuk operasi input dari input stream. Misalnya, pada kelas Location, operator ini digunakan untuk membaca data lokasi dari input stream, sementara pada kelas Plant, operator ini digunakan untuk membaca data tanaman dari input stream.
- Overloading Operator --, ++, -=, += pada Quantifiable untuk melakukan operasi aritmatika pada objek Quantifiable dengan cara yang lebih intuitif, seperti mengurangi atau menambahkan nilai pada objek tersebut.
- Overlaoding Operator += pada Grid untuk memasukkan element nya secara otomatis di elemen paling kiri atas yang kosong.
- Overloading Operator = pada Creature untuk transger nilai dari satu objek Creature ke objek lainnya.

```
bool operator==(const Animal &a);
friend ostream &operator<<(ostream &os, const Animal &a);
T operator[](Location l);
Grid<T>& operator+=(const T &val);
friend istream& operator>>(istream& is, Location& location);
```

```
friend ostream& operator<<(ostream& os, Location& location);
friend ostream &operator<<(ostream &os, const Location &location);
friend ostream& operator<<(ostream &os, const IcosiHexaString& str);
bool operator==(const Plant &a);
friend ostream &operator<<(ostream &os, const Plant &a);
bool operator==(const Plant &a);
friend ostream &operator<<(ostream &os, const Plant &a);
friend istream &operator>>(istream &is, Plant &a);
friend ostream& operator<<(ostream &os, const IcosiHexaString& str);
bool operator==(Quantifiable<T> other);
Quantifiable<T> operator--(int);
Quantifiable<T> operator++(int);
void operator-=(int qty);
void operator+=(int qty);
void operator+=(Quantifiable<T> other);
Creature& operator=(const Creature &);
friend ostream& operator<<(ostream&, const Creature&);
Creature& operator<<(Product&);
bool operator==(Resource&);
friend ostream& operator<<(ostream&,Shop&);
friend ostream& operator<<(ostream& os, const ResourceFactory& factory);
Quantifiable<Resource*>& operator[](int idx);
```

2.3. Abstract Class & Virtual Function

2.3.1. Abstract Class

Berdasarkan rancangan diagram UML yang telah kami sediakan, kami telah mengelompokkan beberapa *entity class* untuk meng-*inherit* suatu superclass tertentu. Diantara beberapa superclass tersebut kami jadikan sebagai abstract class agar dapat memaksimalisasi prinsip DRY (Don't Repeat Yourself). Suatu class dapat dinyatakan sebagai superclass apabila ia memiliki beberapa atribut dan method yang sama dari lebih dari satu class.

Kelas Resource merupakan parent paling tinggi terhadap item - item yang dapat disimpan oleh pemain / toko. Atribut yang dimiliki oleh kelas ini adalah `string code,string name,` dan `int price`.

resource.h

```
class Resource
{
protected:
    string code;
    string name;
    int price;

public:
    Resource() = default;
    virtual ~Resource();
    Resource(string code, string name, int price);

    /**
```

```
     * @brief Mengembalikan code dari resource
     *
     * @return string code dari resource
     */
    string getCode() const;
    /**
     * @brief Mengembalikan nama dari resource
     *
     * @return string nama dari resource
     */
    string getName() const;

    /**
     * @brief Mengembalikan harga dari resource
     *
     * @return int harga dari resource
     */
    int getPrice() const;

    bool operator==(Resource&);

    friend ostream &operator<<(ostream &os, const Resource &res);

    // visitor pattern
    virtual void taken(TakerVisitor*) = 0;
};
```

Kelas Creature merupakan superclass dari Kelas Hewan dan Kelas Tumbuhan. Dengan kata lain, kelas ini merupakan kelas makhluk hidup.

creature.h

```
class Creature: public Resource {
    private:
        vector<Product*> drops;
    public:
        Creature() = default;
        Creature(string code, string name, int price, vector<Product*> drops);
        Creature(Creature&);
        ~Creature();
        Creature& operator=(const Creature &);
        Creature& operator<<(Product&);
        virtual bool isHarvestable() const = 0;
        vector<Product*>& harvest();
        friend ostream& operator<<(ostream&, const Creature&);
};
```

Kelas Product merupakan superclass dari Kelas ProductMaterial, Kelas ProductFood, Kelas ProductAnimal.

product.h

```
class Product : public Resource, public EatenElement
{
    protected:
```

```
        int addedWeight;

    public:
        Product() = default;
        Product(string code, string name, int addedWeight, int price);
        virtual ~Product();
        int getAddedWeight() const;
        void setAddedWeight(int addedWeight);

        // Visitor pattern
        virtual void eaten(EaterVisitor &a) = 0;
        virtual void taken(TakerVisitor* t);
};
```

Kelas Player merupakan superclass dari Kelas Petani, Kelas Peternak, Kelas Walikota.

player.h

```
class Player : public PlayerElement , public EaterVisitor
{
protected:
    const string username;
    int weight;
    int money;
    Grid<Resource *> inventory;
```

```
    virtual int getNetWealth();
    int getKKPRate();
    virtual int getKTKP() = 0;
    int getKKP();
public:
    static int playerCount;
    static int weightToWin;
    static int moneyToWin;
    Player(string &_username);
    virtual ~Player();

    void putInventory(Resource &r);
    void putInventoryAt(Resource &r, Location &location);
    Grid<Resource*>& getInventory();

    void setWeight(int weight);
    void setMoney(int money);
    void addMoney(int money);

    string getUsername();
    void removeInventory(Resource &r, int quantity);

    void removeInvetoryAt(Location &location);
    int getWeight();
    int getMoney();

    bool isWin();
```

```
    virtual TaxReport* bayarPajak(Walikota &walikota) = 0;

    int calculateTax();


    // Visitor pattern
    virtual void executed(CommanderVisitor&) = 0;

    void eat(EatenElement &p);

    void eat(ProductAnimal &p);
    void eat(ProductFruit &p);
    void eat(ProductMaterial &p);

    template<class T>
    T* takeInventory(Location location);

    template<class T>
    vector<T*> takeAllFromInventory();
};
```

Dalam menggambar petak yang ada dalam permainan tersebut, diperlukan beberapa kelas abstract pendukung

griddrawer.h

```
#include "grid.h"

template <class T>
class GridDrawer {
    protected:
        Grid<T> &grid;
    public:
        GridDrawer(Grid<T> &grid);
        virtual void draw() = 0;
};
```

Terdapat juga beberapa Class yang merupakan implementasi dari Design Pattern yang kami gunakan dalam projek kami. Keterangan selanjutnya akan dibahas pada Bonus yang Dikerjakan

playervisitorpattern.h

```
class CommanderVisitor
{
    public:
        virtual void execute(Player*) = 0; // Command overrides this. If derived classes of
Command overrides this, that means the argument type doesn't matter.
        virtual void execute(Petani*) = 0;        // Derived classes override this.
        virtual void execute(Peternak*) = 0;      // Derived classes override this.
        virtual void execute(Walikota*) = 0;      // Derived classes override this.
```

```
};

class PlayerElement
{
    public:
        virtual void executed(CommanderVisitor&) = 0;
};
```

productvisitorpattern.h

```
class EaterVisitor
{
    public:
        void eat(EatenElement);
        virtual void eat(ProductAnimal&) = 0;
        virtual void eat(ProductFruit&) = 0;
        virtual void eat(ProductMaterial&) = 0;
};

class EatenElement
{
    public:
        virtual void eaten(EaterVisitor &v) = 0;
};
```

2.3.2. Virtual Function

Setelah mendesain rancangan class, tentu saat implementasinya diperlukan teknik-teknik khusus yang sudah disediakan C++ untuk memaksimalisasi konsep OOP, salah satunya dengan menggunakan keyword *virtual*. Implementasi yang menggunakan virtual keyword digunakan pada beberapa fungsionalitas seperti prinsip *dynamic binding* dan *visitor pattern* pada design pattern.

Keperluan untuk mengeksekusi command tertentu, tanpa menggunakan dynamic casting, digunakanlah visitor pattern.

playervisitorpattern.h

```
// Visitor pattern to avoid dynamic_cast. Needed to decide the action of each player but the
only information we know is that the type is Player (base class)
// Derived classes of Command will override only one of the following options:
// 1. void execute(Player*);
// 2. void execute(Petani*), void execute(Peternak*), void execute(Walikota*);
class CommanderVisitor
{
   public:
       virtual void execute(Player*) = 0; // Command overrides this. If derived classes of
Command overrides this, that means the argument type doesn't matter.
       virtual void execute(Petani*) = 0;        // Derived classes override this.
       virtual void execute(Peternak*) = 0;      // Derived classes override this.
       virtual void execute(Walikota*) = 0;      // Derived classes override this.
```

```
};

class PlayerElement
{
   public:
       virtual void executed(CommanderVisitor&) = 0;
};
```

Grid drawer sebagai abstract base class sehingga dapat diinherit oleh grid drawer untuk CLI dan grid drawer untuk GUI.

griddrawer.h

```
template <class T>
class GridDrawer {
   protected:
       Grid<T> &grid;
   public:
       GridDrawer(Grid<T> &grid);
       virtual void draw() = 0;
};
```

Command.h yang dipakai sebagai base class terhadap command-command yang bisa dilakukan oleh pengguna.

command.h

```
class Command : public CommanderVisitor { // visitor pattern
   protected:
       MainWindow& window;
       State& state;
   public:
       Command(State&, MainWindow&);

       string formatName(string);

       // visitor pattern
       virtual void execute(Player*);
       virtual void execute(Petani*);
       virtual void execute(Peternak*);
       virtual void execute(Walikota*);
};
```

Method notify pada application GUI sehingga library qt mengoverride untuk *handling error.*

application.h

```
class Application : public QApplication {
private:
   QMainWindow* window;
public:
   Application(int& argc, char** argv);
   void setMainWindow(QMainWindow* window);
   virtual bool notify(QObject *receiver, QEvent *e);
};
```

Keperluan berbagai method menggunakan virtual keyword agar dapat digunakan dynamic binding

player.h

```
class Player : public PlayerElement , public EaterVisitor // visitor pattern
{
protected:
   const string username;
   int weight;
   int money;
   Grid<Resource *> inventory;

   virtual int getNetWealth();
   int getKKPRate();
   virtual int getKTKP() = 0;
```

```
    int getKKP();
public:
    static int playerCount;
    static int weightToWin;
    static int moneyToWin;
    Player(string &_username);
    virtual ~Player();

    void putInventory(Resource &r);
    void putInventoryAt(Resource &r, Location &location);
    Grid<Resource*>& getInventory();

    void setWeight(int weight);
    void setMoney(int money);
    void addMoney(int money);

    string getUsername();
    void removeInventory(Resource &r, int quantity);

    void removeInvetoryAt(Location &location);
    int getWeight();
    int getMoney();

    bool isWin();

    virtual TaxReport* bayarPajak(Walikota &walikota) = 0;
```

```
    int calculateTax();


    // Visitor pattern
    virtual void executed(CommanderVisitor&) = 0;

    void eat(EatenElement &p);

    void eat(ProductAnimal &p);
    void eat(ProductFruit &p);
    void eat(ProductMaterial &p);

    template<class T>
    T* takeInventory(Location location);

    template<class T>
    vector<T*> takeAllFromInventory();
};
```

Digunakan virtual keyword sebagai base class Product untuk penggunaan visitor pattern saat suatu product ingin dimakan.

product.h

```
class Product : public Resource, public EatenElement
{
   protected:
        int addedWeight;

   public:
        Product() = default;
        Product(string code, string name, int addedWeight, int price);
        virtual ~Product();
        int getAddedWeight() const;
        void setAddedWeight(int addedWeight);

        // Visitor pattern
        virtual void eaten(EaterVisitor &a) = 0;
        virtual void taken(TakerVisitor* t);
};
```

productvisitorpattern.h

```
class EaterVisitor
{
```

```
    public:
        void eat(EatenElement);
        virtual void eat(ProductAnimal&) = 0;
        virtual void eat(ProductFruit&) = 0;
        virtual void eat(ProductMaterial&) = 0;
};

class EatenElement
{
    public:
        virtual void eaten(EaterVisitor &v) = 0;
};
```

Creature digunakan sebagai abstract base class yang memerlukan childrennya mengoverride method isHavestable()

creature.h

```
class Creature: public Resource {
```

```
    private:
        vector<Product*> drops;
    public:
        Creature() = default;
        Creature(string code, string name, int price, vector<Product*> drops);
        Creature(Creature&);
        ~Creature();
        Creature& operator=(const Creature &);
        Creature& operator<<(Product&);
        virtual bool isHarvestable() const = 0;
        vector<Product*>& harvest();
        friend ostream& operator<<(ostream&, const Creature&);
};
```

resourcevisitorpattern.h

```
class TakerVisitor
{
    public:
        // Animal* take(Resource*);
```

```
        // Default will throw exception
        virtual void take(Animal*);
        virtual void take(Plant*);
        virtual void take(Product*);
        virtual void take(ProductAnimal*);
        virtual void take(ProductFruit*);
        virtual void take(ProductMaterial*);
        virtual void take(Building*);
};
```

2.4. Template & Generic Classes

Alasan penggunaan template dan generic, diantaranya fleksibilitas untuk membuat kode dengan berbagai jenis data tanpa perlu menulis ulang implementasi untuk setiap jenis data tersebut, reusabilitas untuk membuat kode yang dapat digunakan kembali dalam berbagai konteks dengan hanya mengganti tipe data yang diberikan, dan abstraksi agar lebih lebih mudah dimengertikarena mengurangi duplikasi dan kompleksitas yang tidak perlu.

Penggunaan template dan generic classes digunakan pada kelas Grid dan Location untuk menyimpan dan memroses elemen-elemen dalam bentuk matriks berbasis vektor sebagai inventory yang menyimpan berbagai jenis elemen dengan berbagai macam tipe kelas. Kelas Quantifiable juga menggunakan template untuk menyimpan nilai dari semua jenis kelas yang dapat diubah quantity atau jumlah dari objek kelas tersebut. Kelas Taker juga menggunakan template untuk menerapkan pola Visitor dalam pengambilan objek dari berbagai jenis. Pola Visitor memungkinkan operasi yang berbeda tergantung pada tipe objek yang dikunjungi. Penggunaan template dalam kelas Taker memungkinkan kelas tersebut untuk berinteraksi dengan berbagai jenis objek (seperti Animal, Product, Plant.

```
template <class T>
class Grid {
    private:
        vector<vector<T>> element;
        vector<vector<bool>> isFilled;
        int countFilled;
        int countNotFilled;
...
};
```

```
class Player : public PlayerElement , public EaterVisitor // visitor pattern
{
    template<class T>
    T* takeInventory(Location location);

    template<class T>
    vector<T*> takeAllFromInventory();
};
```

```
template <class T>
class Quantifiable
{
private:
    T value;
    int quantity;
...
};
```

```
template <class T>
class Taker : public TakerVisitor
{
private:
    T* ptr;
...
};
template <>
class Taker<Product> : public TakerVisitor
{
private:
    Product* ptr;
...
};
```

2.5. Exception

2.5.1. Kelas Animal

Penggunaan exception pada kelas Animal adalah untuk mengatasi kasus pemanggilan method *eat* saat suatu animal makan suatu product yang bukan sesuai dengan jenisnya, misalnya Herbivore memakan ProductFruit. Kegunaannya Exception pada kasus ini adalah untuk menghindari eksekusi kode yang tidak sesuai dengan kebutuhan atau constraint dari kelas yang telah didefinisikan serta menampilkan pesan kesalahan.

```
class CannotEatException : public exception
{
```

```
private:
    const string message;
public:
    CannotEatException(const Animal &animal, const Product &product);
    const char* what() const throw();
};
```

2.5.2. Kelas Building

Penggunaan exception dalam kelas Building memiliki beberapa tujuan yaitu untuk menangani situasi di mana pemanggilan method bangun dilakukan oleh peran (role) yang bukan walikota. Selain itu, exception juga berguna untuk menangani kasus di mana slot inventaris tidak mencukupi untuk menambahkan bangunan baru, material yang dimiliki oleh walikota tidak mencukupi untuk membangun suatu bangunan, atau uang yang tersedia tidak mencukupi untuk membeli bangunan. Exception juga digunakan untuk menangani kebutuhan pencetakan bahan baku yang diperlukan namun tidak terpenuhi. Dengan menggunakan exception, eksekusi kode yang tidak sesuai dengan kebutuhan atau batasan yang telah ditentukan dalam kelas dapat dihindari, dan pesan kesalahan yang jelas dapat ditampilkan kepada pengguna.

```
class RoleWaliKotaException: public exception{
    const char* what() const throw();

};
class SlotNotAvailableException: public exception{
    const char* what() const throw() ;
};
```

```
class NotEnoughMaterialException: public exception{
    const char* what() const throw();
};

class NotEnoughMoneyException: public exception{
    const char* what() const throw() ;
};

class MissingResourcesException: public exception{
    private:
        map<string, int> missingResources;
    public:
        MissingResourcesException(map<string, int> missingResources);
        map<string, int> &getMissingResources();
        const char* what() const throw();
};
```

2.5.3. Kelas Beli

Penggunaan exception pada kelas beli bertujuan untuk menangani kasus pembelian suatu barang jika mengakses nomor yang tidak sesuai dengan command pada layar, kuantitas yang diinput oleh user melebihi kuantitas barang yang ada di toko, uang untuk membeli suatu barang tidak mencukupi, dan ketika penyimpanan dalam inventory tidak mencukupi. Penggunaan exception ini untuk mencegah constraint case yang telah disebutkan sebelumnya dan menampilkan pesan kesalahan ke layar sebagai langkah preventif terjadinya error.

```
class BeliOutOfRange: public exception{
```

```
    const char* what() const throw() {
            return "Barang tidak ditemukan";
     }
};

class BarangTidakCukup: public exception{
    const char* what() const throw() {
            return "Kuantitas tidak mencukupi";
     }
};

class UangTidakCukup: public exception{
    const char* what() const throw() {
            return "Uang tidak mencukupi";
     }
};

class PenyimpananTidakCukup: public exception{
    const char* what() const throw() {
            return "Slot penyimpanan tidak mencukupi";
     }
};
```

2.5.4. Kelas Command

Penggunaan exception pada kelas Command umumnya digunakan untuk mengatasi kasus ketika suatu command diakses oleh seseorang yang tidak sesuai dengan rolenya, ketika user memasukkan command yagn tidak sesuai dan tidak ada pada command yang tersedia dan exception child. Penggunaan exception ini untuk mencegah constraint case yang telah disebutkan sebelumnya dan menampilkan pesan kesalahan ke layar sebagai langkah preventif terjadinya error.

```
class CommandNotAllowedException: public exception {
    private:
        string message;
    public:
        CommandNotAllowedException(string command);
        const char* what() const throw();
};

class CommandNotExistException: public exception {
    private:
        string message;
    public:
        CommandNotExistException(string command);
        const char* what() const throw();
};

class InvalidInputLocationListException: public exception {
    public:
```

```
        InvalidInputLocationListException();
        const char* what() const throw();
};

class BuildingNotFoundException: public exception {
    public:
        BuildingNotFoundException();
        const char* what() const throw();
};

class AnimalNotFoundException: public exception {
    private:
        string message;
    public:
        AnimalNotFoundException();
        const char* what() const throw();
};

class EmptyInventoryException: public exception {
    private:
        string message;
    public:
        EmptyInventoryException();
        const char* what() const throw();
};

class NoFoodException: public exception {
```

```
    private:
        string message;
    public:
        NoFoodException();
        const char* what() const throw();
};

class NoAnimalInInventoryException: public exception {
    private:
        string message;
    public:
        NoAnimalInInventoryException();
        const char* what() const throw();
};

class PeternakanPenuhException: public exception {
    private:
        string message;
    public:
        PeternakanPenuhException();
        const char* what() const throw();
};
```

2.5.5. Kelas Location

Penggunaan exception pada kelas Location untuk error handling terkait penggunaan format baris dan kolom pada pemetaan suatu barang. Exception ini digunakan ketika pengguna memasukkan nilai baris dan kolom yang tidak sesuai dengan format yang ditentukan, seperti format A01. Handling ini dirancang untuk mengatasi input yang tidak sesuai dengan ketentuan yang telah ditetapkan dalam kelas Location saat ingin mengakses inventory dan resources dan menampilkan pesan kesalahan ke layar.

```
class LocationException : public exception {
    public:
        LocationException();
        const char* what() const throw();
};
```

2.5.6. Kelas ResourceFactory

Penggunaan exception pada kelas ResourceFactory untuk error handling terkait load config txt yang terdapat pada suatu folder berdasarkan input user, adapun kasusnya adalah ketika sistem tidak dapat menemukan folder yang diminta atau jika file konfigurasi yang dibutuhkan tidak ada dalam folder yang dimaksud. Exception ini akan menampilkan pesan kesalahan ke layar.

```
FileNotFoundException::FileNotFoundException(const string& path) : message("File not found: " +
path) {}

const char* FileNotFoundException::what() const throw(){
    return message.c_str();
}
```

```
FolderNotFoundException::FolderNotFoundException(const string& path) : message("Folder not
found: " + path) {}

const char* FolderNotFoundException::what() const throw(){
    return message.c_str();
};
```

2.5.7. Kelas ResourceVisitorPattern

Penggunaan exception pada kelas ini difokuskan pada error handling yang terjadi saat suatu role mencoba mengambil item yang tidak sesuai. Kelas ini diimplementasikan dengan pattern visitor sehingga tidak diperlukan dynamic cast. Dengan menggunakan exception, sistem dapat menampilkan pesan kesalahan secara langsung ke layar.

```
class NotTakableException : public exception{
    private:
        const string message;
    public:
        NotTakableException(const Resource& r);
        const char* what() const throw();
};
```

2.5.8. Kelas Shop

Penggunaan exception pada kelas shop digunakan untuk error handling pada beberapa kasus, seperti ketika item yang diinginkan tidak ditemukan dalam shop, ketika stock barang kosong sehingga tidak dapat diproses, ketika jumlah uang yang dimiliki pengguna tidak mencukupi untuk melakukan transaksi, atau ketika role pengguna tidak sesuai dengan yang dibutuhkan untuk akses ke fitur tertentu dalam shop.

Dengan menggunakan exception, sistem dapat menampilkan pesan kesalahan secara langsung ke layar pengguna, memberikan informasi yang jelas dan spesifik mengenai masalah yang terjadi.

```
class ItemShopNotFoundException : public exception {
public:
    const char *what() const throw() override {
        return "Item is not available in shop";
    }
};

class ItemShopNotEqualException : public exception {
public:
    const char *what() const throw() override {
        return "Both Item is not equal";
    }
};

class ItemShopEmptyException : public exception {
public:
    const char *what() const throw() override {
        return "Stock is empty";
```

```
    }
};

class UangTidakCukupException : public exception {
public:
    const char *what() const throw() override {
        return "Insufficient balance";
    }
};

class PetaniException : public exception {
public:
    const char *what() const throw() override {
        return "Farmer Role can't buy this item";
    }
};

class PeternakException : public exception {
public:
    const char *what() const throw() override {
        return "Rancher Role can't buy this item";
    }
};

class WalikotaException : public exception {
public:
    const char *what() const throw() override {
```

```
        return "Mayor can't buy this item";
    }
};
```

2.5.9. Kelas State

Penggunaan exception pada kelas state digunakan untuk error handling pada kasus ketika suatu state diakses oleh role yang tidak sesuai. Dengan menggunakan exception, sistem dapat menampilkan pesan kesalahan secara langsung ke layar pengguna, memberikan informasi yang jelas dan spesifik mengenai masalah yang terjadi

```
class InvalidPlayerTypeException : public exception
{
private:
    const string message;
public:
    InvalidPlayerTypeException(const string &type);
    const char* what() const throw();
};
```

2.6. C++ Standard Template Library

2.6.1. Vector

Beberapa kelas mengimplementasikan STL vector, seperti pada kelas Quantifiable, Grid, Player, Creature, Shop, dan State beserta turunannya. STL vector digunakan karena menyediakan array dinamis yang dapat mengatur penyimpanannya secara otomatis yang memungkinkan penambahan, penghapusan, dan akses elemen dengan fleksibilitas tinggi melalui fungsi-fungsi yang tersedia dengan mudah dan efisien, memberikan fleksibilitas yang diperlukan dalam pengelolaan data.

Penggunaan vector di kelas Creature dan turunannya diimplementasikan dengan memanfaatkan fitur generic berupa Product. Fitur ini digunakan untuk menyimpan array of product yang dihasilkan oleh Creature secara dinamis. Sementara pada kelas Building, vector digunakan untuk menyimpan resep yang diperlukan untuk membangun suatu bangunan.Kelas Grid menggunakan vector dengan prinsip matriks untuk melakukan pemetaan pada inventory, ladang, atau peternakan terkait barang-barang yang disimpan di dalamnya. Ini juga mencakup fungsi isFilled untuk memeriksa apakah suatu elemen di dalamnya kosong atau sudah terisi.Di kelas Petani, vector digunakan untuk menyimpan informasi mengenai list para petani, dan ini hanya sebagian kecil dari penggunaan vector di dalam program. Keputusan untuk menggunakan vector disebabkan oleh sifatnya yang dinamis, memungkinkan nilai-nilainya untuk bertambah seiring dengan perkembangan game state yang sedang berjalan.

```
class Creature: public Resource {
    private:
        vector<Product*> drops;
   ...
};
```

```
class Building : public Resource
{
private:
    vector<Quantifiable<ProductMaterial*>>
recipe;
...
};
```

```
template <class T>
class Grid {
    private:
        vector<vector<T>> element;
        vector<vector<bool>> isFilled;
...
};
```

```
class Petani : public Player
{
private:
    Grid<Plant*> ladang;
    int getNetWealth();
    int getKTKP();
    static vector <Petani*> listPetani;
...
};
```

```
class Player : public PlayerElement , public
EaterVisitor
{
...
    template<class T>
    vector<T*> takeAllFromInventory();
...
};
```

```
class Walikota : public Player
{
private:
public:
    Walikota(string &username);
    vector<TaxReport *>
pungutPajak(vector<Player*> &p, int
numberOfPlayer);
...
};
```

```
class ResourceFactory : map<string,
function<Resource*()>>{
    private:
        map<string, vector<Product*>> dropsMap;
...
};
```

```
class Shop {
    private:
        vector<Quantifiable<Resource*>> stock;
...
};
```

```
class State {
    private:
        vector<Player*> playerList;
...
};
```

2.6.2. Pair

STL pair dalam C++ merupakan sebuah template class yang digunakan untuk mengelompokkan dua objek dengan tipe yang berbeda menjadi satu unit. Alasan penggunaannya adalah terletak pada kelebihan utama dari STL ini, yaitu kemudahan penggunaannya dan fleksibilitas dalam memilih tipe data untuk kedua elemen. Dengan menggunakan STL pair, dapat mengakses dan memanipulasi kedua nilai ini sebagai satu unit dengan mudah. STL ini digunakan pada class Beli yaitu untuk mengecek apakah siatu item pada resource sudah dibeli, validity checking, welcome message,dan printStocks dengan memasangkannya bersama tipe boolean. STL ini juga digunakan sebagai base class dari location yang menyimpan secara berpasangan baris dan kolom letak suatu item dalam inventory, ladang, maupun peternakan.

```
    class Beli: public Command {
        private:
            Shop &shop;
        public:
            bool isBuyable(pair<Quantifiable<Resource*>,bool> pair);
                void validityChecking(vector<pair<Quantifiable<Resource*>,bool>> stok, Player*
  p,int idxItem,int quantity);

            pair<int,int> welcomeMessage(vector<pair<Quantifiable<Resource*>,bool>>,Player*);
            void printStocks(vector<pair<Quantifiable<Resource*>,bool>> stock);
};
```

```
    class Location: pair<int, int> {
     public:
  ...
  }
```

2.6.3. Map

STL map dalam C++ adalah sebuah container associative yang menghubungkan antara key (key) dengan nilai (value) yang terkait. Salah satu keunggulan utama dari STL map adalah efisiensi dalam operasi pencarian, penambahan, dan penghapusan data berdasarkan key, karena map menggunakan struktur data tree seperti red-black tree. STL map juga memberikan fleksibilitas dalam penggunaan tipe data sebagai key dan nilai seperti dictionary pada bahasa lain. STL ini digunakan pada kelas CLIGame untuk mengakses suatu command dengan string berupa input user. Digunakan juga pada kelas State untuk mengakses bangunan terkait berdasarkan unique codenya, dan digunakan pada state untuk pembacaan string dari file txt pada setiap linenya.

```
class CLIGame {
    private:
        void initializeCommand();
        State state;
        ResourceFactory factory;
        map<string, Command*> commands;
...
}
```

```
class State {
    public:
        map<string, Building*>& getRecipeMap();
...
};
```

```
class FileReader {
    private:
        map<string, string> lineData;
```

```
...
}
```

2.6.4. Regex

STL regex dalam C++ adalah sebuah library untuk melakukan pattern matching sebuah string. Hal ini akan berguna saat melakukan validasi input lokasi pada grid untuk lokasi lebih dari satu dengan format tertentu. Misalnya untuk input A01, B05, E02. Maka kita dapat melakukan validasi input ini dengan regex. Kita juga bisa memisah input tersebut dengan regex untuk menghasilkan larik lokasi grid.

```
vector<Location> Command::inputListLocation(const string &line) {
    regex validation(R"(([A-Z]+\d+)(\s*,\s*([A-Z]\d+))*)");
    smatch match;
    if (!regex_search(line, match, validation) || match.suffix().length() > 0) {
        throw InvalidInputLocationListException();
    }

    vector<Location> result;
    regex pattern(R"(\w+\d+)");

    auto words_begin = sregex_iterator(line.begin(), line.end(), pattern);
    auto words_end = sregex_iterator();
    ...
};
```

2.6.5. Set

STL set dalam C++ adalah sebuah container untuk menyimpan elemen secara unik. Hal ini akan berguna saat ingin menampilkan tanaman atau hewan apa saja yang ada pada ladang atau peternakan dengan menuliskan code dan namanya. Dengan menggunakan set, elemennya tidak akan ada yang duplikat.

```
void CetakLadang::printInfo(Grid<Plant *> &ladang) {
    set<pair<string, string>> codesAndNames;
    ...
}
```

```
void CetakPeternakan::printInfo(Grid<Animal *> &peternakan){
    set<pair<string, string>> codesAndNames;
    ...
}
```

2.6.6. Algorithm

STL algorithm dalam C++ adalah sebuah library untuk mempermudah melakukan algoritma-algoritma tertentu dengan hanya melakukan pemanggilan fungsi dari library tersebut. Algorithm ini akan berguna saat ingin melakukan sorting urutan player, membuat string input command menjadi huruf kapital, dan lain-lain.

```
void State::addPlayer(string type, string name){
    ...
    sort(playerList.begin(), playerList.end(), [](Player* a, Player* b) {
        return a->getUsername() < b->getUsername();
    });
```

```
    ...
}
```

```
void CLIGame::run() {
    string command;
    while (true) {
        cout << ">> ";
        cin >> command;
        try {
            transform(command.begin(), command.end(), command.begin(), ::toupper); // uppercase
       ...
}
```

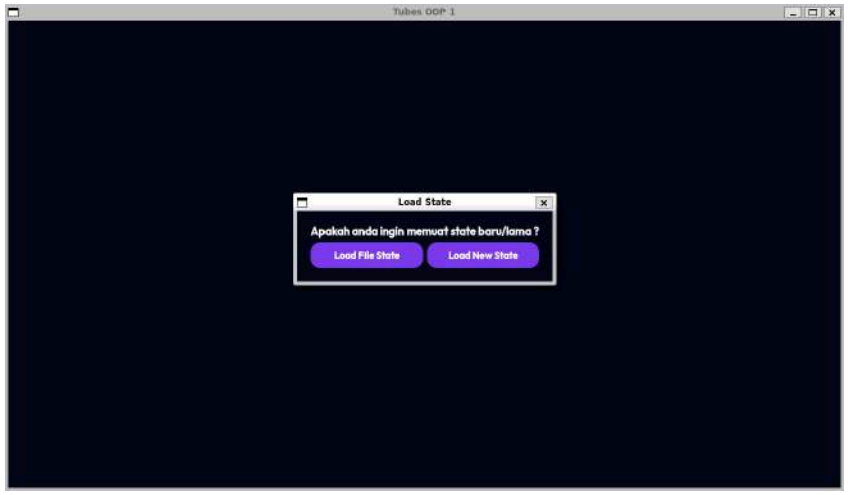
3. Bonus Yang dikerjakan

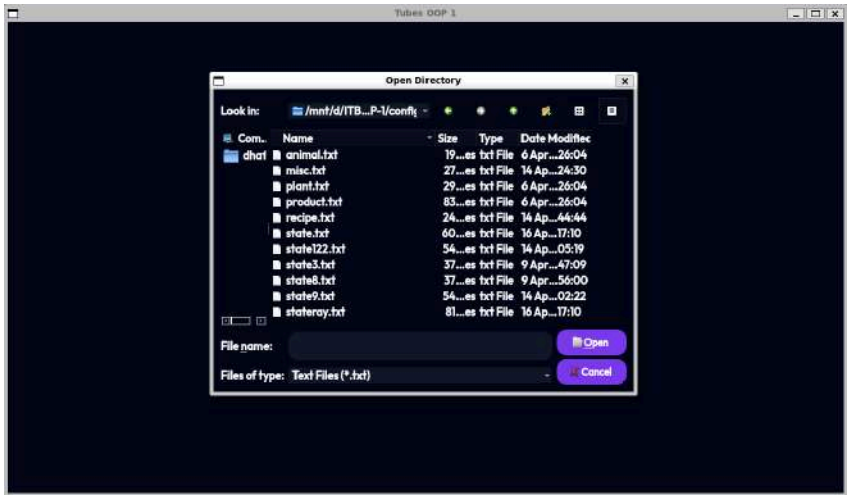
3.1. Bonus yang diusulkan oleh spek

3.1.1. Antarmuka Pengguna Grafis (APG)/*Graphical User Interface*

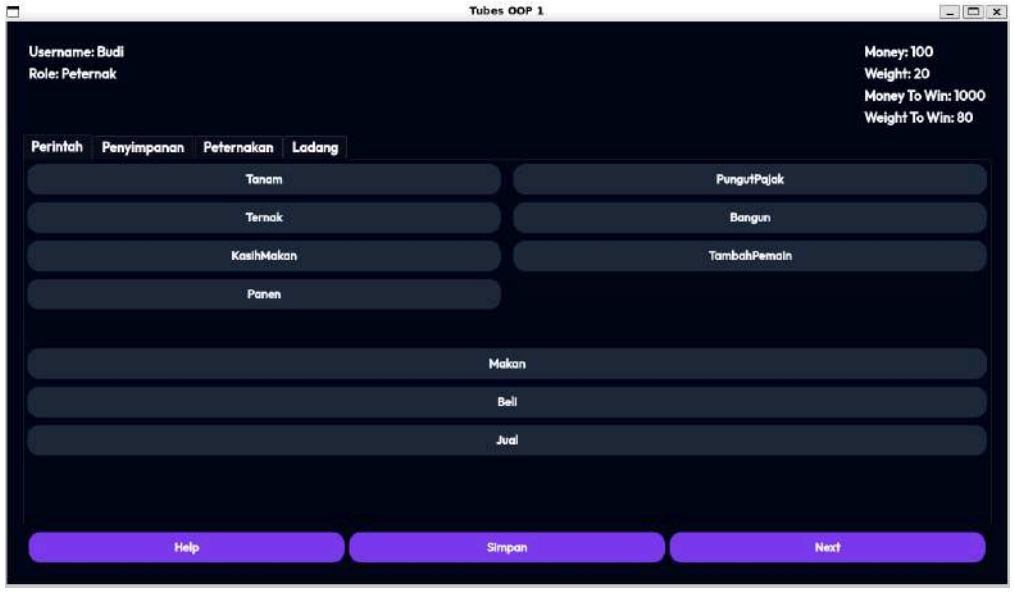
GUI pada game dibuat menggunakan kakas qt5. Class pada gui dibagi menjadi 2 jenis yaitu Command dan Widgets. Setiap jenis command merupakan child class dari class Command. Setiap jenis Widget merupakan child class dari QWidget yang merupakan class built in dari qt5.

3.1.1.1. Muat

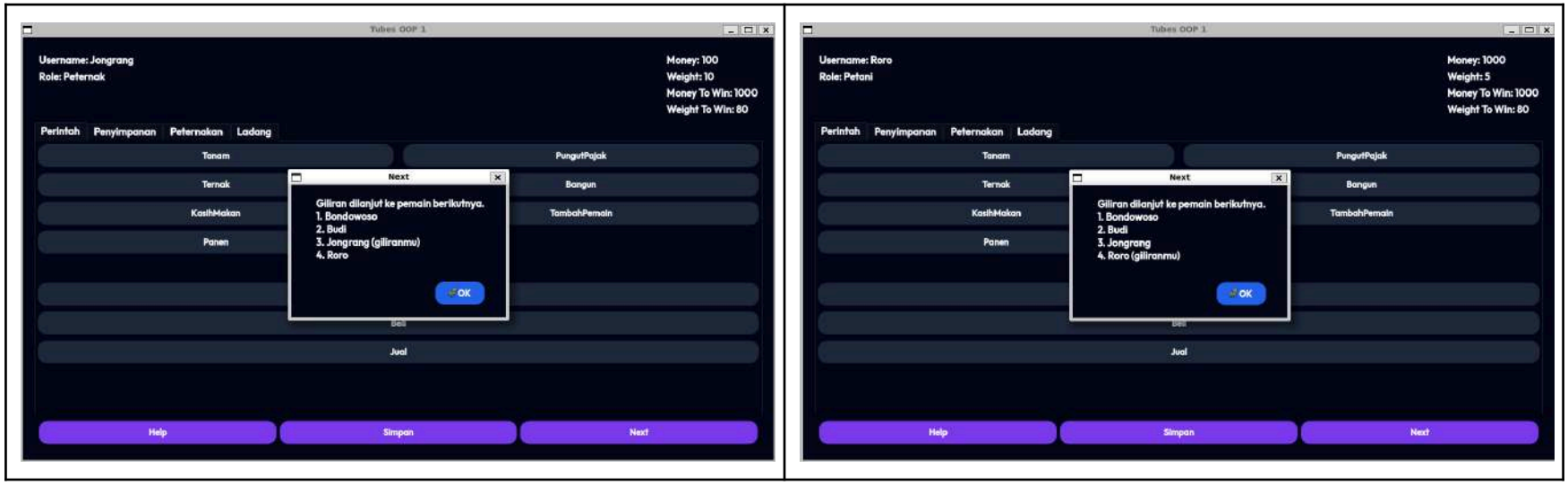

Tubes OOP 1
Load State
Apakah anda ingin memuat state baru/lama ?
Load File State
Load New State


Tubes OOP 1
Open Directory
Look in:
/mnt/d/ITB...P-1/config
Com..
dhat
Name
Size
Type
Date Modifiec
animal.txt 19...es txt File 6 Apr...26:04
misc.txt 27...es txt File 14 Ap...24:30
plant.txt 29...es txt File 6 Apr...26:04
product.txt 83...es txt File 6 Apr...26:04
recipe.txt 24...es txt File 14 Ap...44:44
state.txt 60...es txt File 16 Ap...17:10
state122.txt 54...es txt File 14 Ap...05:19
state3.txt 37...es txt File 9 Apr...47:09
state8.txt 37...es txt File 9 Apr...56:00
state9.txt 54...es txt File 14 Ap...02:22
stateray.txt 81...es txt File 16 Ap...17:10
File name:
Open
Files of type:
Text Files (*.txt)
Cancel

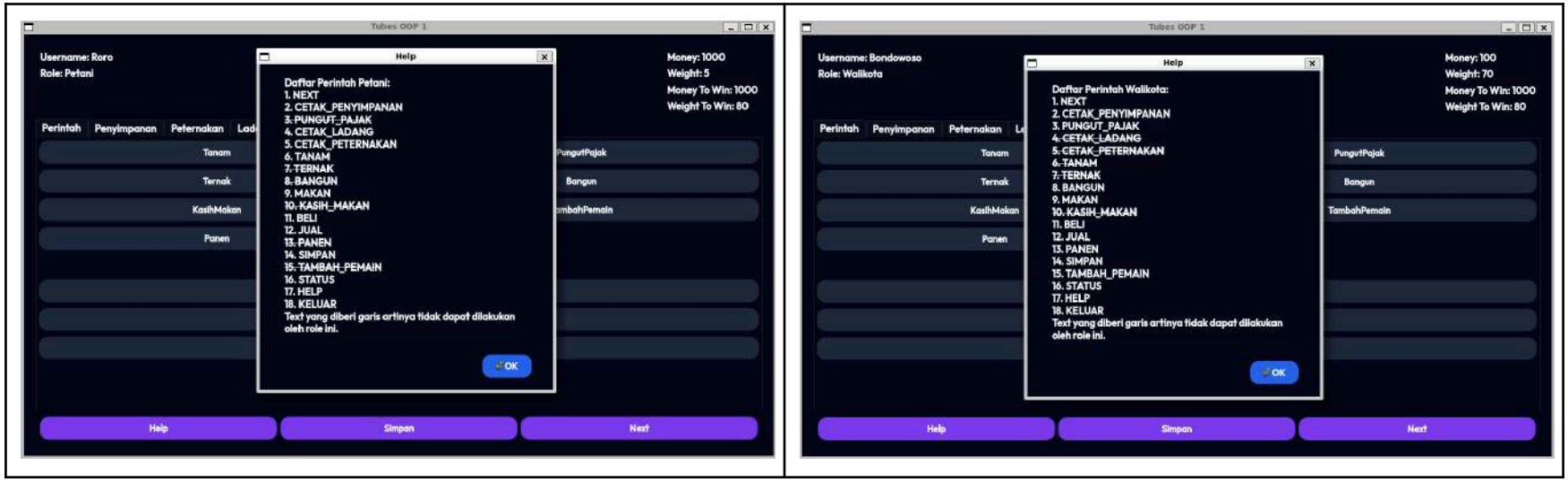
3.1.1.2. Main Menu



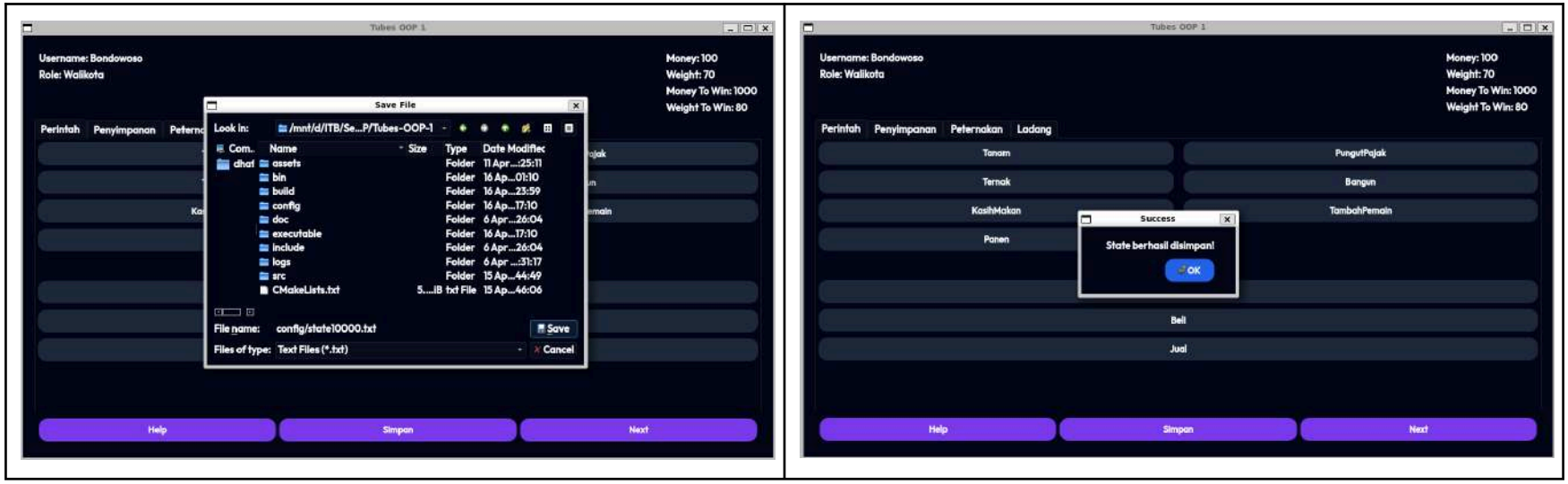
3.1.1.3. Next


Tubes OOP 1
Username: Jongrang
Role: Peternak
Money: 100
Weight: 10
Money To Win: 1000
Weight To Win: 80
Perintah
Penyimpanan
Peternakan
Ladang
Tanam
PungutPajak
Ternak
Bangun
KasihMakan
TambahPemain
Panen
Beli
Jual
Help
Simpan
Next
Next
Giliran dilanjut ke pemain berikutnya.
1. Bondowoso
2. Budi
3. Jongrang (giliranmu)
4. Roro
OK
Tubes OOP 1
Username: Roro
Role: Petani
Money: 1000
Weight: 5
Money To Win: 1000
Weight To Win: 80
Perintah
Penyimpanan
Peternakan
Ladang
Tanam
PungutPajak
Ternak
Bangun
KasihMakan
TambahPemain
Panen
Beli
Jual
Help
Simpan
Next
Next
Giliran dilanjut ke pemain berikutnya.
1. Bondowoso
2. Budi
3. Jongrang
4. Roro (giliranmu)
OK

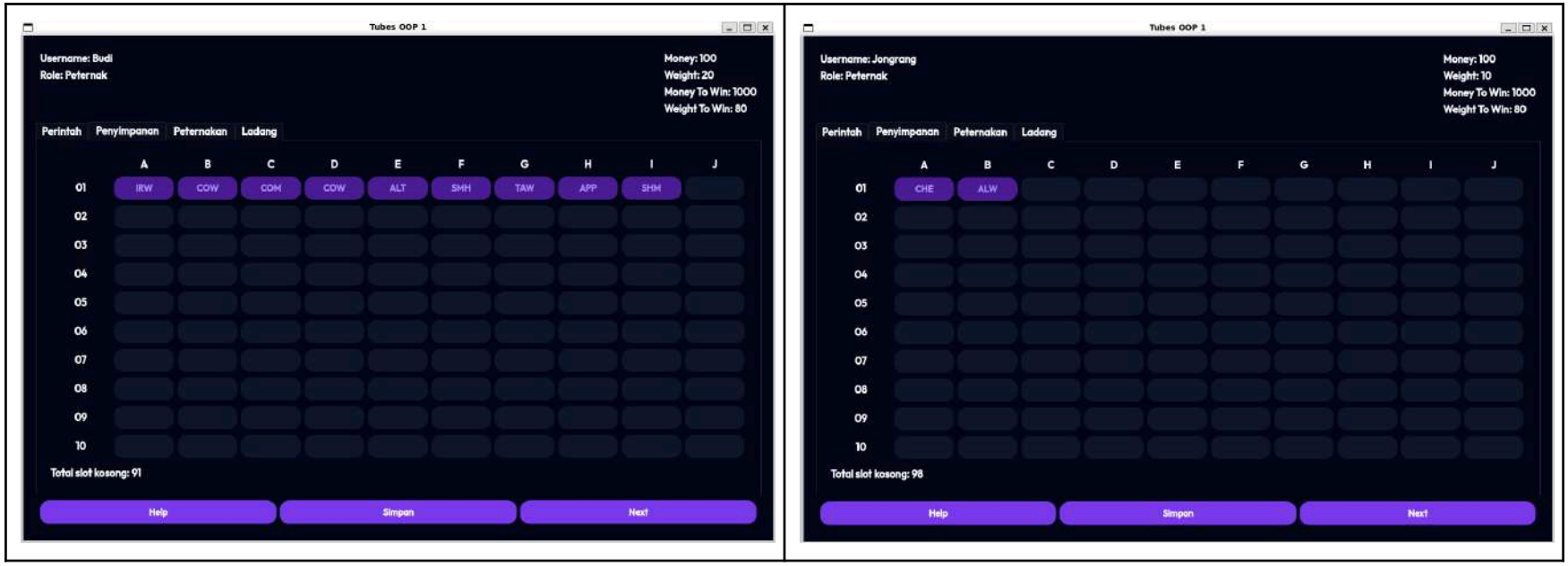
3.1.1.4. Help


Tubes OOP 1
Username: Roro
Role: Petani
Money: 1000
Weight: 5
Money To Win: 1000
Weight To Win: 80
Perintah
Penyimpanan
Peternakan
Tanam
Ternak
KasihMakan
Panen
PungutPajak
Bangun
TambahPemain
Help
Simpan
Next
Help
Daftar Perintah Petani:
1. NEXT
2. CETAK_PENYIMPANAN
3. PUNGUT_PAJAK
4. CETAK_LADANG
5. CETAK_PETERNAKAN
6. TANAM
7. TERNAK
8. BANGUN
9. MAKAN
10. KASIH_MAKAN
11. BELI
12. JUAL
13. PANEN
14. SIMPAN
15. TAMBAH_PEMAIN
16. STATUS
17. HELP
18. KELUAR
Text yang diberi garis artinya tidak dapat dilakukan oleh role ini.
OK
Tubes OOP 1
Username: Bondowoso
Role: Walikota
Money: 100
Weight: 70
Money To Win: 1000
Weight To Win: 80
Help
Daftar Perintah Walikota:
1. NEXT
2. CETAK_PENYIMPANAN
3. PUNGUT_PAJAK
4. CETAK_LADANG
5. CETAK_PETERNAKAN
6. TANAM
7. TERNAK
8. BANGUN
9. MAKAN
10. KASIH_MAKAN
11. BELI
12. JUAL
13. PANEN
14. SIMPAN
15. TAMBAH_PEMAIN
16. STATUS
17. HELP
18. KELUAR
Text yang diberi garis artinya tidak dapat dilakukan oleh role ini.
OK

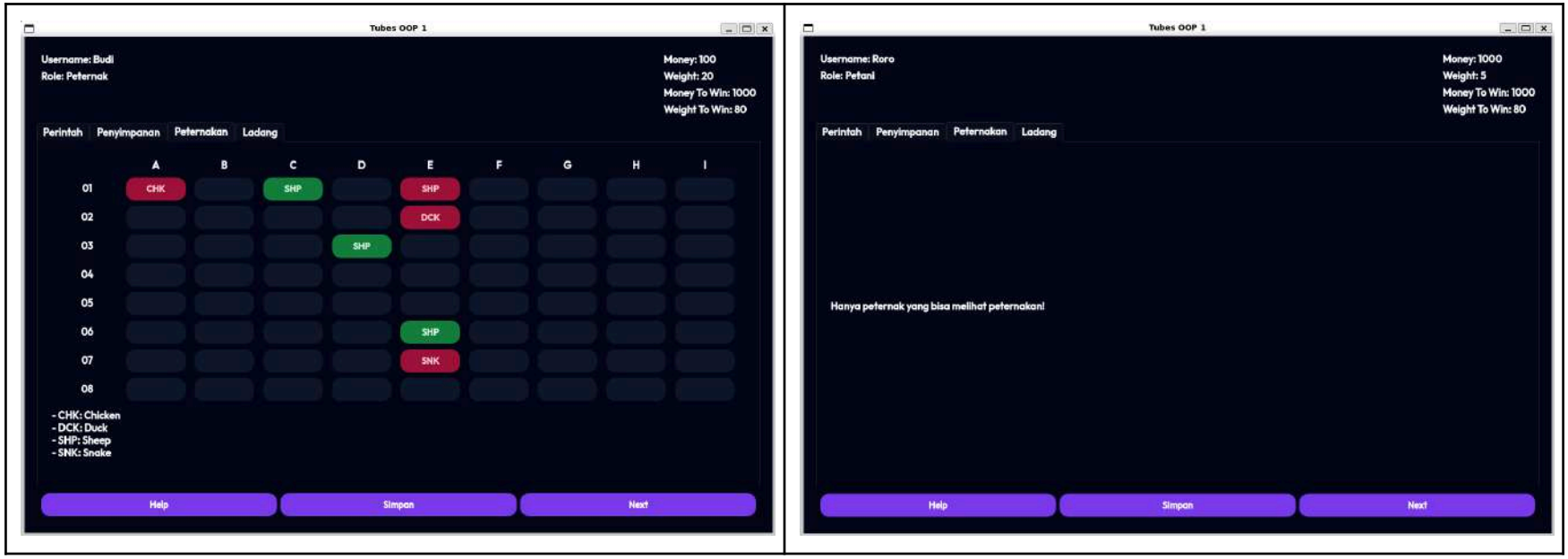
3.1.1.5. Simpan


Tubes OOP 1
Username: Bondowoso
Role: Walikota
Money: 100
Weight: 70
Money To Win: 1000
Weight To Win: 80
Perintah
Penyimpanan
Peternakan
Ladang
Tanam
Ternak
KasihMakan
Panen
PungutPajak
Bangun
TambahPemain
Beli
Jual
Help
Simpan
Next
Save File
Look in:
/mnt/d/ITB/Se...P/Tubes-OOP-1
Name
Size
Type
Date Modifiec
assets
bin
build
config
doc
executable
include
logs
src
CMakeLists.txt
File name:
config/state10000.txt
Files of type:
Text Files (*.txt)
Save
Cancel
Success
State berhasil disimpan!
OK

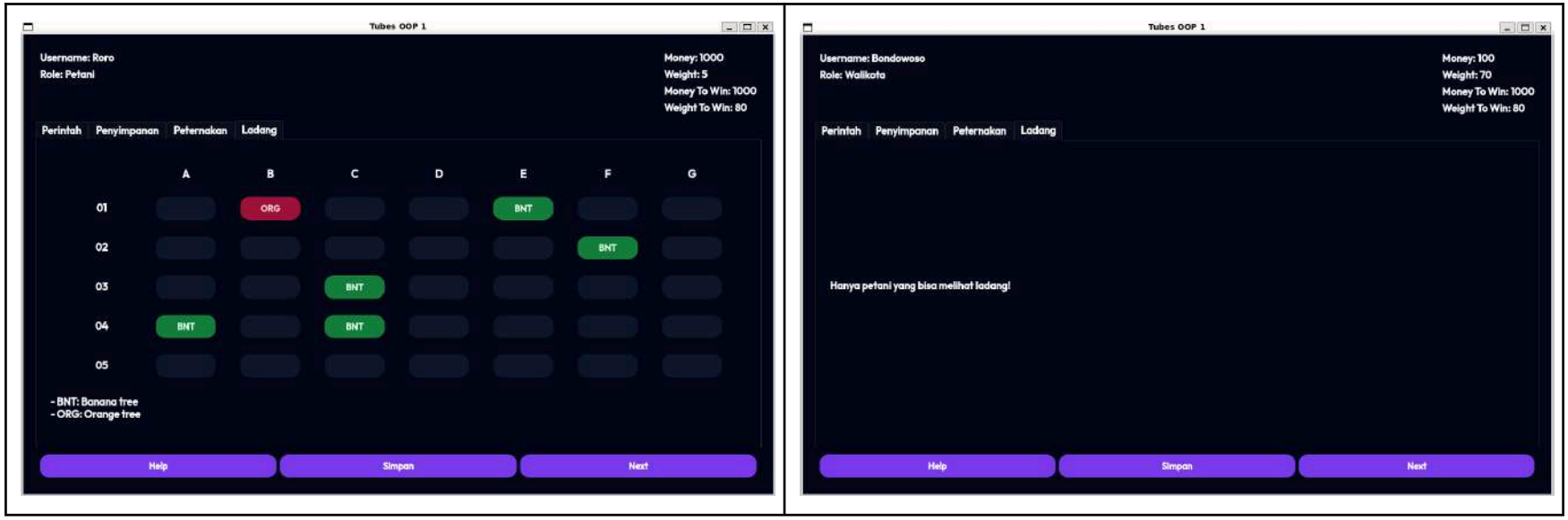
3.1.1.6. Cetak Penyimpanan



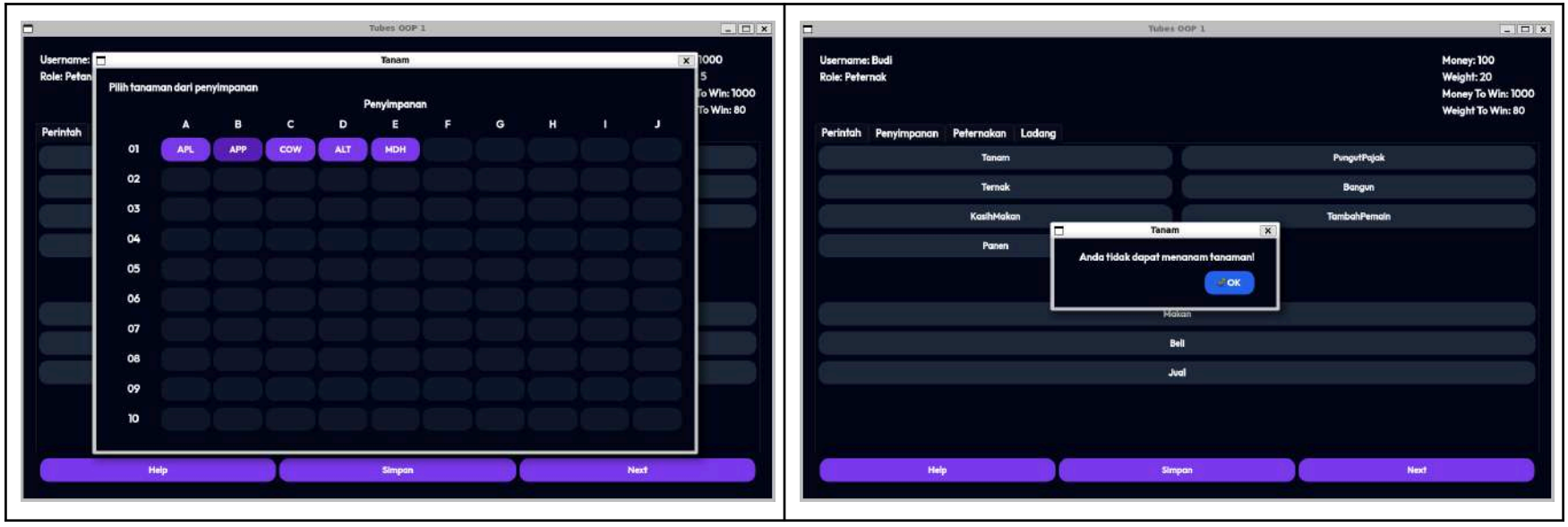
3.1.1.7. Cetak Peternakan

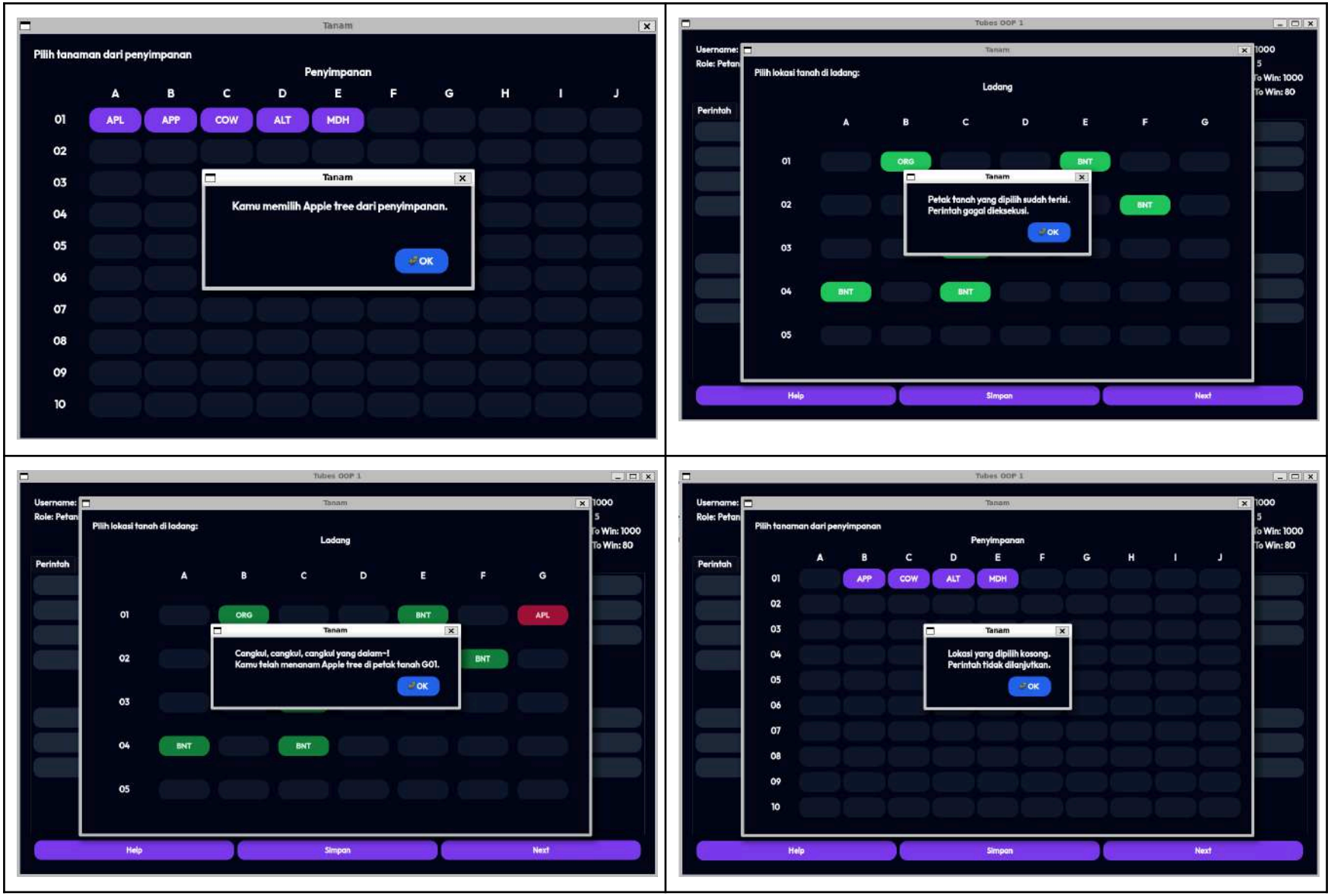


3.1.1.8. Cetak Ladang

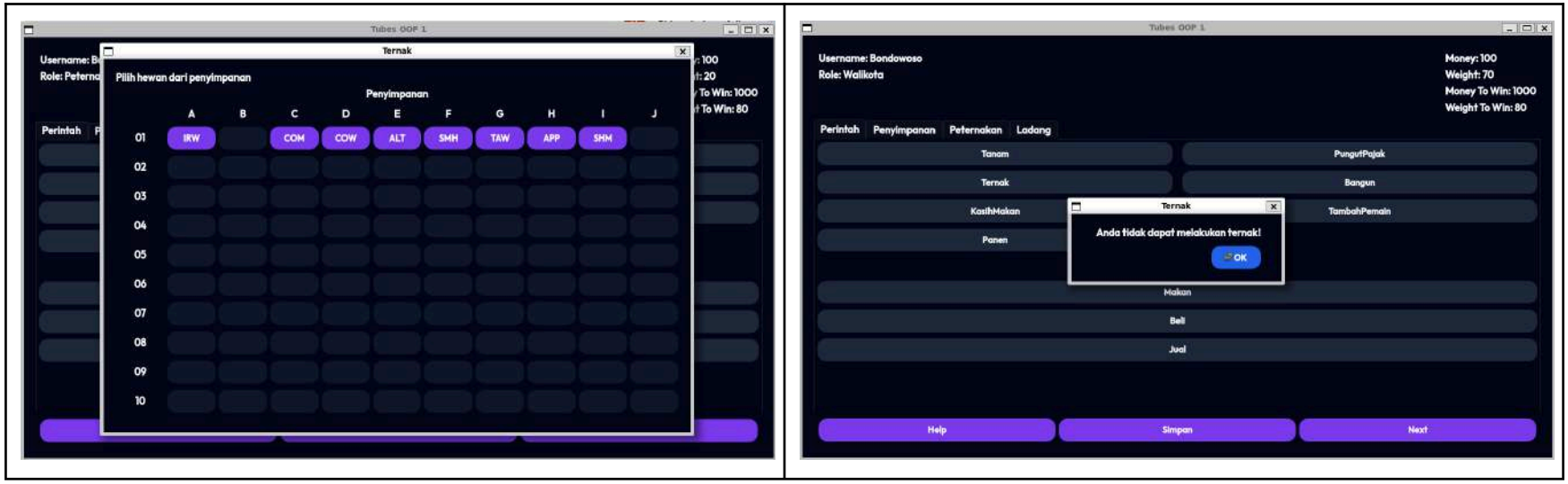


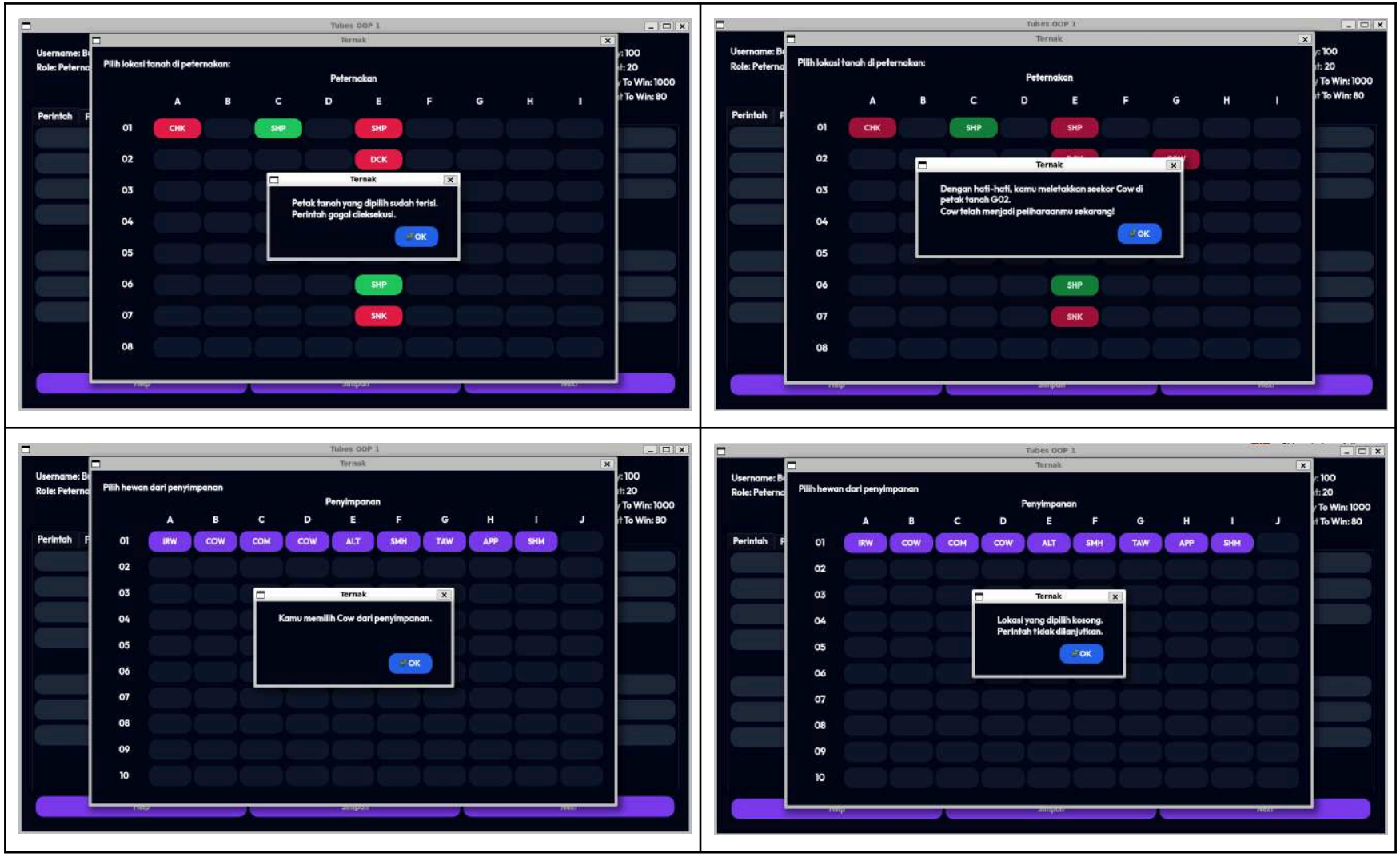
3.1.1.9. Tanam


Tubes OOP 1
Tanam
Pilih tanaman dari penyimpanan
Penyimpanan
A B C D E F G H I J
01 APL APP COW ALT MDH
02 03 04 05 06 07 08 09 10
Perintah
Help
Simpan
Next
Tubes OOP 1
Username: Budi
Role: Peternak
Money: 100
Weight: 20
Money To Win: 1000
Weight To Win: 80
Perintah
Penyimpanan
Peternakan
Ladang
Tanam
Ternak
KasihMakan
Panen
PungutPajak
Bangun
TambahPemain
Tanam
Anda tidak dapat menanam tanaman!
OK
Beli
Jual
Help
Simpan
Next

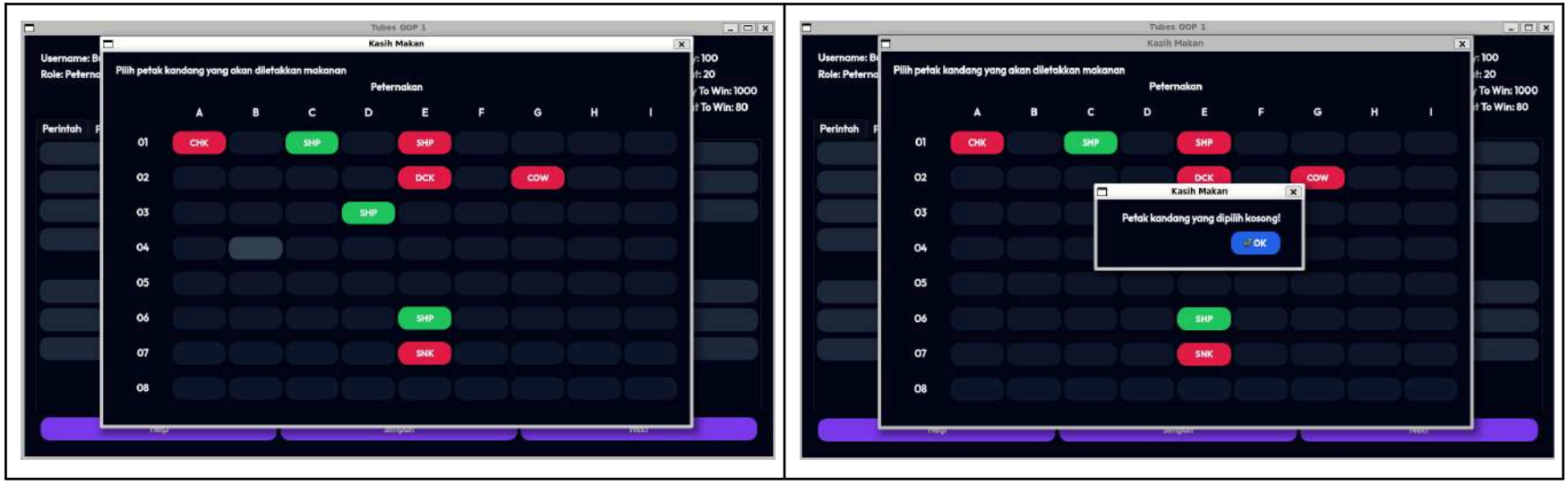

Tanam
Pilih tanaman dari penyimpanan
Penyimpanan
A B C D E F G H I J
01 APL APP COW ALT MDH
02
03
04
05
06
07
08
09
10
Tanam
Kamu memilih Apple tree dari penyimpanan.
OK
Tubes OOP 1
Username:
Role: Petan
Tanam
Pilih lokasi tanah di ladang:
Ladang
Perintah
A B C D E F G
01 ORG BNT
02 BNT
03
04 BNT BNT
05
Tanam
Petak tanah yang dipilih sudah terisi.
Perintah gagal dieksekusi.
OK
Help
Simpan
Next
1000
5
To Win: 1000
To Win: 80
Tubes OOP 1
Username:
Role: Petan
Tanam
Pilih lokasi tanah di ladang:
Ladang
Perintah
A B C D E F G
01 ORG BNT APL
02 BNT
03
04 BNT BNT
05
Tanam
Cangkul, cangkul, cangkul yang dalam~!
Kamu telah menanam Apple tree di petak tanah G01.
OK
Help
Simpan
Next
1000
5
To Win: 1000
To Win: 80
Tubes OOP 1
Username:
Role: Petan
Tanam
Pilih tanaman dari penyimpanan
Penyimpanan
Perintah
A B C D E F G H I J
01 APP COW ALT MDH
02
03
04
05
06
07
08
09
10
Tanam
Lokasi yang dipilih kosong.
Perintah tidak dilanjutkan.
OK
Help
Simpan
Next
1000
5
To Win: 1000
To Win: 80

3.1.1.10. Ternak




Tubes OOP 1
Ternak
Pilih lokasi tanah di peternakan:
Peternakan
Petak tanah yang dipilih sudah terisi.
Perintah gagal dieksekusi.
OK
Dengan hati-hati, kamu meletakkan seekor Cow di petak tanah G02.
Cow telah menjadi peliharaanmu sekarang!
Pilih hewan dari penyimpanan
Penyimpanan
Kamu memilih Cow dari penyimpanan.
Lokasi yang dipilih kosong.
Perintah tidak dilanjutkan.

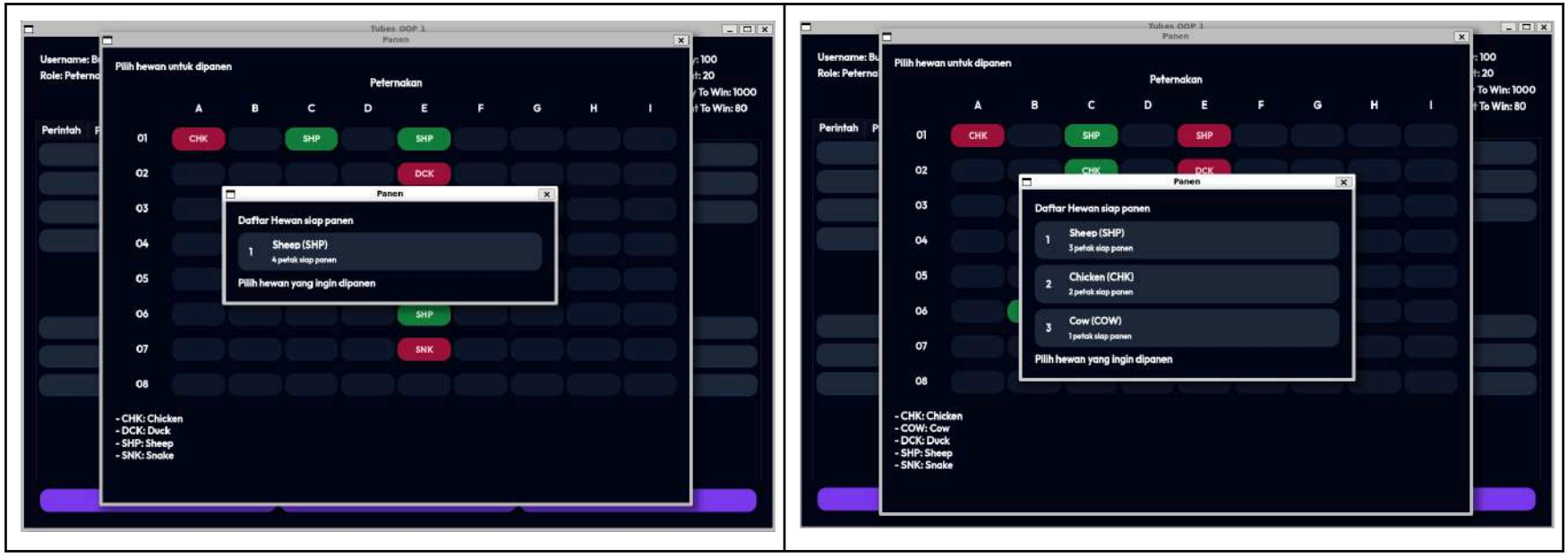
3.1.1.11. Kasih Makan

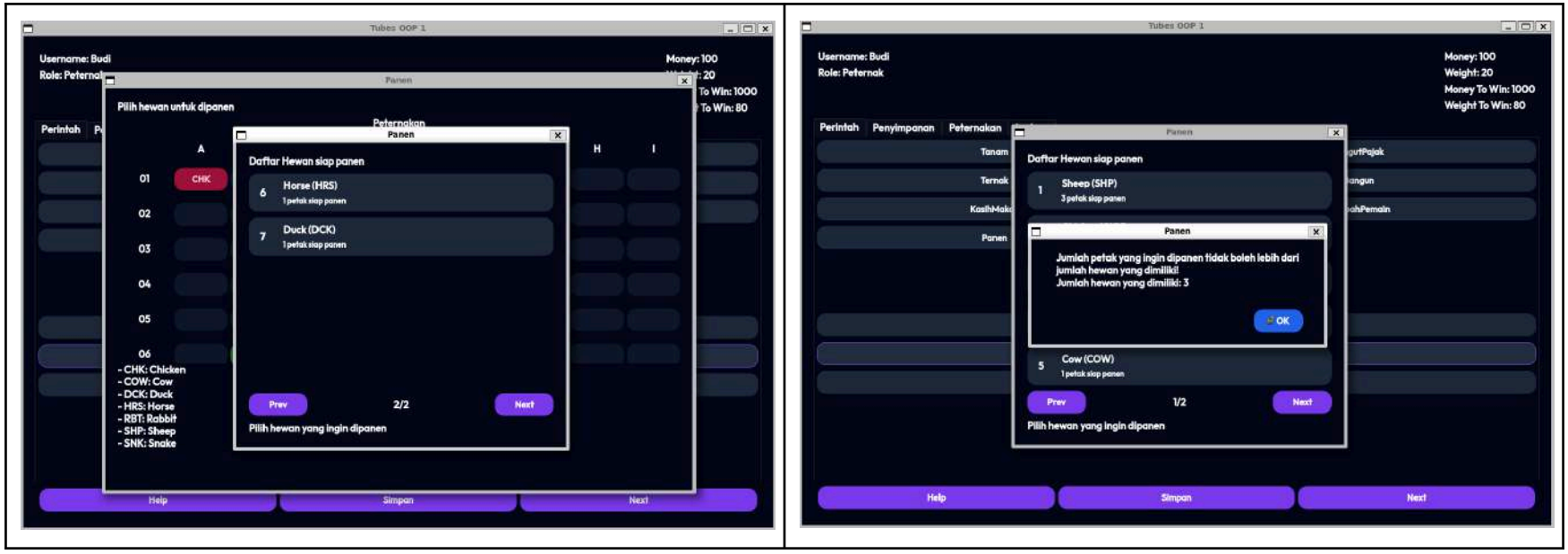


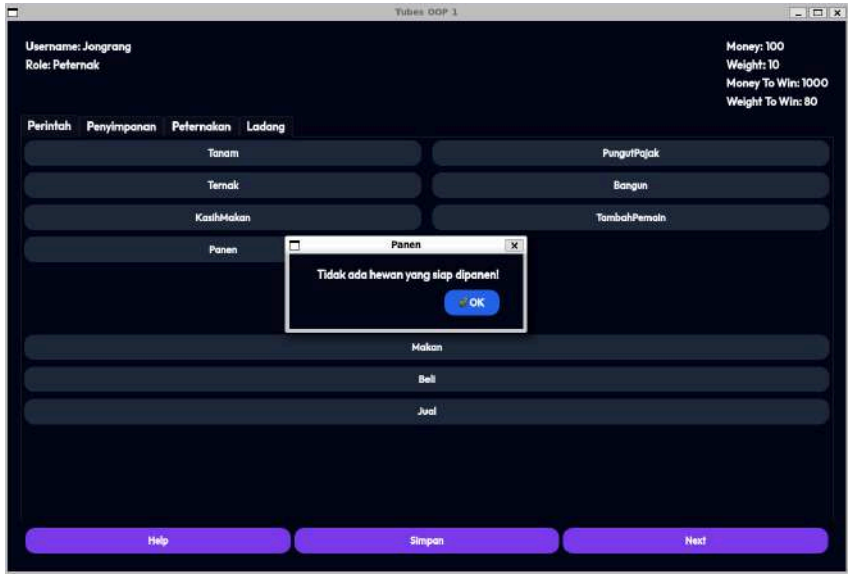

Tubes OOP 1
Username: Budi
Money: 100
Kasih Makan
Kamu memilih Sheep untuk diberi makan.
Pilih pangan yang akan diberikan:
Penyimpanan
A B C D E F G H I J
01 IRW COW COM COW ALT SMH TAW APP SHM
02
03
04
05
06
07
08
09
10
Kasih Makan
Benda tersebut tidak dapat dimakan oleh hewan!
OK
Help
Simpan
Next

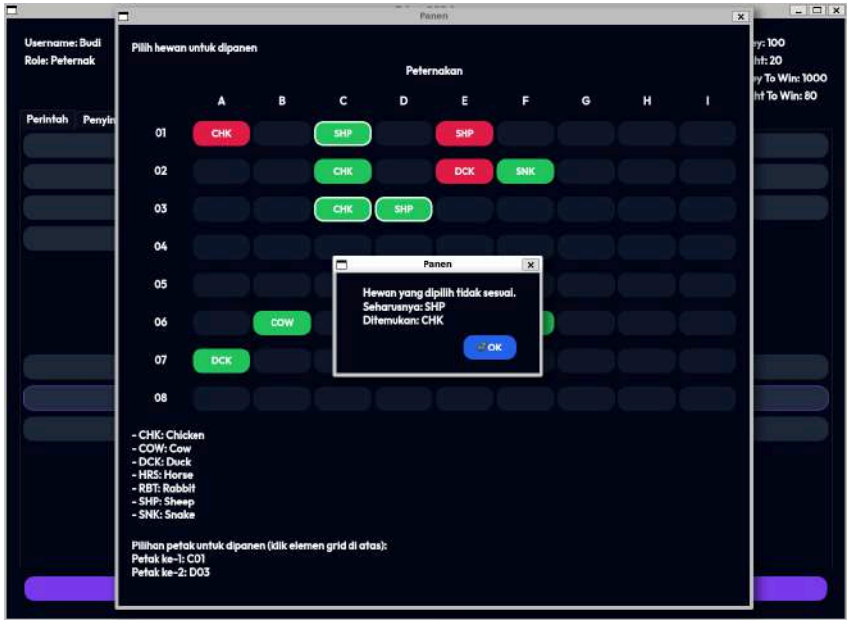

Tubes OOP 1
Username: Budi
Money: 100
Kasih Makan
Kamu memilih Sheep untuk diberi makan.
Pilih pangan yang akan diberikan:
Penyimpanan
A B C D E F G H I J
01 IRW COW COM COW ALT SMH TAW APP SHM
02
03
04
05
06
07
08
09
10
Kasih Makan
Sheep tidak bisa makan cow meat
OK
Help
Simpan
Next
Tubes OOP 1
Username: Budi
Money: 100
Kasih Makan
Kamu memilih Sheep untuk diberi makan.
Pilih pangan yang akan diberikan:
Penyimpanan
A B C D E F G H I J
01 IRW COW COM COW ALT SMH TAW SHM
02
03
04
05
06
07
08
09
10
Kasih Makan
Sheep telah diberi makan Apple !
Beratnya menjadi 16.
OK
Help
Simpan
Next

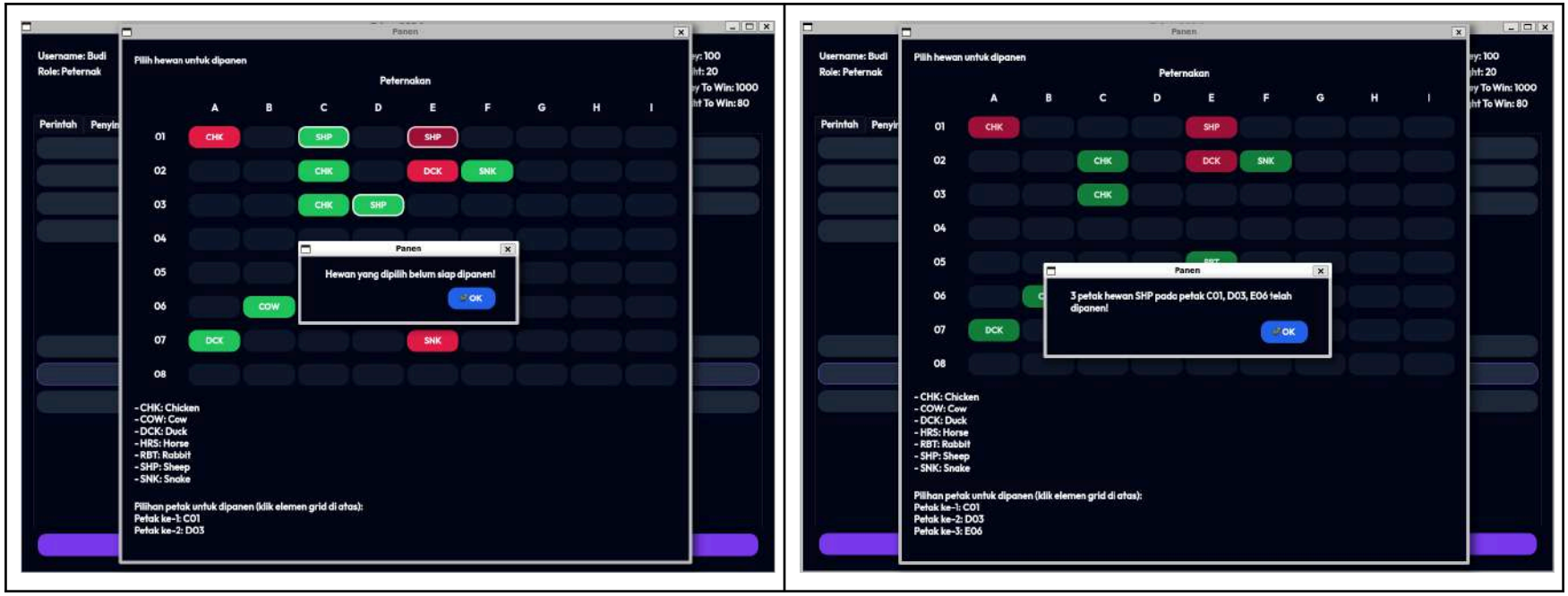
3.1.1.12. Panen


Pilih hewan untuk dipanen
Peternakan
Panen
Daftar Hewan siap panen
1 Sheep (SHP)
4 petak siap panen
Pilih hewan yang ingin dipanen
- CHK: Chicken
- DCK: Duck
- SHP: Sheep
- SNK: Snake
Pilih hewan untuk dipanen
Peternakan
Panen
Daftar Hewan siap panen
1 Sheep (SHP)
3 petak siap panen
2 Chicken (CHK)
2 petak siap panen
3 Cow (COW)
1 petak siap panen
Pilih hewan yang ingin dipanen
- CHK: Chicken
- COW: Cow
- DCK: Duck
- SHP: Sheep
- SNK: Snake

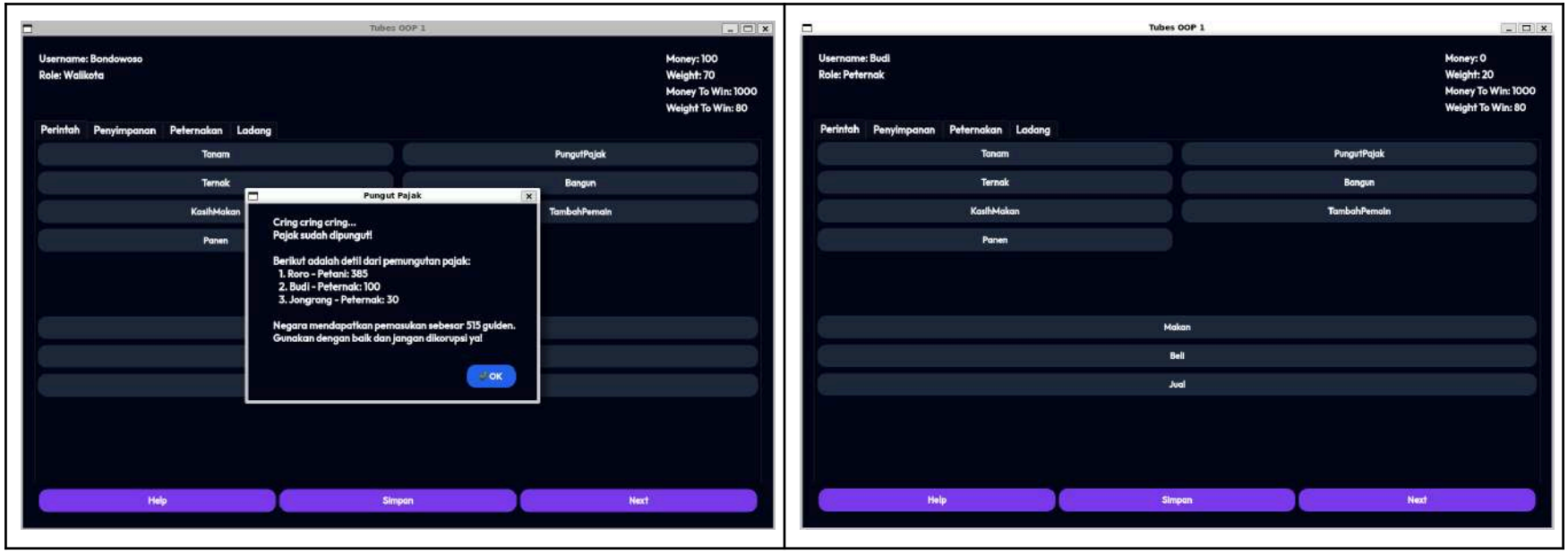

Tubes OOP 1
Username: Budi
Role: Peternak
Money: 100
Weight: 20
Money To Win: 1000
Weight To Win: 80
Perintah
Penyimpanan
Peternakan
Panen
Pilih hewan untuk dipanen
Daftar Hewan siap panen
Horse (HRS)
1 petak siap panen
Duck (DCK)
1 petak siap panen
Prev
2/2
Next
Pilih hewan yang ingin dipanen
- CHK: Chicken
- COW: Cow
- DCK: Duck
- HRS: Horse
- RBT: Rabbit
- SHP: Sheep
- SNK: Snake
Help
Simpan
Next
Sheep (SHP)
3 petak siap panen
Jumlah petak yang ingin dipanen tidak boleh lebih dari jumlah hewan yang dimiliki!
Jumlah hewan yang dimiliki: 3
OK
Cow (COW)
1 petak siap panen
1/2


Tubes OOP 1
Username: Jongrang
Role: Peternak
Money: 100
Weight: 10
Money To Win: 1000
Weight To Win: 80
Perintah
Penyimpanan
Peternakan
Ladang
Tanam
PungutPajak
Ternak
Bangun
KasihMakan
TambahPemain
Panen
Panen
Tidak ada hewan yang siap dipanen!
OK
Makan
Beli
Jual
Help
Simpan
Next

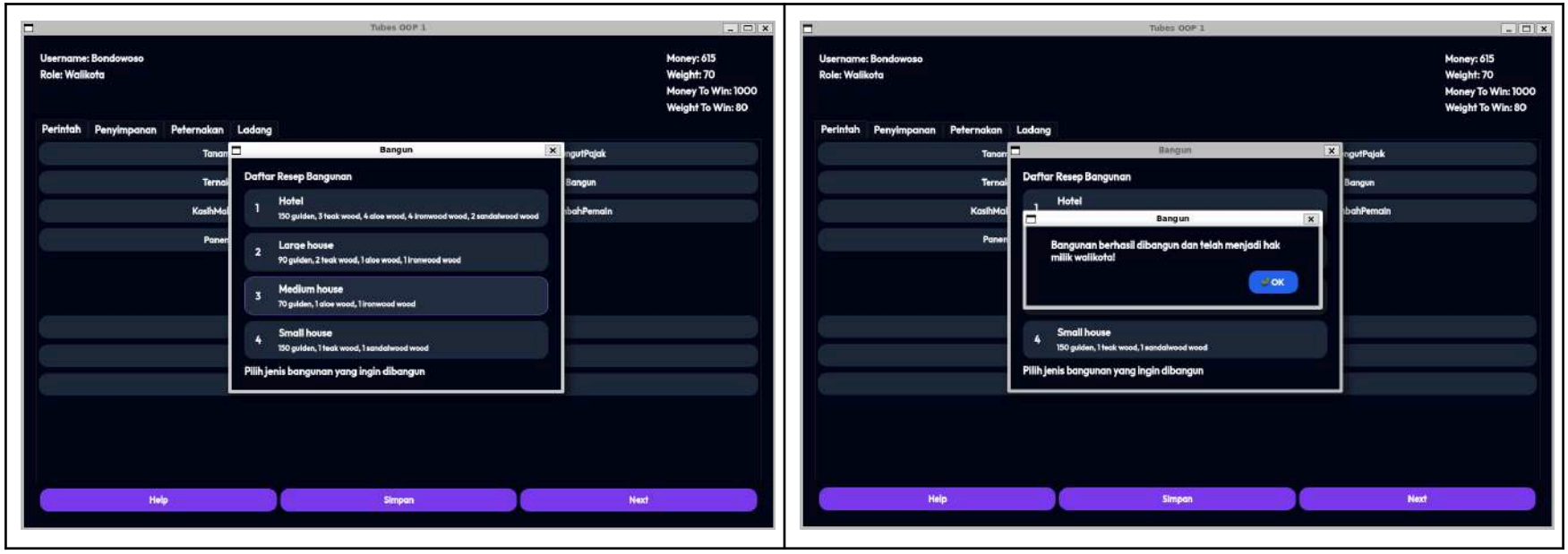

Username: Budi
Role: Peternak
Pilih hewan untuk dipanen
Peternakan
A B C D E F G H I
01 CHK SHP SHP
02 CHK DCK SNK
03 CHK SHP
04
05
06 COW
07 DCK
08
Panen
Hewan yang dipilih tidak sesuai.
Seharusnya: SHP
Ditemukan: CHK
OK
- CHK: Chicken
- COW: Cow
- DCK: Duck
- HRS: Horse
- RBT: Rabbit
- SHP: Sheep
- SNK: Snake
Pilihan petak untuk dipanen (klik elemen grid di atas):
Petak ke-1: C01
Petak ke-2: D03

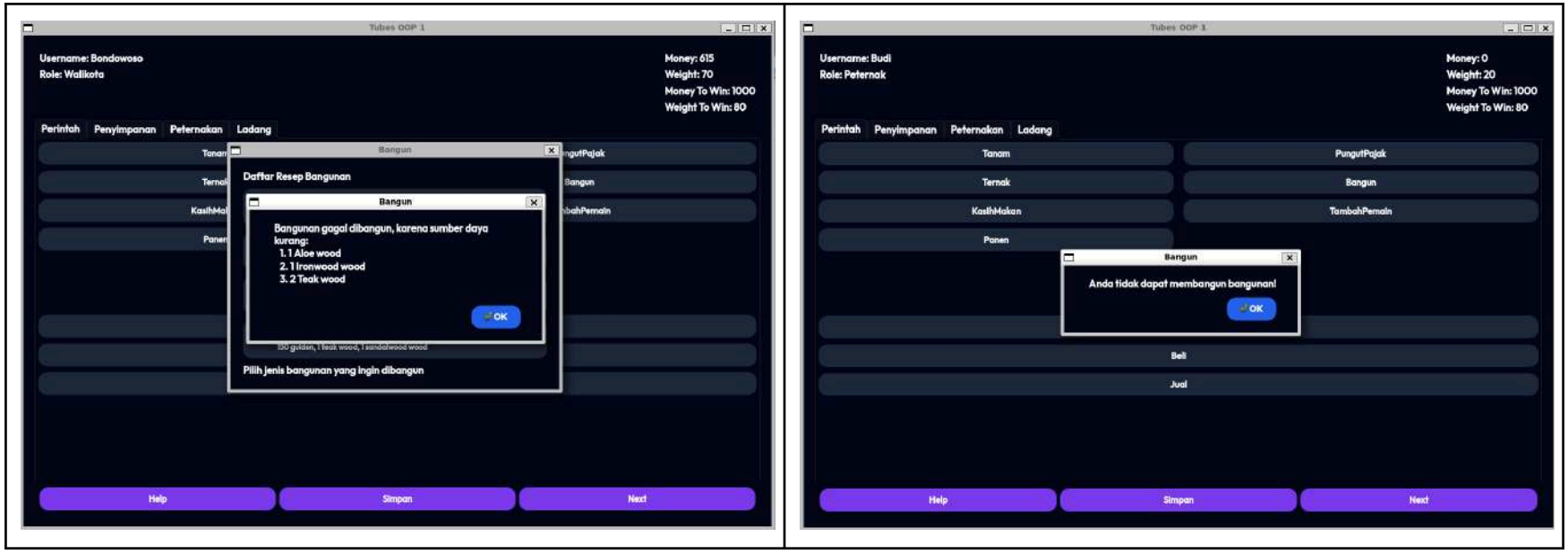

Username: Budi
Role: Peternak
Pilih hewan untuk dipanen
Peternakan
Panen
Hewan yang dipilih belum siap dipanen!
OK
- CHK: Chicken
- COW: Cow
- DCK: Duck
- HRS: Horse
- RBT: Rabbit
- SHP: Sheep
- SNK: Snake
Pilihan petak untuk dipanen (klik elemen grid di atas):
Petak ke-1: C01
Petak ke-2: D03
Username: Budi
Role: Peternak
Pilih hewan untuk dipanen
Peternakan
Panen
3 petak hewan SHP pada petak C01, D03, E06 telah dipanen!
OK
- CHK: Chicken
- COW: Cow
- DCK: Duck
- HRS: Horse
- RBT: Rabbit
- SHP: Sheep
- SNK: Snake
Pilihan petak untuk dipanen (klik elemen grid di atas):
Petak ke-1: C01
Petak ke-2: D03
Petak ke-3: E06

3.1.1.13. Pungut Pajak

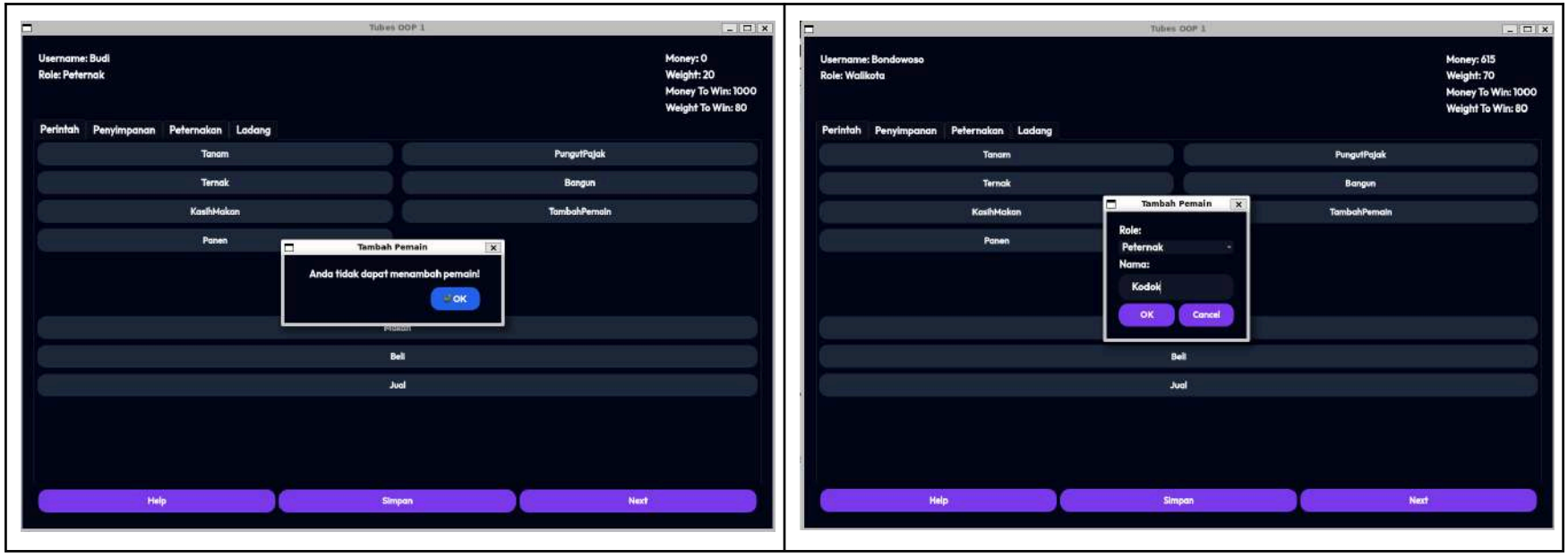


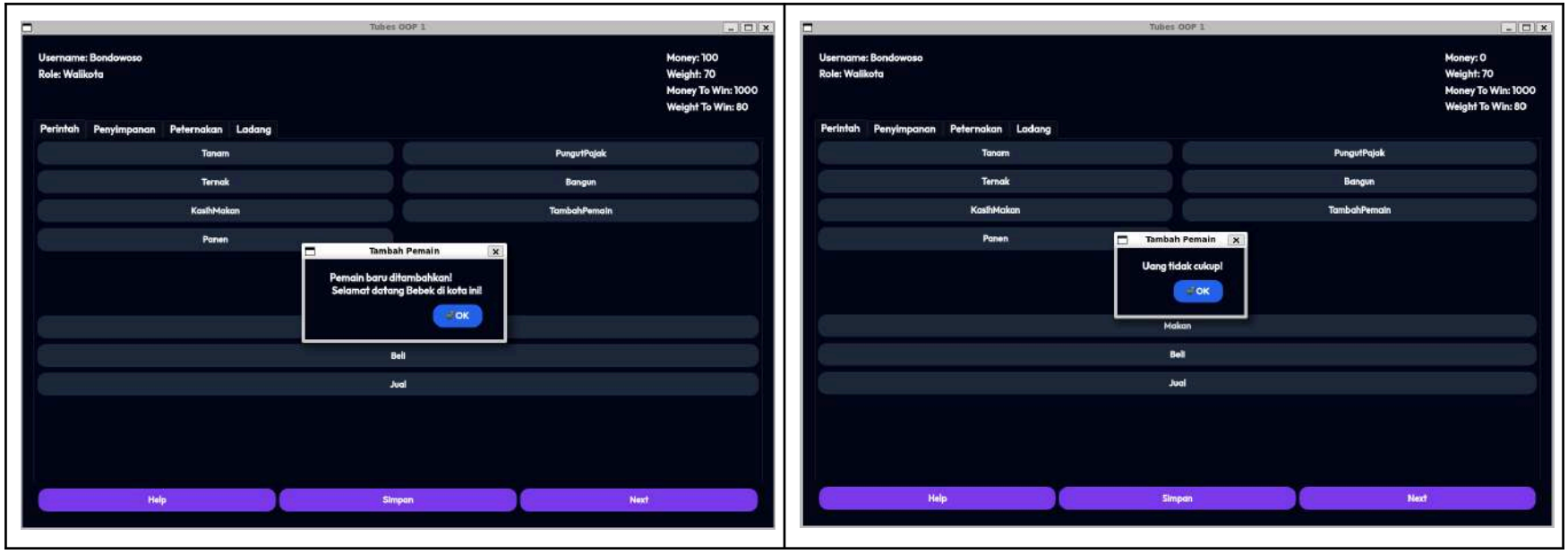
3.1.1.14. Bangun



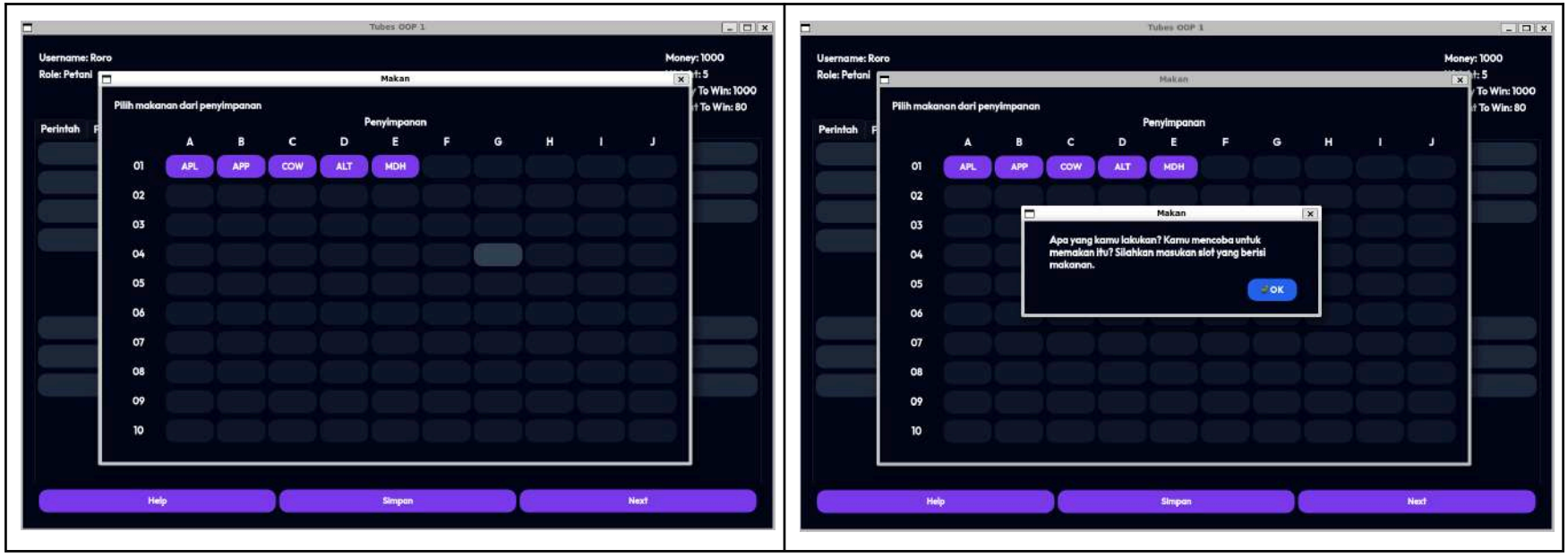

Tubes OOP 1
Username: Bondowoso
Role: Walikota
Money: 615
Weight: 70
Money To Win: 1000
Weight To Win: 80
Perintah
Penyimpanan
Peternakan
Ladang
Bangun
Daftar Resep Bangunan
Bangun
Bangunan gagal dibangun, karena sumber daya kurang:
1. 1 Aloe wood
2. 1 Ironwood wood
3. 2 Teak wood
OK
Pilih jenis bangunan yang ingin dibangun
Help
Simpan
Next
Tubes OOP 1
Username: Budi
Role: Peternak
Money: 0
Weight: 20
Money To Win: 1000
Weight To Win: 80
Perintah
Penyimpanan
Peternakan
Ladang
Tanam
Ternak
KasihMakan
Panen
PungutPajak
Bangun
TambahPemain
Bangun
Anda tidak dapat membangun bangunan!
OK
Beli
Jual
Help
Simpan
Next

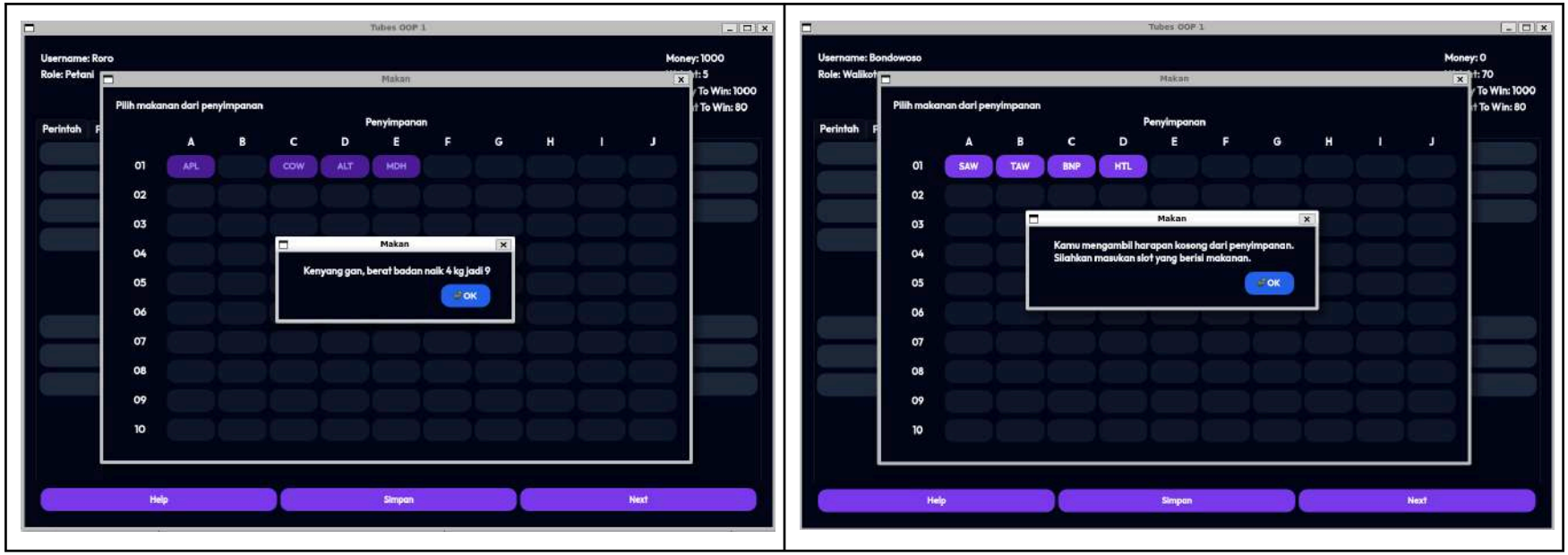
3.1.1.15. Tambah Pemain



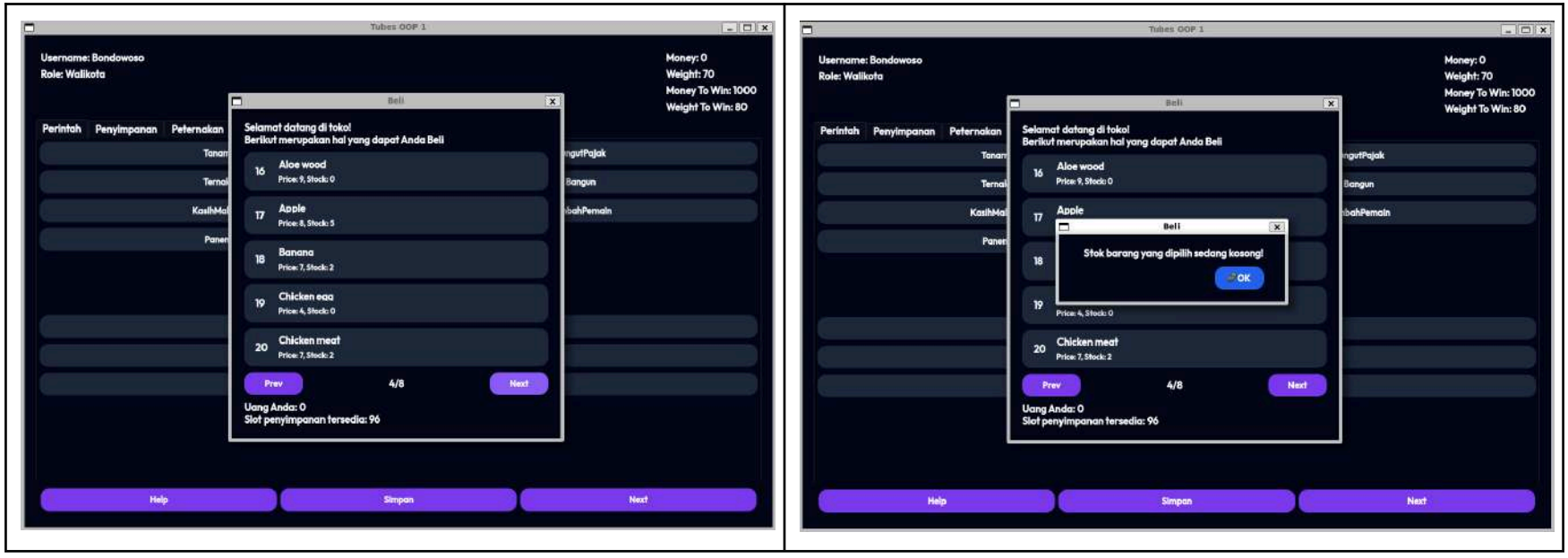

Tubes OOP 1
Username: Bondowoso
Role: Walikota
Money: 100
Weight: 70
Money To Win: 1000
Weight To Win: 80
Perintah
Penyimpanan
Peternakan
Ladang
Tanam
PungutPajak
Ternak
Bangun
KasihMakan
TambahPemain
Panen
Tambah Pemain
Pemain baru ditambahkan!
Selamat datang Bebek di kota ini!
OK
Beli
Jual
Help
Simpan
Next
Tubes OOP 1
Username: Bondowoso
Role: Walikota
Money: 0
Weight: 70
Money To Win: 1000
Weight To Win: 80
Perintah
Penyimpanan
Peternakan
Ladang
Tanam
PungutPajak
Ternak
Bangun
KasihMakan
TambahPemain
Panen
Tambah Pemain
Uang tidak cukup!
OK
Makan
Beli
Jual
Help
Simpan
Next

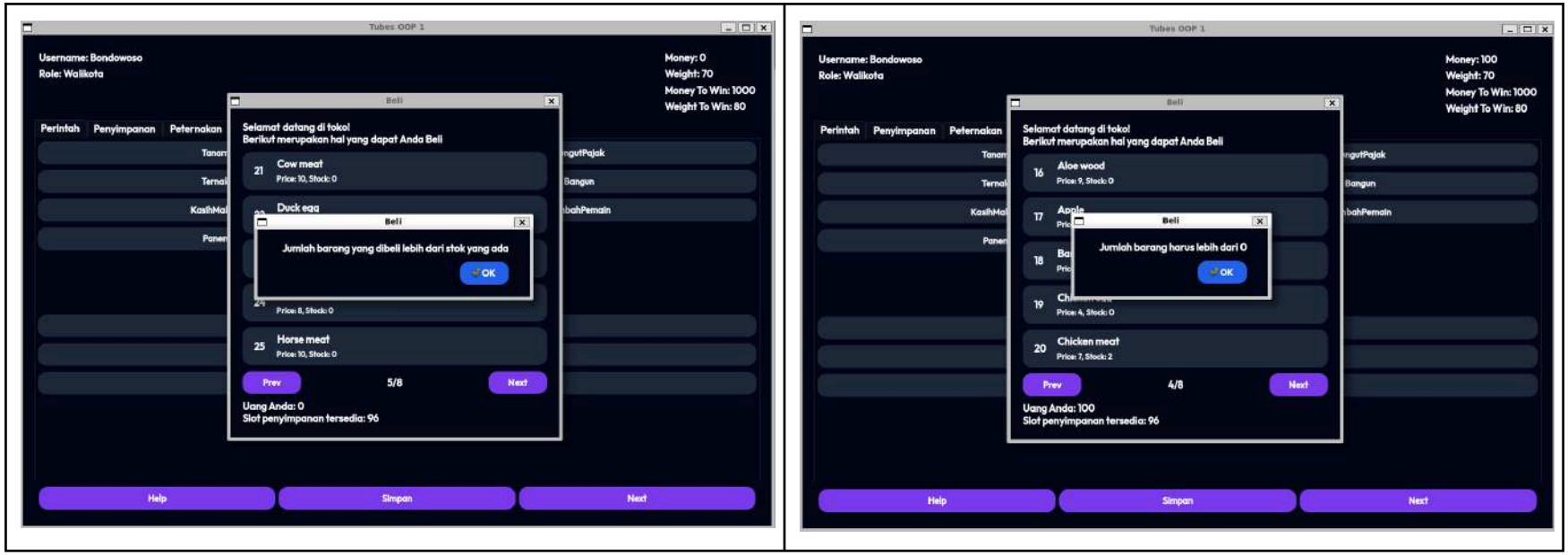
3.1.1.16. Makan

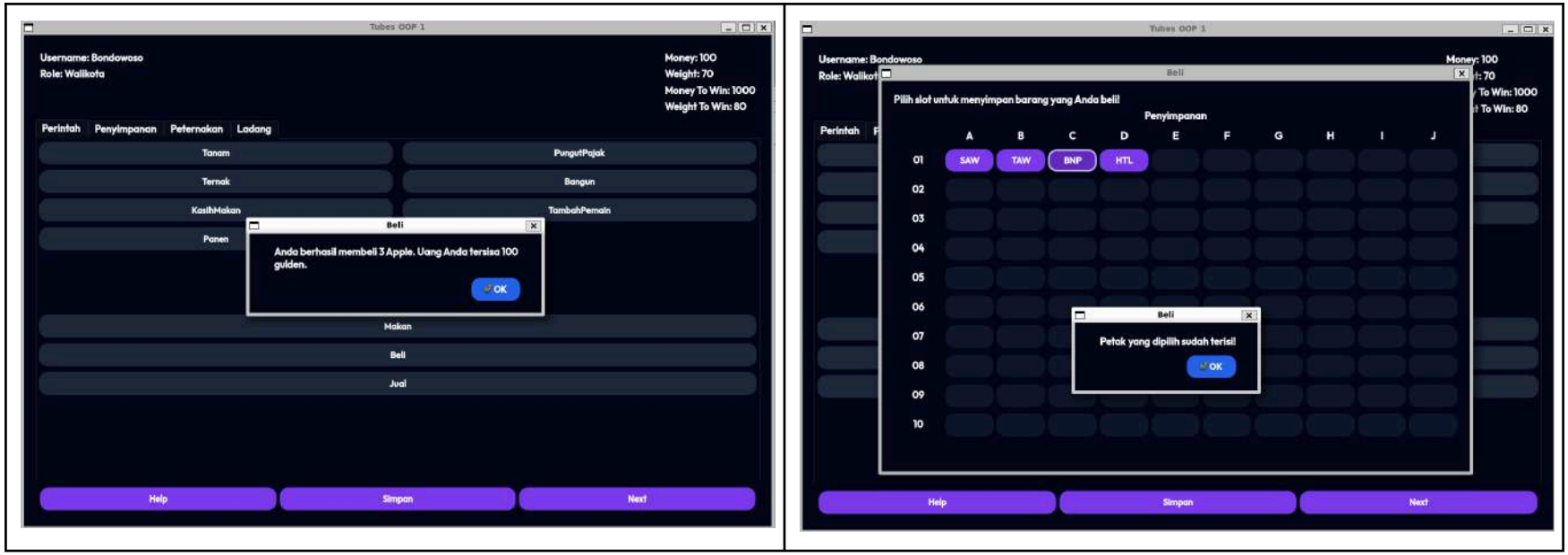

Tubes OOP 1
Username: Roro
Role: Petani
Money: 1000
Makan
Pilih makanan dari penyimpanan
Penyimpanan
A B C D E F G H I J
01 02 03 04 05 06 07 08 09 10
APL APP COW ALT MDH
Perintah
Help
Simpan
Next
Makan
Apa yang kamu lakukan? Kamu mencoba untuk memakan itu? Silahkan masukan slot yang berisi makanan.
OK

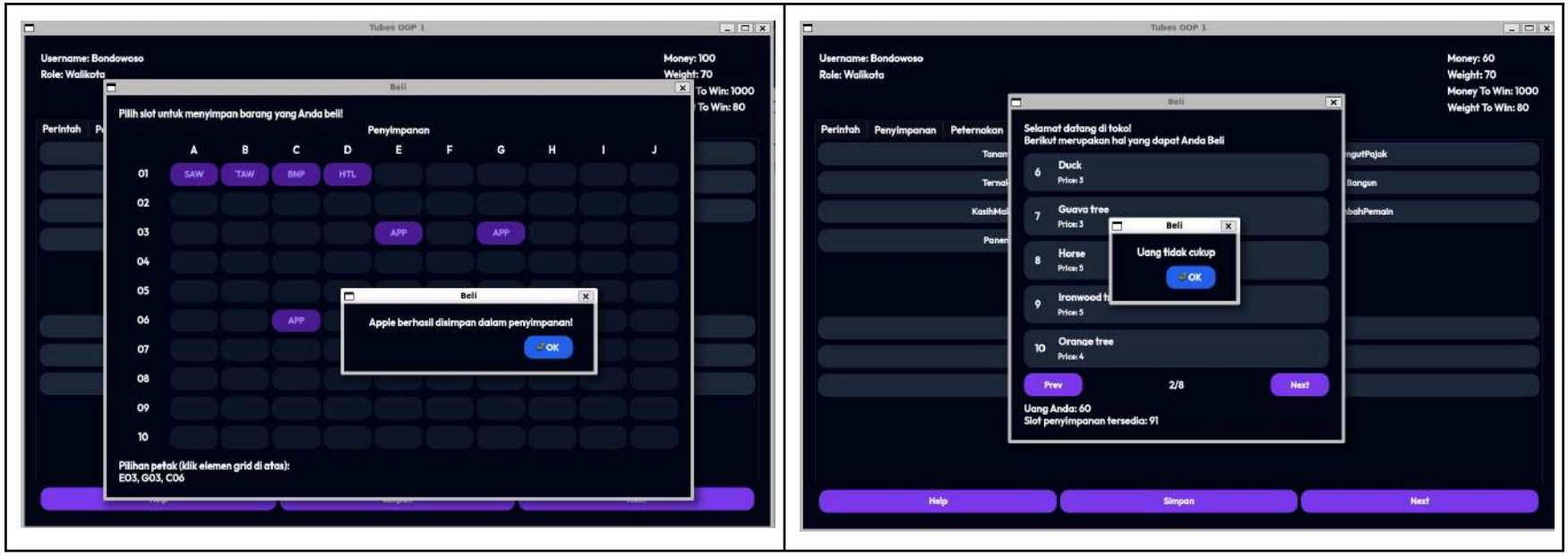

Tubes OOP 1
Username: Roro
Role: Petani
Money: 1000
Makan
Pilih makanan dari penyimpanan
Penyimpanan
A B C D E F G H I J
01 APL COW ALT MDH
02
03
04
05
06
07
08
09
10
Makan
Kenyang gan, berat badan naik 4 kg jadi 9
OK
Help
Simpan
Next
Tubes OOP 1
Username: Bondowoso
Role: Walikot
Money: 0
Makan
Pilih makanan dari penyimpanan
Penyimpanan
A B C D E F G H I J
01 SAW TAW BNP HTL
02
03
04
05
06
07
08
09
10
Makan
Kamu mengambil harapan kosong dari penyimpanan.
Silahkan masukan slot yang berisi makanan.
OK
Help
Simpan
Next

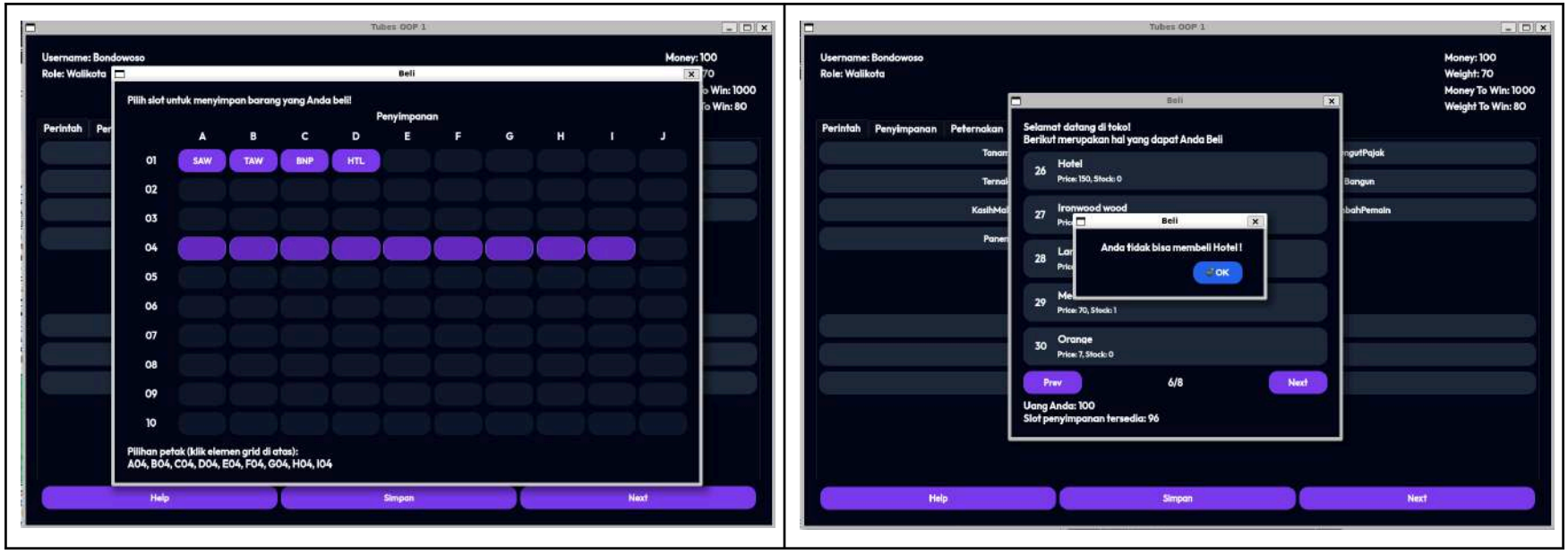
3.1.1.17. Beli


Tubes OOP 1
Username: Bondowoso
Role: Walikota
Money: 0
Weight: 70
Money To Win: 1000
Weight To Win: 80
Perintah
Penyimpanan
Peternakan
Beli
Selamat datang di toko!
Berikut merupakan hal yang dapat Anda Beli
16 Aloe wood
Price: 9, Stock: 0
17 Apple
Price: 8, Stock: 5
18 Banana
Price: 7, Stock: 2
19 Chicken eaa
Price: 4, Stock: 0
20 Chicken meat
Price: 7, Stock: 2
Prev
4/8
Next
Uang Anda: 0
Slot penyimpanan tersedia: 96
Help
Simpan
Next
Stok barang yang dipilih sedang kosong!
OK

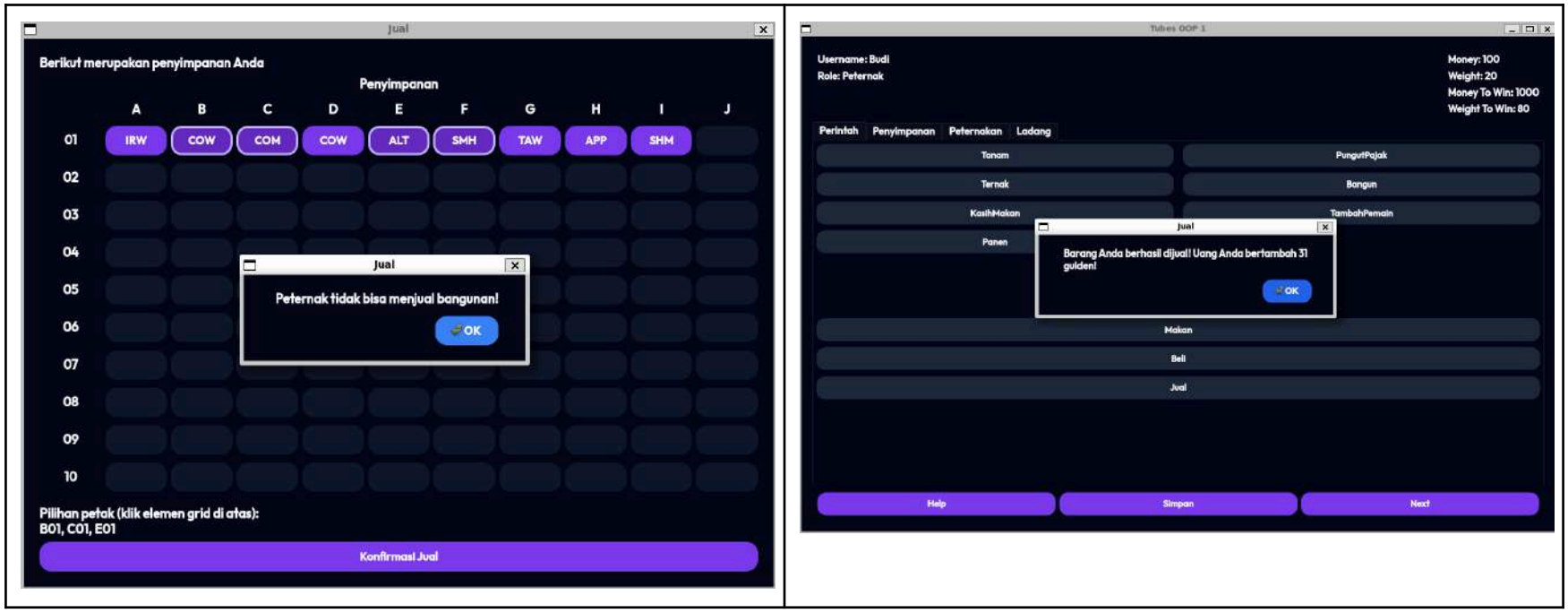

Tubes OOP 1
Username: Bondowoso
Role: Walikota
Money: 0
Weight: 70
Money To Win: 1000
Weight To Win: 80
Perintah
Penyimpanan
Peternakan
Beli
Selamat datang di toko!
Berikut merupakan hal yang dapat Anda Beli
21 Cow meat
Price: 10, Stock: 0
Duck egg
Price: 8, Stock: 0
25 Horse meat
Price: 10, Stock: 0
Prev
5/8
Next
Uang Anda: 0
Slot penyimpanan tersedia: 96
Jumlah barang yang dibeli lebih dari stok yang ada
OK
Help
Simpan
Next
Tubes OOP 1
Username: Bondowoso
Role: Walikota
Money: 100
Weight: 70
Money To Win: 1000
Weight To Win: 80
Perintah
Penyimpanan
Peternakan
Beli
Selamat datang di toko!
Berikut merupakan hal yang dapat Anda Beli
16 Aloe wood
Price: 9, Stock: 0
17 Apple
18
19
Price: 4, Stock: 0
20 Chicken meat
Price: 7, Stock: 2
Prev
4/8
Next
Uang Anda: 100
Slot penyimpanan tersedia: 96
Jumlah barang harus lebih dari 0
OK
Help
Simpan
Next

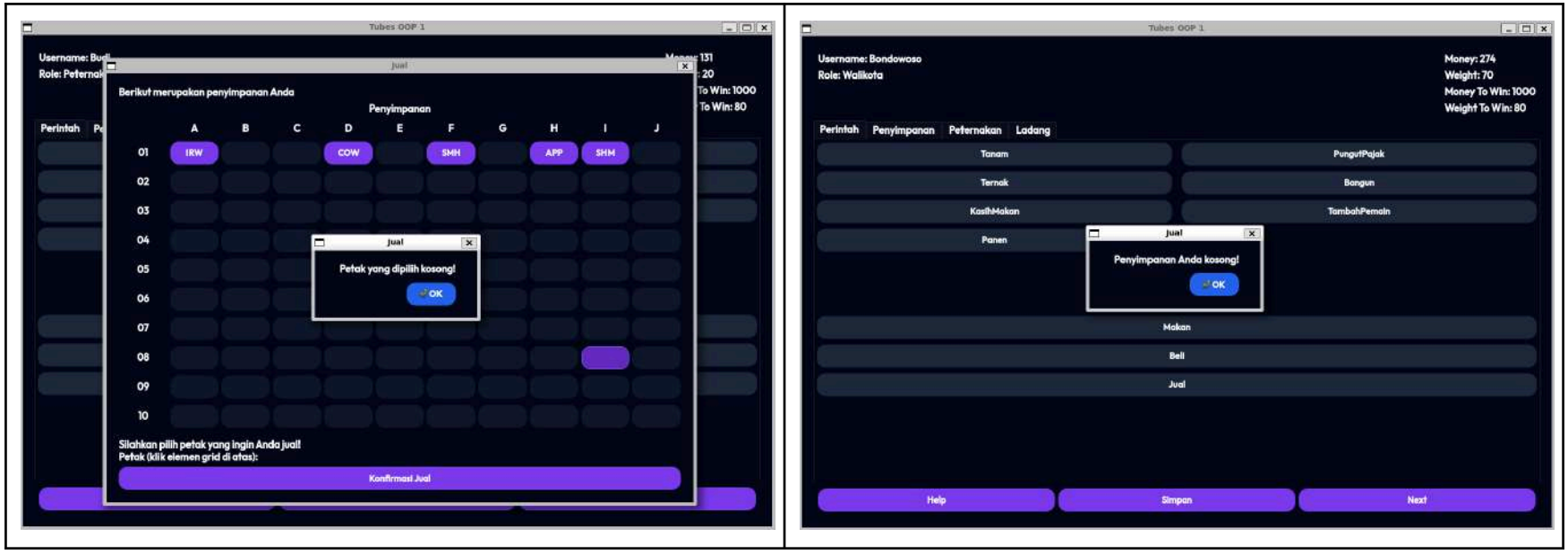

Tubes OOP 1
Username: Bondowoso
Role: Walikota
Money: 100
Weight: 70
Money To Win: 1000
Weight To Win: 80
Perintah
Penyimpanan
Peternakan
Ladang
Tanam
PungutPajak
Ternak
Bangun
KasihMakan
TambahPemain
Panen
Beli
Anda berhasil membeli 3 Apple. Uang Anda tersisa 100 gulden.
OK
Makan
Beli
Jual
Help
Simpan
Next
Pilih slot untuk menyimpan barang yang Anda beli!
Penyimpanan
A B C D E F G H I J
01 02 03 04 05 06 07 08 09 10
SAW
TAW
BNP
HTL
Petak yang dipilih sudah terisi!


Tubes OOP 1
Username: Bondowoso
Role: Walikota
Money: 100
Weight: 70
To Win: 1000
To Win: 80
Perintah
Beli
Pilih slot untuk menyimpan barang yang Anda beli!
Penyimpanan
A B C D E F G H I J
01 SAW TAW BNP HTL
02
03 APP APP
04
05
06 APP
07
08
09
10
Pilihan petak (klik elemen grid di atas):
E03, G03, C06
Beli
Apple berhasil disimpan dalam penyimpanan!
OK
Tubes OOP 1
Username: Bondowoso
Role: Walikota
Money: 60
Weight: 70
Money To Win: 1000
Weight To Win: 80
Perintah
Penyimpanan
Peternakan
Beli
Selamat datang di toko!
Berikut merupakan hal yang dapat Anda Beli
6 Duck
Price: 3
7 Guava tree
Price: 3
8 Horse
Price: 5
9
Price: 5
10 Orange tree
Price: 4
Prev
2/8
Next
Uang Anda: 60
Slot penyimpanan tersedia: 91
Beli
Uang tidak cukup
OK
Help
Simpan
Next


Tubes OOP 1
Username: Bondowoso
Role: Walikota
Money: 100
Beli
Pilih slot untuk menyimpan barang yang Anda beli!
Penyimpanan
A B C D E F G H I J
01 SAW TAW BNP HTL
02
03
04
05
06
07
08
09
10
Pilihan petak (klik elemen grid di atas):
A04, B04, C04, D04, E04, F04, G04, H04, I04
Help
Simpan
Next
Tubes OOP 1
Username: Bondowoso
Role: Walikota
Money: 100
Weight: 70
Money To Win: 1000
Weight To Win: 80
Perintah
Penyimpanan
Peternakan
Beli
Selamat datang di toko!
Berikut merupakan hal yang dapat Anda Beli
26 Hotel
Price: 150, Stock: 0
27 Ironwood wood
Beli
Anda tidak bisa membeli Hotel !
OK
29
Price: 70, Stock: 1
30 Orange
Price: 7, Stock: 0
Prev
6/8
Next
Uang Anda: 100
Slot penyimpanan tersedia: 96
Help
Simpan
Next

3.1.1.18. Jual


Jual
Berikut merupakan penyimpanan Anda
Penyimpanan
A B C D E F G H I J
01 IRW COW COM COW ALT SMH TAW APP SHM
02
03
04
05
06
07
08
09
10
Jual
Peternak tidak bisa menjual bangunan!
OK
Pilihan petak (klik elemen grid di atas):
B01, C01, E01
Konfirmasi Jual
Tubes OOP 1
Username: Budi
Role: Peternak
Money: 100
Weight: 20
Money To Win: 1000
Weight To Win: 80
Perintah
Penyimpanan
Peternakan
Ladang
Tanam
PungutPajak
Ternak
Bangun
KasihMakan
TambahPemain
Panen
Jual
Barang Anda berhasil dijual! Uang Anda bertambah 31 gulden!
OK
Makan
Beli
Jual
Help
Simpan
Next

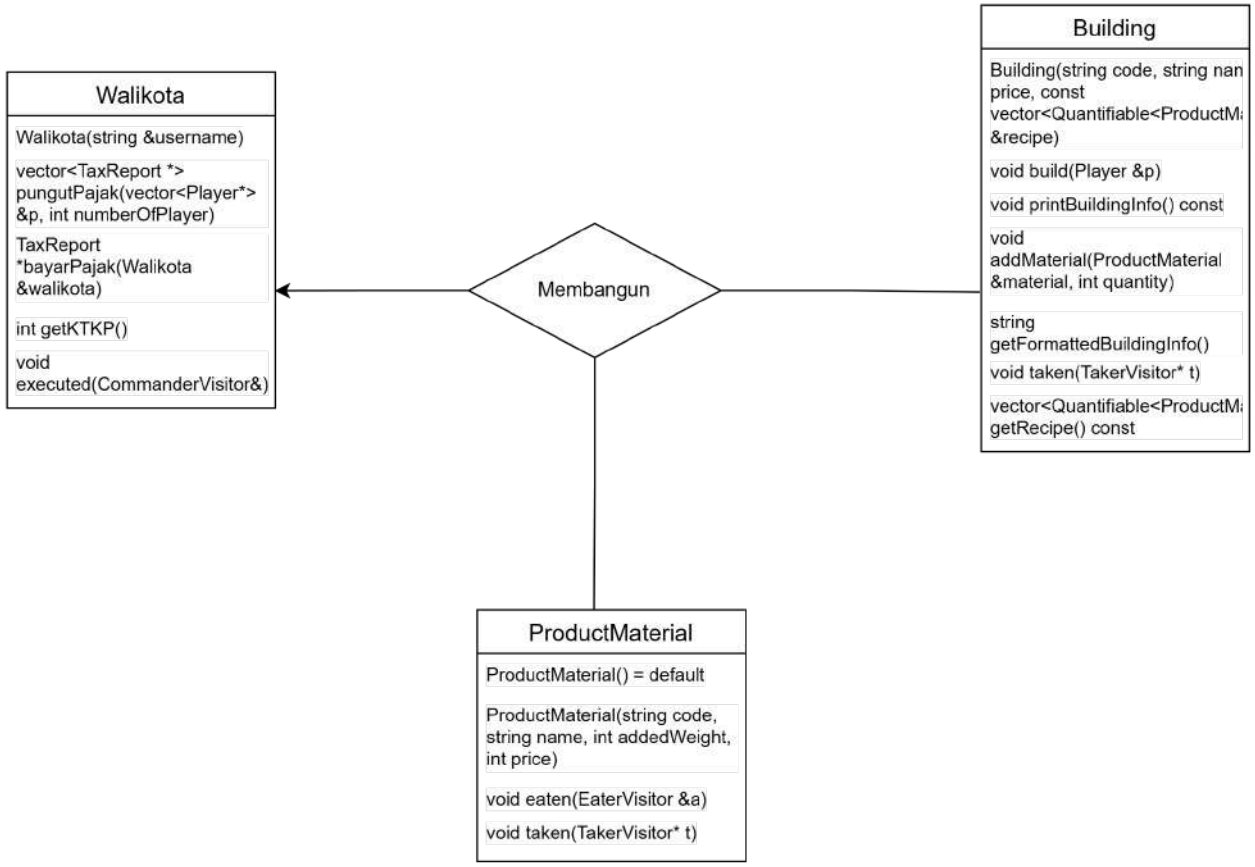


3.1.2. Diagram Sistem non UML

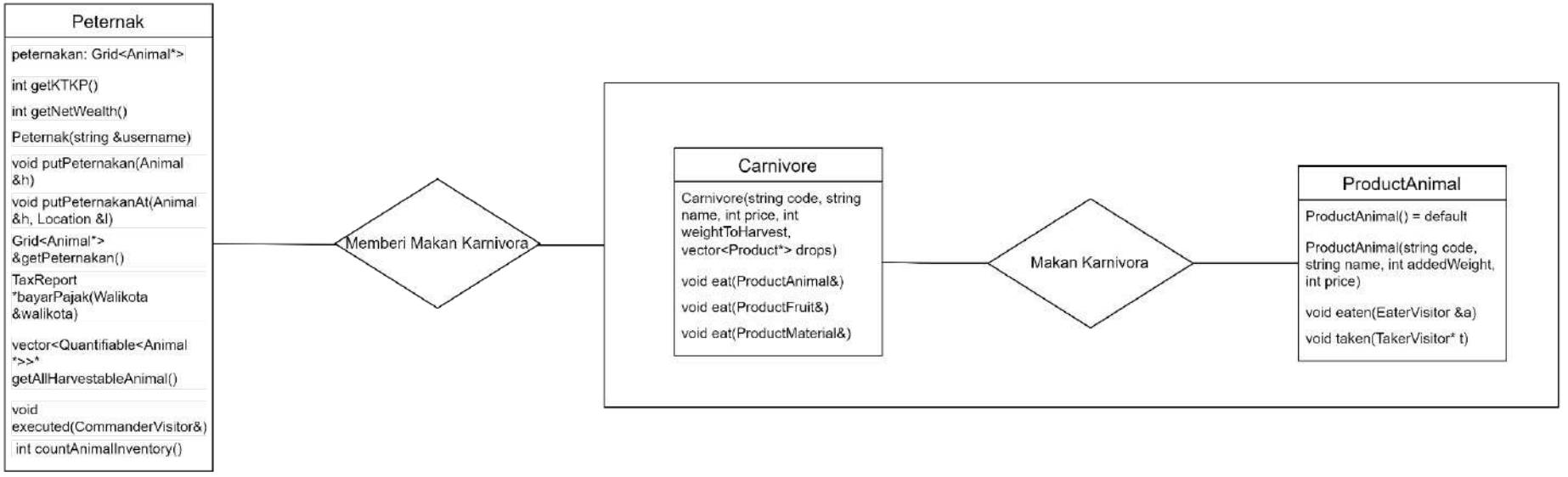
Kami memilih untuk menggunakan diagram ERD atau Entity Relationship Diagram sebagai alat visual untuk menggambarkan struktur data. Pemilihan ERD ini didasarkan pada beberapa alasan yang mempertimbangkan kebutuhan dan kemudahan pengimplementasian. ERD memiliki berkorespondensi dengan UML atau Unified Modeling Language, yang telah diimplementasikan pada spek wajib. Meskipun notasi yang digunakan berbeda, konsep-konsep seperti entitas, atribut, dan hubungan tetap terdapat dalam kedua jenis diagram ini.

Selain itu, ERD juga cocok untuk menggambarkan struktur data pada level tinggi dengan jelas. Pengguna dapat dengan mudah melihat bagaimana entitas saling terhubung dan berinteraksi satu sama lain. Fleksibilitas ERD dalam hal modifikasi juga menjadi pertimbangan penting. Ketika terdapat perubahan dalam struktur data atau hubungan antara entitas, diagram ini dapat diperbarui dengan mudah tanpa memerlukan perubahan yang signifikan. Diagram ini cenderung lebih mudah dimengerti bahkan oleh mereka yang tidak memiliki latar belakang teknis yang kuat.

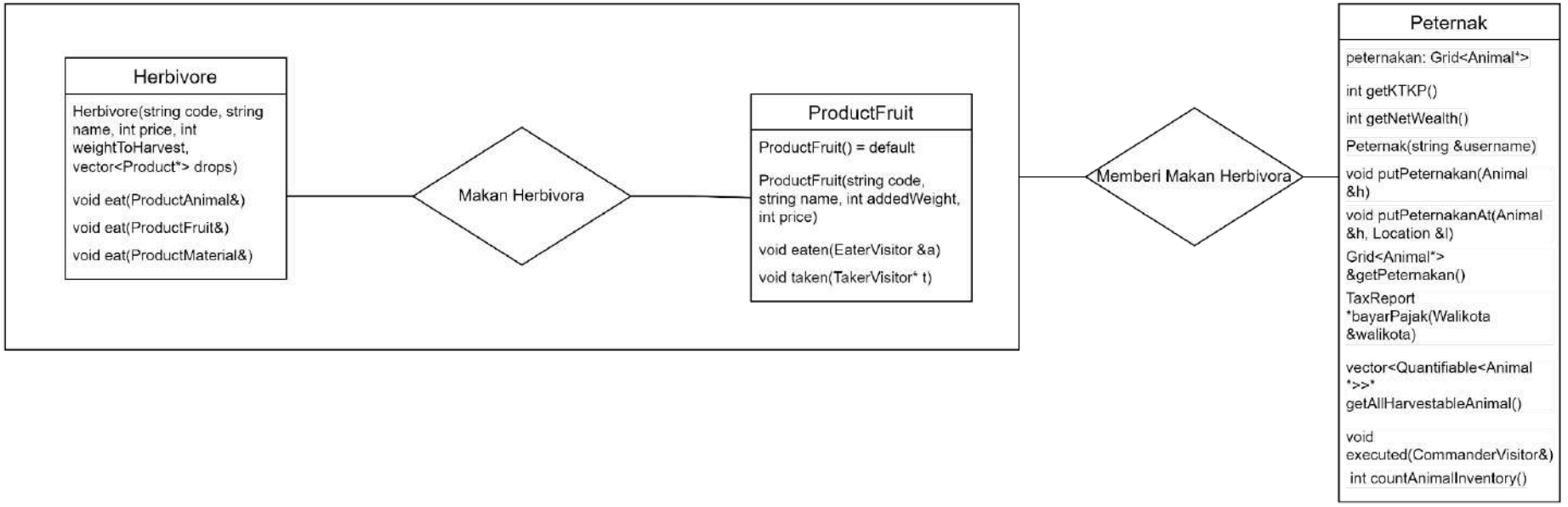
Berikut cuplikan diagram ERD:



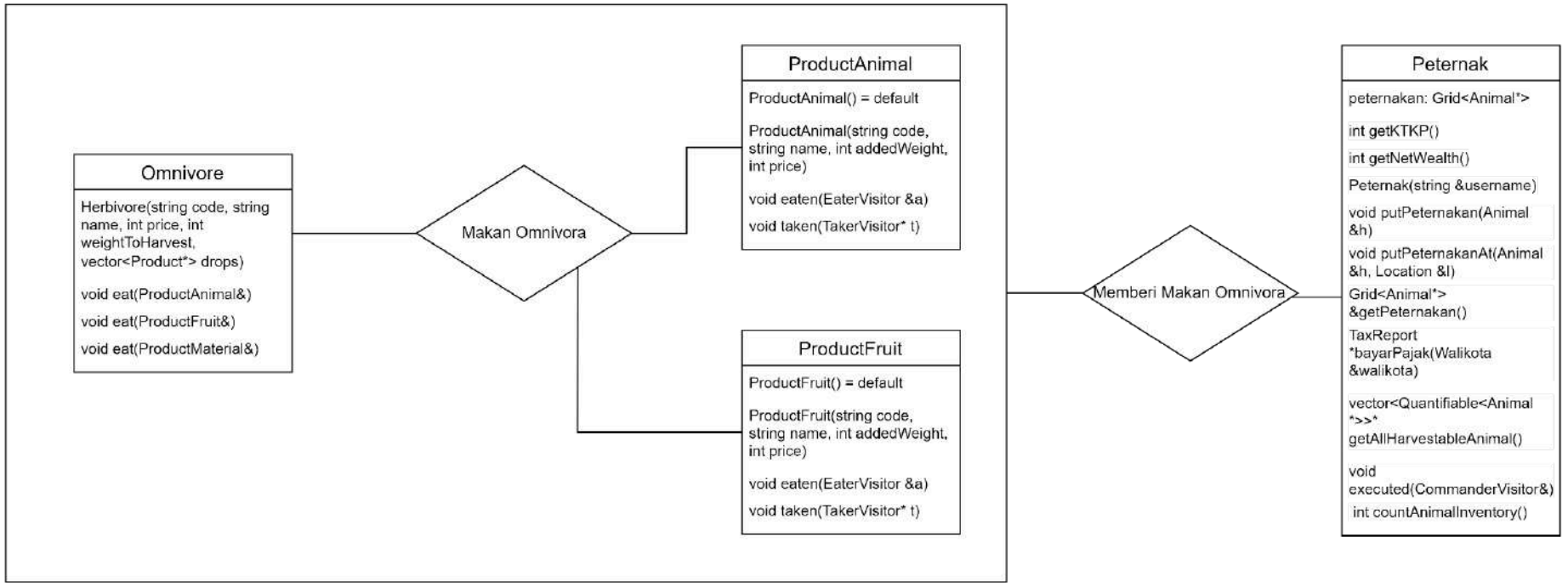
Gambar 3.1.2.1 Diagram ERD Memberi Makan pada Karnivora



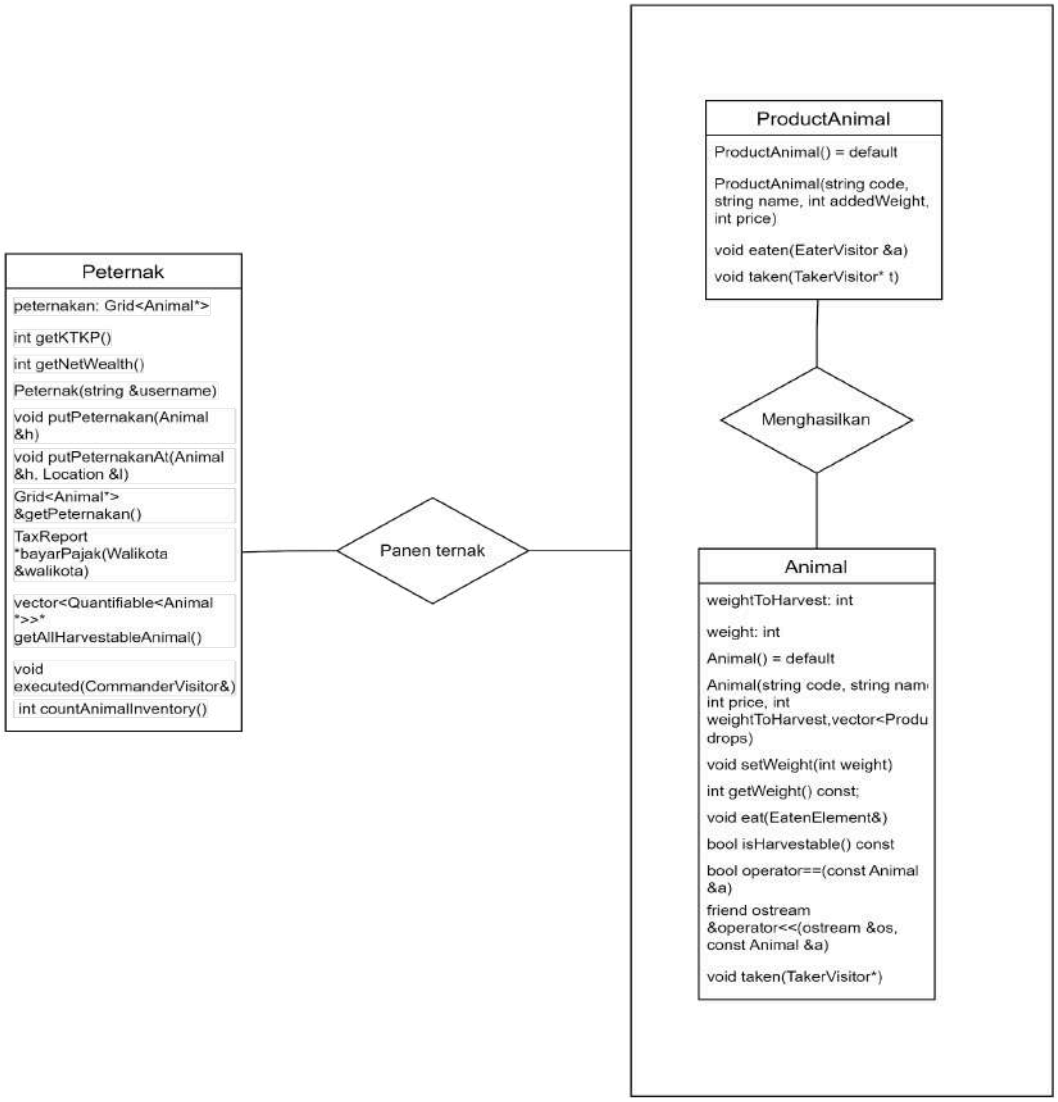
Gambar 3.1.2.2 Diagram ERD Memberi Makan pada Hewan Karnivora



Gambar 3.1.2.3 Diagram ERD Memberi Makan Hewan Herbivora



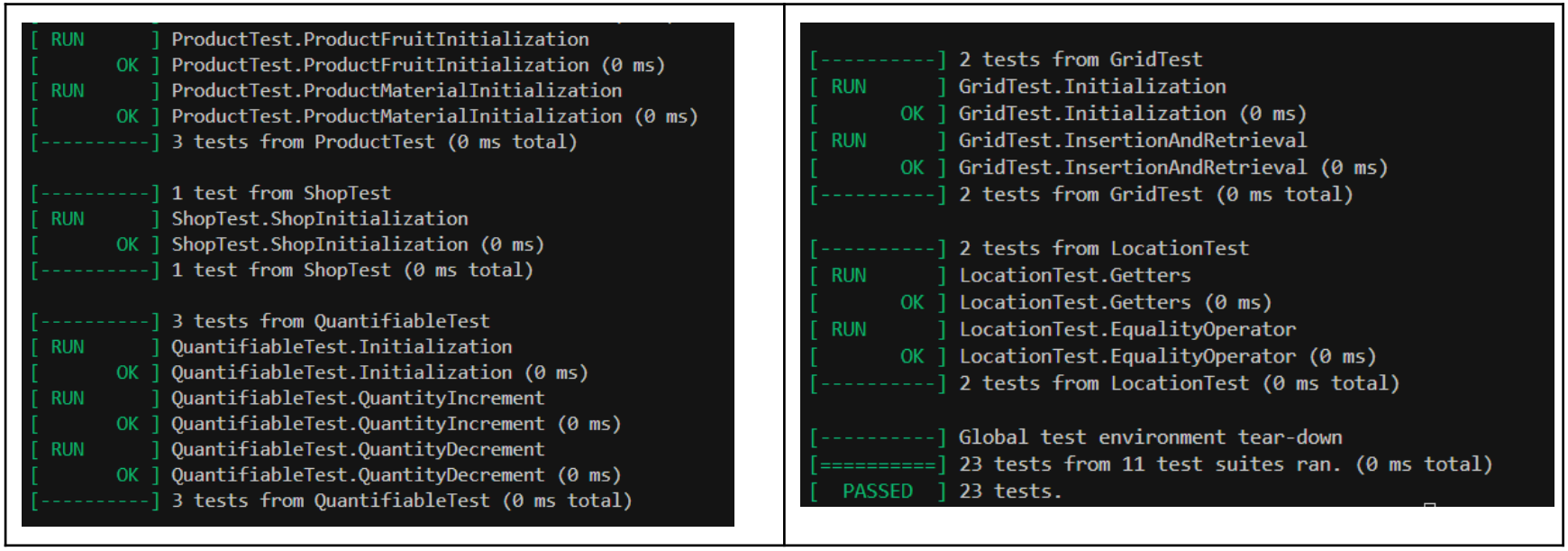
Gambar 3.1.2.4 Diagram ERD Memberi Makan pada Hewan Omnivora



Gambar 3.1.2.5 Diagram ERD Peternak Memanen Ternak

3.1.3. Unit Testing

Unit Testing Implementation menggunakan kerangka gtest buatan Google untuk menguji kode. Contoh screenshot di bawah hanya sebagian, untuk lebih lengkapnya berada pada pengumpulan tugas besar.



```
#include <gtest/gtest.h>
#include "tubesoop1/animal/animal.h"
#include "tubesoop1/animal/herbivore.h"
#include "tubesoop1/animal/omnivore.h"
#include "tubesoop1/animal/carnivore.h"

TEST(AnimalTest, AnimalInitiation)
{
    vector<Product*> emptyDrops;
    Herbivore tempH("HRB","HERBIVORE",5,10,emptyDrops);
    Omnivore tempO("OMN","OMNIVORE",2,12,emptyDrops);
    Carnivore tempC("CRN","CARNIVORE",3,30,emptyDrops);

    EXPECT_EQ("HRB", tempH.getCode());
    EXPECT_EQ("HERBIVORE", tempH.getName());
    EXPECT_EQ(5, tempH.getPrice());
    EXPECT_EQ(10, tempH.getWeightToHarvest());

    EXPECT_EQ("OMN", tempO.getCode());
    EXPECT_EQ("OMNIVORE", tempO.getName());
    EXPECT_EQ(2, tempO.getPrice());
    EXPECT_EQ(12, tempO.getWeightToHarvest());

    EXPECT_EQ("CRN", tempC.getCode());
    EXPECT_EQ("CARNIVORE", tempC.getName());
    EXPECT_EQ(3, tempC.getPrice());
    EXPECT_EQ(30, tempC.getWeightToHarvest());
```

```
}

TEST(AnimalTest, AnimalWeightManipulation)
{
    vector<Product*> emptyDrops;
    Herbivore tempH("HRB","HERBIVORE",5,10,emptyDrops);
    Omnivore tempO("OMN","OMNIVORE",2,12,emptyDrops);
    Carnivore tempC("CRN","CARNIVORE",3,30,emptyDrops);
    tempH.setWeight(100);
    tempO.setWeight(100);
    tempC.setWeight(100);

    EXPECT_EQ(100, tempH.getWeight());
    EXPECT_EQ(100, tempC.getWeight());
    EXPECT_EQ(100, tempO.getWeight());
}

TEST(AnimalTest, AnimalHarvestability)
{
    vector<Product*> drops;
    Herbivore temp("ANM", "Animal", 10, 50, drops);

    // By default, weight is 0, so it should not be harvestable
    EXPECT_FALSE(temp.isHarvestable());

    // Setting weight to be equal to weightToHarvest, it should be harvestable
    temp.setWeight(50);
```

```
    EXPECT_TRUE(temp.isHarvestable());
}
```

```
#include <gtest/gtest.h>
#include "tubesoop1/building/building.h"
#include "tubesoop1/product/product.h"
#include "tubesoop1/player/player.h"

TEST(BuildingTest, BuildingInitialization)
{
    vector<Quantifiable<ProductMaterial*>> recipe;
    Building temp("BDG", "Building", 100, recipe);

    EXPECT_EQ("BDG", temp.getCode());
    EXPECT_EQ("Building", temp.getName());
    EXPECT_EQ(100, temp.getPrice());
    EXPECT_EQ(0, temp.getRecipe().size());
}
```

```
#include <gtest/gtest.h>
#include <sstream>
#include "tubesoop1/plant/plant.h"
#include "tubesoop1/resource/creature.h" // Assuming this path and header file exist and are
correct.

// Mock dependencies
```

```
// Assuming Creature and Product classes are defined properly in their headers.
// If not, please define mock classes here. For brevity, it's assumed they are defined.

class MockCreature : public Creature {
    // Mock implementation of Creature's methods if needed
};

class MockProduct : public Product {
    // Mock implementation of Product's methods if needed
};

TEST(PlantTest, ConstructorWithParameters) {
    vector<Product*> drops; // Assuming a proper implementation of Product
    Plant plant("P001", "Sunflower", 50, 10, drops);
    EXPECT_EQ(plant.getAge(), 0); // Assuming initial age is set to 0 in all constructors
EXPECT_EQ(plant.isHarvestable(), false);
}

TEST(PlantTest, AgeManagement) {
    Plant plant;
    plant.setAge(5);
    EXPECT_EQ(plant.getAge(), 5);

    plant.addAge(3);
    EXPECT_EQ(plant.getAge(), 8);
}
```

```
TEST(PlantTest, Harvestability) {
    Plant plant("P001", "Sunflower", 50, 10, {});
    plant.setAge(5);
    EXPECT_FALSE(plant.isHarvestable());

    plant.setAge(10);
    EXPECT_TRUE(plant.isHarvestable());
}

TEST(PlantTest, EqualityOperator) {
    Plant plant1("P001", "Sunflower", 50, 10, {});
    Plant plant2("P001", "Sunflower", 50, 10, {});

    EXPECT_TRUE(plant1 == plant2);

    Plant plant3("P002", "Rose", 70, 5, {});
    EXPECT_FALSE(plant1 == plant3);
}


int main(int argc, char **argv) {
    ::testing::InitGoogleTest(&argc, argv);
    return RUN_ALL_TESTS();
}
```

```
#include <gtest/gtest.h>
#include "tubesoop1/product/product.h"
```

```
// #include "productvisitorpattern.h"
#include "tubesoop1/animal/animal_exception.h"

// Mock EaterVisitor for testing
class MockEaterVisitor : public EaterVisitor {
public:
    void eat(ProductAnimal& animal) override {
        // Mock implementation
    }

    void eat(ProductFruit& fruit) override {
        // Mock implementation
    }

    void eat(ProductMaterial& material) override {
        // Mock implementation
    }
};

TEST(ProductTest, ProductAnimalInitialization)
{
    // Test ProductAnimal initialization
    ProductAnimal animal("ANM", "Animal", 20, 80);

    EXPECT_EQ("ANM", animal.getCode());
    EXPECT_EQ("Animal", animal.getName());
    EXPECT_EQ(20, animal.getAddedWeight());
```

```
    EXPECT_EQ(80, animal.getPrice());
}

TEST(ProductTest, ProductFruitInitialization)
{
    // Test ProductFruit initialization
    ProductFruit fruit("FRU", "Fruit", 10, 30);

    EXPECT_EQ("FRU", fruit.getCode());
    EXPECT_EQ("Fruit", fruit.getName());
    EXPECT_EQ(10, fruit.getAddedWeight());
    EXPECT_EQ(30, fruit.getPrice());
}

TEST(ProductTest, ProductMaterialInitialization)
{
    // Test ProductMaterial initialization
    ProductMaterial material("MTL", "Material", 15, 50);

    EXPECT_EQ("MTL", material.getCode());
    EXPECT_EQ("Material", material.getName());
    EXPECT_EQ(15, material.getAddedWeight());
    EXPECT_EQ(50, material.getPrice());
}
```

3.1.4. Design Pattern

3.1.1.19. Factory with Lambda Pattern

Factory pattern merupakan pola desain dalam pemrograman untuk menginstantiate sebuah object tanpa perlu mengekspose bagaimana cara mengconstruct object tersebut. Pola desain ini berguna agar jika sebuah method ingin menggunakan sebuah object, tidak perlu langsung menyentuh constructor dari kelas tersebut. Jika kelas tersebut dimodifikasi, maka kita akan kesulitan jika kita memanggil constructornya dibanyak tempat. Dengan menggunakan factory pattern, kita bisa memisah hal ini sehingga yang perlu diubah hanyalah creator dari class nya saja. Creator class ini lah yang akan memanggil constructor object tadi dan menentukan bagaimana menginstansiasi object tersebut. Jika kita ingin menginstansiasi object tadi lagi tapi dengan cara yang berbeda, kita juga bisa membuat Creator class baru untuk memberikan variasi dari object tadi.

Umumnya pada pola desain factory, terdapat abstract class/interface untuk Creator dan Produk, lalu terdapat concrete class untuk creator dan produk. Namun terdapat cara lain yaitu dengan mengganti abstract class/interface creator menjadi type function atau lambda yang akan memanggil constructor dari sebuah class, atau bisa juga didefine terlebih dahulu cara pembuatannya didalam fungsi lambda tersebut lalu kembalikan hasil instansiasinya. Kemudian concrete class dari creator diubah menjadi sebuah map yang memiliki key yaitu apa yang ingin kita instansiasi (perlu diingat bahwa keynya bisa apapun sehingga tetap flexibel, misalnya untuk factory pattern biasa terdapat parameter untuk instansiasi, maka dalam pattern ini, keynya bisa berupa class wrapper untuk parameter tersebut), dan value yaitu lambda function tadi. Dengan cara ini kita dapat mengurangi code complex karena banyaknya concrete creator class yang harus dibuat untuk mendefine bagaimana masing-masing object diinstansiasi. Pattern ini juga sama flexibelnya dengan factory pattern biasa karena kita dapat dengan bebas menentukan bagaimana menginstansiasi class yang diinginkan di dalam fungsi lambda ini. Keuntungan dari factory pattern dengan lambda dibanding factory pattern adalah, tidak perlu membuat banyak concrete class yang sebenarnya isinya mirip semua sehingga redundant dan kebanyakan boilerplate code atau code duplikat, tidak perlu bolak-balik membaca creator class karena cara pembuatan nya sudah ada dalam fungsi lambda tersebut. Kelemahan dari pattern ini adalah jika cara pembuatan object sangat complex sehingga isi dari lambda function sangat panjang. Namun dari beberapa pendapat justru menganggap pattern ini lebih flexibel karena kalau memang

proses creation object tersebut sangat complex, maka kemungkinan proses complex tersebut sudah ada sebagai beberapa fungsi statik didalam class yang ingin diinstansiasi sehingga tinggal digunakan. Atau bisa juga justru factory pattern tidak tepat untuk digunakan untuk menginstansiasi object tersebut. Dapat dilihat pada contoh implementasi di bawah bahwa map berupa key yaitu string dan valuenya adalah constructor dari kelas yang diinginkan.

```
class ResourceFactory : map<string, function<Resource*()>>{
   ...
    public:

        /**
         * Load the content of the file in the configPath to the translator
        */
        ResourceFactory(string configPath);

        ~ResourceFactory();

        /**
         * Translate the key to the Resource instance
        */
        Resource* translate(string key);
    ...
```

```
ResourceFactory::ResourceFactory(string configPath){
    ...
    ifstream file;
```

```
...
file.open(productPath); if(!file.is_open()) throw FileNotFoundException(productPath);
...
    insert({name, [=](){return new ProductMaterial(kode, name, addedWeight, price);}});
...
    insert({name, [=](){return new Omnivore(kode, name, price, weightToHarvest, dropsMapValue);}});
...
```

3.1.1.20. Visitor Pattern

Visitor pattern merupakan pola desain dalam pemrograman yang dapat digunakan untuk menentukan perilaku dari sebuah method berdasarkan type dari object yang diberikan dengan hanya memberikan object sebagai parent type dari type object tersebut. Dengan pattern ini kita tidak perlu melakukan banyak sekali dynamic_cast, karena melalui inheritance dan function overload, kita dapat mentukan method mana yang akan dieksekusi. Pattern ini memanfaatkan konsep inheritance dan function overload. Umumnya akan terdapat sebuah interface visitor dan element. Visitor akan melakukan visit ke element, dan element akan mengaccept visitor tersebut dan memilihkan method mana di visitor yang harus dijalankan. Jadi misalnya di Visitor terdapat beberapa method yang dioverload dengan type yg berbeda beda, misal diberi nama visit. Jadi misalnya ada visit(typeA), visit(typeB), visit(typeC). Masing-masing type tadi adalah child class dari element. Lalu ada satu method yang type parameternya adalah element, misal visit(element), yang isinya hanyalah memanggil element.accept(this). Element akan memiliki satu method dengan parameter visitor misalnya accept yang isinya hanyalah visitor.visit(this). Maka di sini element seakan-akan memilihkan method mana dari visitor yang harus dipanggil. Misal jika rupanya type dari element yang dipass ke visit(element) adalah typeB, maka yang jalan ada lah fungsi visit(typeB).

Pattern ini sangat berguna untuk masalah double dispatch, yaitu proses memilih method mana yang ingin dipilih berdasarkan child class dari object yang diberikan sebagai argument. Biasanya pattern ini akan mudah dimengerti untuk sebuah kasus yang tidak mudah menentukan siapa yang memiliki tugasnya. Misalnya makan dan dimakan. Apakah kita membuat fungsi makan di animal, atau fungsi dimakan di product. Bisa juga dengan execute dan

executed, visit dan visited, dan sebagainya. Contoh kasusnya adalah untuk memilih type mana yang bisa dimakan oleh hewan, misalnya herbivora tidak bisa makan daging, tapi karnivora. Bisa juga untuk memilih bagaimana perilaku sebuah method jika kita masukkan tipe player tertentu. Misalnya petani tidak bisa membangun tapi walikota bisa.

Dapat dilihat pada implementasi dibawah misalnya untuk hewan, kita dapat memilih tipe produk mana yang bisa dimakan oleh hewan, mana yang tidak bisa dimakan oleh hewan dengan hanya perlu memasukkan type Product, tidak perlu menggunakan dynamic_cast untuk menentukan apakah sebenarnya typenya adalah ProductAnimal, ProductFruit, atau ProductMaterial. Untuk command, bisa juga dilihat pada CLIGame::run bahwa kita hanya perlu memasukkan type Player ke dalam Command::execute. Kita tidak perlu menggunakan dynamic_cast untuk mengetahui apakah Player itu adalah Petani, Peternak, atau Walikota. Dengan pattern visitor, method yang diinginkan akan otomatis terpilih. Pembuatan command baru juga sangat mudah. Kita bisa dengan mudah membuat class command yang baru dengan menginherit class command, lalu pilih apakah ingin mengoveride player saja sehingga otomatis tidak apapun typenya maka akan memiliki perilaku yang sama, bisa juga override satu-satu dan masing-masing memiliki perilaku yang berbeda.

Sebagai tambahan, dapat dilihat pada class Command bahwa ada 3 method yang implementasi nya kosong, tapi bukan pure virtual. Hal ini memang sengaja dilakukan agar child classnya tidak diwajibkan untuk mengimplementasikan 3 method tersebut jika tidak perlu peduli type dari Player. Misalnya ada pada cetak penyimpanan. Pada cetak penyimpanan, kasus untuk type playernya walikota, peternak, atau petani tidak dioverride karena tidak perlu peduli apa type playernya. Jika dibuat pure virtual, maka cetak penyimpanan harus mengimplementasi 3 fungsi tadi dan akan redundant karena ketiganya memanggil sebuah fungsi yang sama.

```
class EaterVisitor
{
    public:
        void eat(EatenElement);
        virtual void eat(ProductAnimal&) = 0;
```

```
        virtual void eat(ProductFruit&) = 0;
        virtual void eat(ProductMaterial&) = 0;
};

class EatenElement
{
    public:
        virtual void eaten(EaterVisitor &v) = 0;
};
```

```
class Product : public Resource, public EatenElement
{
    ...
        virtual void eaten(EaterVisitor &a) = 0;
    ...
```

```
void ProductFruit::eaten(EaterVisitor &a)
{
    a.eat(*this);
}
void ProductMaterial::eaten(EaterVisitor &a)
{
    a.eat(*this);
}
void ProductAnimal::eaten(EaterVisitor &a)
{
    a.eat(*this);
}
```

```
class Animal: public Creature, public EaterVisitor
{
    ...
    void eat(EatenElement&);
```

```
class Carnivore: public Animal {
    public:
        Carnivore(string code, string name, int price, int weightToHarvest, vector<Product*> drops);
        void eat(ProductAnimal&);
        void eat(ProductFruit&);
        void eat(ProductMaterial&);
};
```

```
void Carnivore::eat(ProductAnimal &p)
{
    weight += p.getAddedWeight();
}
void Carnivore::eat(ProductFruit &p)
{
    throw CannotEatException(*this, p);
}
void Carnivore::eat(ProductMaterial &p)
{
    throw CannotEatException(*this, p);
```

```
void Omnivore::eat(ProductAnimal &p)
{
    weight += p.getAddedWeight();
}
void Omnivore::eat(ProductFruit &p)
```

```
{
    weight += p.getAddedWeight();
}
void Omnivore::eat(ProductMaterial &p)
{
    throw CannotEatException(*this, p);
}
```

```
class CommanderVisitor
{
    public:
        virtual void execute(Player*) = 0; // Command overrides this. If derived classes of
Command overrides this, that means the argument type doesn't matter.
        virtual void execute(Petani*) = 0;        // Derived classes override this.
        virtual void execute(Peternak*) = 0;      // Derived classes override this.
        virtual void execute(Walikota*) = 0;      // Derived classes override this.
};

class PlayerElement
{
    public:
        virtual void executed(CommanderVisitor&) = 0;
};
```

```
class Command : public CommanderVisitor { // visitor pattern
    protected:
        State& state;
```

```
        vector<Location> inputListLocation(const string &line);
    public:
        Command(State&);
        int stringToInt(const string& str);
        // visitor pattern
        virtual void execute(Player*);
        virtual void execute(Petani*);
        virtual void execute(Peternak*);
        virtual void execute(Walikota*);

};
```

```
class CetakPenyimpanan: public Command {
    public:
        CetakPenyimpanan(State&);
        void print(Grid<Resource*>&); // Will also be used by other commands
        // visitor pattern
        void execute(Player*);
};
```

```
class Bangun: public Command {
    public:
        Bangun(State&);
        // visitor pattern
        void execute(Petani*);
        void execute(Peternak*);
        void execute(Walikota*);
};
```

```
void CetakPenyimpanan::execute(Player *player) {
```

```
    Grid<Resource*> &inventory = player->getInventory();
    print(inventory);
    cout << "\nTotal slot kosong: " << inventory.getCountNotFilled() << "\n\n";
}
```

```
void Bangun::execute(Petani* petani) {
    throw CommandNotAllowedException("BANGUN");
}
void Bangun::execute(Peternak* peternak) {
    throw CommandNotAllowedException("BANGUN");
}
void Bangun::execute(Walikota* walikota) {
    cout << "Resep bangunan yang ada adalah sebagai berikut." << endl;
    ...
```

```
void CLIGame::run() {

    string command;
    ...
            Command *c = choose(command);
            Player* player = state.getCurrentPlayer();
            c->execute(player);

    ...
}
void CLIGame::initializeCommand() {
    commands["HELP"] = new Help(state);
    commands["NEXT"] = new Next(state);
    commands["CETAK_PENYIMPANAN"] = new CetakPenyimpanan(state);
    commands["PUNGUT_PAJAK"] = new PungutPajak(state);
    commands["CETAK_LADANG"] = new CetakLadang(state);
```

```
    ...
}
```

3.1.1.21. Observer Pattern

Observer pattern merupakan pola desain dalam pemrograman yang dapat digunakan agar kita dapat melakukan sebuah pekerjaan tertentu saat sebuah class sedang melakukan tugas tertentu (terjadi sebuah event), tanpa perlu membuat class tersebut tau tentang pekerjaan tertentu tersebut. Class tersebut hanya perlu memberikan notifikasi kepada semua yang mengsubscribe ke sebuah event pada class tersebut. pekerjaan tersebut juga tidak harus hanya satu, bisa saja ada banyak pekerjaan yang semuanya berbeda beda, dan dilakukan disaat class tadi memberikan notifikasi. Pola desain ini berguna untuk implementasi GUI misalnya pada button, GridView (grid berisi button, dapat mengeluarkan informasi kordinat button saat notifikasi klik), Choice Dialog, dan lain-lain. Misalnya button sedang diklik, button tidak perlu tau apa yang terjadi saat diklik. Tapi object-object lain yang akan subscribe ke event click dari button dan mendefine apa yang akan dilakukan saat klik terjadi. Pada c++ pattern ini dapat diimplementasikan menggunakan std::functional, lalu object di luar akan mengisikan variabel fungsi tersebut dengan fungsi yang diinginkan. Namun karena framework GUI yang digunakan adalah qt5, terdapat fitur signal tersendiri yang dapat digunakan agar aplikasi lebih stabil.

```
class GridSignal : public QWidget {
    Q_OBJECT
signals:
    void cellClicked(Location location);
};
```

```
template<class T>
class GridView : public GridSignal {
private:
```

```
    Grid<T>* grid;
    ...
public:
    GridView();
    ...
};
```

```
class ChoiceDialog : public QDialog {
    Q_OBJECT
private:
    ...
public:
    ChoiceDialog(QWidget* parent, const QVector<pair<string, string>>& choices, const QString& title,
    ...
signals:
    void choiceMade(int index);
    ...
};
```

3.1.1.22. Strategy Pattern

Strategy pattern merupakan pola desain dalam pemrograman untuk mengubah perilaku sebuah object dengan membungkusnya pada strategi tertentu tanpa perlu memanggil fungsi tertentu secara langsung. Pola desain ini akan berguna jika kita memiliki banyak cara untuk melakukan sebuah tugas yang sama, menghasilkan sebuah output berbeda, tapi dengan dependensi yang sama. Jadi dependensi tersebut akan terpisah dari strategi dan tidak perlu tau tentang strategi tersebut. Dengan strategi ini, kita dapat dengan mudah mengganti strategi yang diinginkan dengan hanya menukar class. Contoh penggunaannya adalah pada class GridDrawerCLI<T>. Class ini merupakan child class dari class GridDrawer<T> yang memiliki dependensi ke Grid<T> dan dapat melakukan

fungsi draw. Dengan cara ini, jika nanti waktu kita ingin mengubah strategi menggambar yang baru, kita hanya perlu membuat class baru, inherit parent class, lalu implementasikan fungsi draw nya. Grid<T> sendiri juga merupakan dependensi yang digunakan tanpa perlu tau class Drawer nya sehingga dapat digunakan untuk hal lain.

```
template <class T>
class GridDrawer {
    protected:
        Grid<T> &grid;
    public:
        GridDrawer(Grid<T> &grid);
        virtual void draw() = 0;
};

template <class T>
class GridDrawerCLI: public GridDrawer<T> {
    private:
        void drawRowLine();
        void drawContents(int row);
    public:
        GridDrawerCLI(Grid<T> &grid);
        void draw();
};
```

3.2. Bonus Kreasi Mandiri

3.2.1. ASCII Art

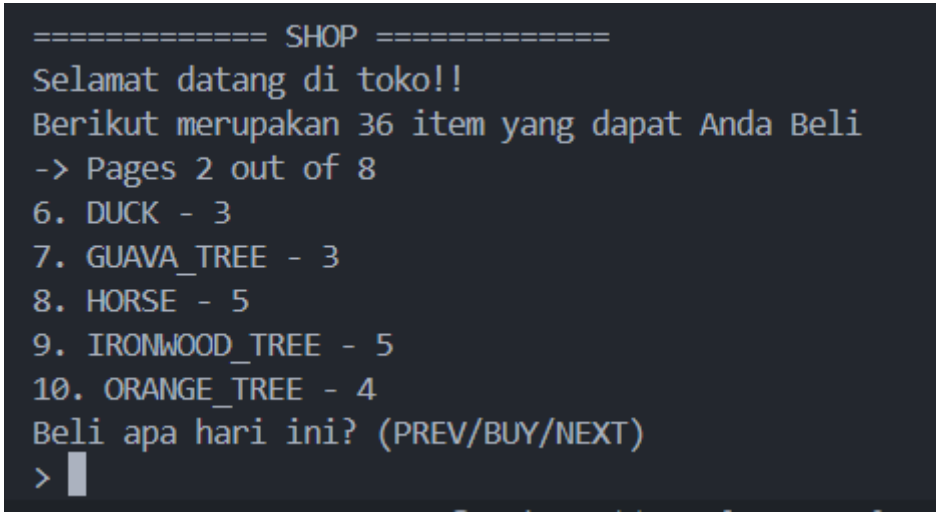
ASCII Art ditampilkan saat permainan dimulai. Saat di awal-awal permainan, akan ditampilkan logo domba.

3.2.2. Pagination

Pada toko, terdapat sebanyak 36 total barang yang dapat dijual. Kami menganggap bahwa apabila kita menampilkan semuanya di dalam satu layar akan mengotori *command line interface* sehingga dapat mengganggu pengalam dari pengguna. Kami memutus menggunakan *Pagination* dengan setiap page-nya hanya menampilkan sebanyak maksimum lima item saja.

```
============ SHOP ============
Selamat datang di toko!!
Berikut merupakan 36 item yang dapat Anda Beli
-> Pages 1 out of 8
1. ALOE_TREE - 6
2. APPLE_TREE - 4
3. BANANA_TREE - 3
4. CHICKEN - 3
5. COW - 6
Beli apa hari ini? (PREV/BUY/NEXT)
>
```

```
============ SHOP ============
Selamat datang di toko!!
Berikut merupakan 36 item yang dapat Anda Beli
-> Pages 8 out of 8
36. TEAK_WOOD - 9 (0)
Beli apa hari ini? (PREV/BUY/NEXT)
>
```



beli.cpp

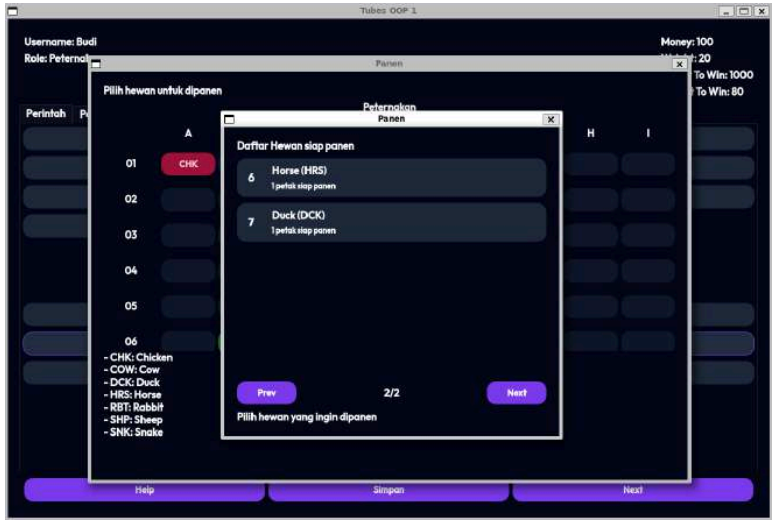
```
...
while (true){
            cout<<"========== SHOP ==========\n";
            cout << "Selamat datang di toko!!" << endl;
            cout << "Berikut merupakan "<<stock.size()<<" item yang dapat Anda Beli" << endl;
            cout<<"-> Pages "<<currentPos+1<<" out of "<<pages<<"\n";
            for(int i=0; i<pageSize(stock,currentPos);i++){
                int dispIdx = i + (currentPos*5) + 1;
                Quantifiable<Resource*> rsc = stock[dispIdx - 1].first;
                string productName = rsc.getValue()->getName();
                int productPrice = rsc.getValue()->getPrice();
                int quantity = rsc.getQuantity();
```

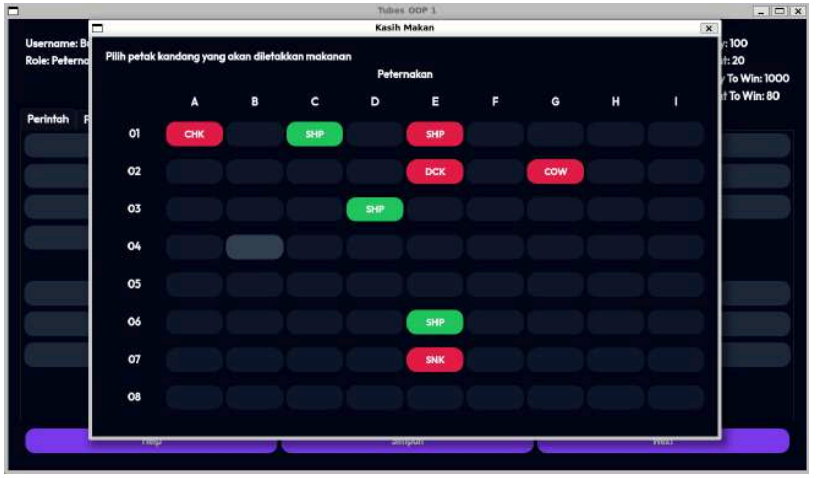
```
            cout<<dispIdx<<". ";
            cout<<productName;
            cout<<" - ";
            cout<< productPrice;

            // Check if quantity is unlimited or not
            if(!Quantifiable<Resource*>::isInfinite(rsc)){
                cout<< " ("<<quantity<<")\n";
            }
            else {
                cout<<endl;
            }
        }

        cout<<"Beli apa hari ini? (PREV/BUY/NEXT)\n";
        cout<<"> ";
        while(!(cin >> choice)){
            cout << "Invalid input type, try again\n";
            cout<<"> ";
        }

        transform(choice.begin(),choice.end(),choice.begin(),::toupper);

        if(choice == "BUY"){
            break;
        }
        else if(choice == "NEXT"){
            currentPos = (currentPos + 1) % pages;
            cout<<endl;

        }
```

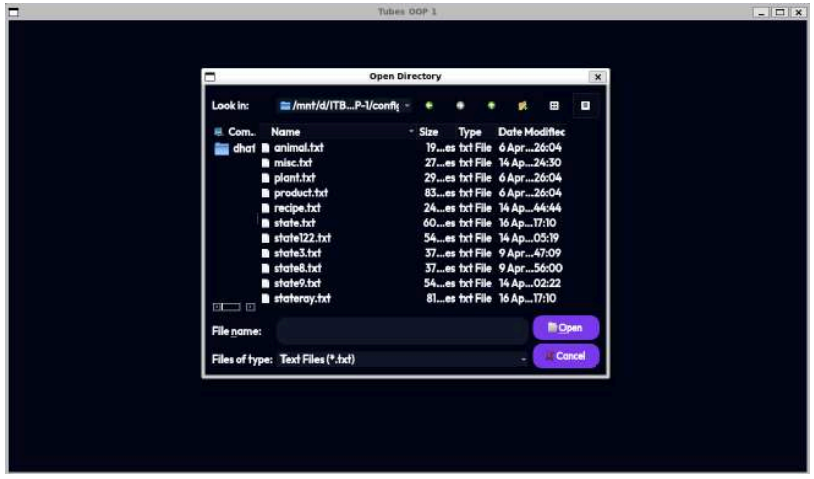
```
            else if(choice == "PREV"){
                currentPos = ((currentPos - 1) + pages) % pages;
                cout<<endl;
            }
            else{
                cout<< "Input tidak valid, silahkan ulangi!";
            }
...
```

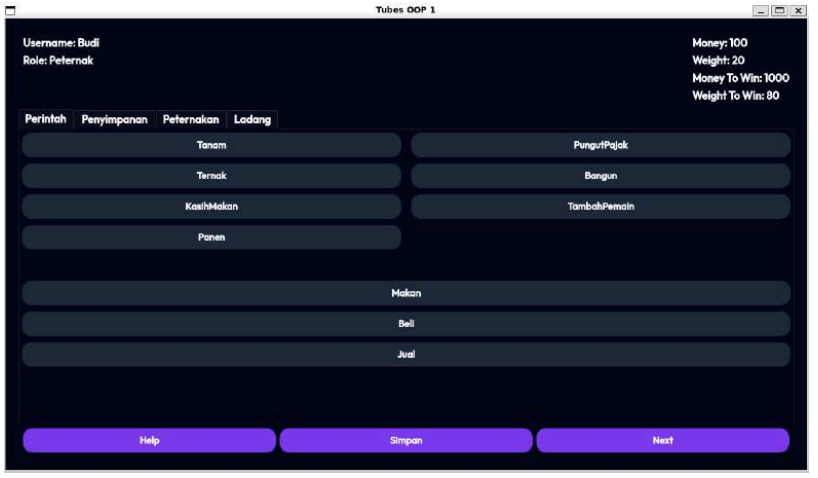
3.2.3. GUI Custom Input

Input pada GUI dapat dilakukan dengan lebih mudah karena tidak perlu dengan mengetik. Misalnya untuk pemilihan command hanya perlu mengklik command yang diinginkan. Untuk input path file saat simpan dan muat ada pop up untuk menentukan pathnya. Untuk Pemilihan item banyak seperti pada shop dan panen ada pagination. Untuk memilih lokasi pada grid tidak perlu mengetik lokasi di grid, hanya perlu mengklik lokasinya. Saat memilih mana yg ingin dipanen bisa memilih multiple button untuk dipanen.


Tubes OOP 1
Username: Budi
Role: Peternak
Money: 100
Weight: 20
Money To Win: 1000
Weight To Win: 80
Panen
Pilih hewan untuk dipanen
Peternakan
Panen
Daftar Hewan siap panen
6
Horse (HRS)
1 petak siap panen
7
Duck (DCK)
1 petak siap panen
Prev
2/2
Next
Pilih hewan yang ingin dipanen
- CHK: Chicken
- COW: Cow
- DCK: Duck
- HRS: Horse
- RBT: Rabbit
- SHP: Sheep
- SNK: Snake
Help
Simpan
Next


Tubes OOP 1
Kasih Makan
Pilih petak kandang yang akan diletakkan makanan
Peternakan
CHK
SHP
SHP
DCK
COW
SHP
SHP
SNK


Tubes OOP 1
Open Directory
Look in: /mnt/d/ITB...P-1/config
animal.txt
misc.txt
plant.txt
product.txt
recipe.txt
state.txt
state122.txt
state3.txt
state8.txt
state9.txt
stateray.txt
File name:
Open
Files of type: Text Files (*.txt)
Cancel


Tubes OOP 1
Username: Budi
Role: Peternak
Money: 100
Weight: 20
Money To Win: 1000
Weight To Win: 80
Perintah
Penyimpanan
Peternakan
Ladang
Tanam
Ternak
KasihMakan
Panen
PungutPajak
Bangun
TambahPemain
Makan
Beli
Jual
Help
Simpan
Next

4. Pembagian Tugas

Modul (dalam poin spek)	Implementer	Tester
Next	13522014, 13522057, 13522066, 13522084, 13522095	13522014, 13522057, 13522066, 13522084, 13522095
Cetak Penyimpanan	13522014, 13522057, 13522066, 13522084, 13522095 13522014, 13522057, 13522066, 13522084, 13522095	13522014, 13522057, 13522066, 13522084, 13522095
Pungut Pajak	13522014, 13522057, 13522066, 13522084, 13522095	13522014, 13522057, 13522066, 13522084, 13522095
Cetak Ladang dan Peternakan	13522014, 13522057, 13522066, 13522084, 13522095	13522014, 13522057, 13522066, 13522084, 13522095
Tanam	13522014, 13522057, 13522066, 13522084, 13522095	13522014, 13522057, 13522066, 13522084, 13522095
Ternak	13522014, 13522057, 13522066, 13522084, 13522095	13522014, 13522057, 13522066, 13522084, 13522095
Bangun Bangunan	13522014, 13522057, 13522066, 13522084, 13522095	13522014, 13522057, 13522066, 13522084, 13522095
Makan	13522014, 13522057, 13522066, 13522084, 13522095	13522014, 13522057, 13522066, 13522084, 13522095

Memberi Pangan	13522014, 13522057, 13522066, 13522084, 13522095	13522014, 13522057, 13522066, 13522084, 13522095
Membeli	13522014, 13522057, 13522066, 13522084, 13522095	13522014, 13522057, 13522066, 13522084, 13522095
Menjual	13522014, 13522057, 13522066, 13522084, 13522095	13522014, 13522057, 13522066, 13522084, 13522095
Memanen	13522014, 13522057, 13522066, 13522084, 13522095	13522014, 13522057, 13522066, 13522084, 13522095
Muat	13522014, 13522057, 13522066, 13522084, 13522095	13522014, 13522057, 13522066, 13522084, 13522095
Simpan	13522014, 13522057, 13522066, 13522084, 13522095	13522014, 13522057, 13522066, 13522084, 13522095
Tambah pemain	13522014, 13522057, 13522066, 13522084, 13522095	13522014, 13522057, 13522066, 13522084, 13522095
Bonus 1: Diagram Non UML	13522057, 13522095	13522014, 13522057, 13522066, 13522084, 13522095
Bonus 2: GUI	13522084	13522014, 13522057, 13522066, 13522084, 13522095
Bonus 3: Unit Testing	13522014	13522014, 13522057, 13522066, 13522084, 13522095

Bonus 4: Design Pattern	13522084	13522014, 13522057, 13522066, 13522084, 13522095
Extra Bonus 1 : ASCII Art	13522066	13522014, 13522057, 13522066, 13522084, 13522095
Extra Bonus 2: Pagination	13522014	13522014, 13522057, 13522066, 13522084, 13522095
Extra Bonus 3: GUI Custom Input	13522084	13522014, 13522057, 13522066, 13522084, 13522095

5. Lampiran

Diagram: Link Diagram Keseluruhan

Github: Link Github Repositori

Laporan asistensi: Link Asistensi