BAB 4
IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Sistem

Implementasi sistem adalah tahap pengujian kelayakan pada sistem yang dibuat sehingga aplikasi yang dibuat tidak menyimpang dari perancangan sistemnya.

4.1.1 Perangkat Pendukung yang Digunakan

Dalam proses pembuatan aplikasi ini, tentunya membutuhkan perangkat keras (*hardware*) dan perangkat lunak (*software*). Berikut merupakan penjelasan dari perangkat keras dan perangkat lunak yang digunakan.

4.1.1.1 Perangkat Keras (*Hardware*)

Perangkat keras yang digunakan selama pengujian aplikasi ini memiliki spesifikasi sebagai berikut :

1. Prosesor : 1.80 GHz
2. Memory : 2048MB RAM
3. VGA : 512MB
4. Hardisk : 500GB
5. Monitor
6. Mouse dan Keyboard.

4.1.1.2 Perangkat Lunak (*Software*)

Perangkat lunak yang digunakan selama pengujian aplikasi ini memiliki spesifikasi sebagai berikut :

a. Sistem Operasi Windows 7
b. JDK 6

4.1.2 Implementasi Aplikasi

Pada Implementasi aplikasi diperlihatkan beberapa tampilan dari aplikasi yang dibangun.

4.1.2.1 Implementasi Antarmuka

Adapun implementasi antarmuka pada aplikasi *game Farming Life* adalah sebagai berikut :

a. Tampilan Menu Utama



Gambar 4.1 Tampilan Menu Utama

Halaman ini menampilkan menu utama yang memiliki 3 pilihan berbeda yaitu MULAI untuk memulai permainna baru, LANJUT untuk melanjutkan permainan sebelumnnya dan KELUAR untuk keluar dari aplikasi.

b. Tampilan Intro



Gambar 4.2 Tampilan Intro

Halaman ini menampilkan intro game dan kotak nama sebagai kotak untuk nama karakter yang akan digunakan dalam permainan.

c. Tampilan Petunjuk Permainan



Gambar 4.3 Tampilan Petunjuk Permainan

Halaman ini menampilkan Petunjuk Permainan pada awal permainan.

d. Tampilan Petunjuk Menanam



Gambar 4.4 Tampilan Petunjuk Menanam 1

Halaman ini menampilkan petunjuk menanam bagian 1



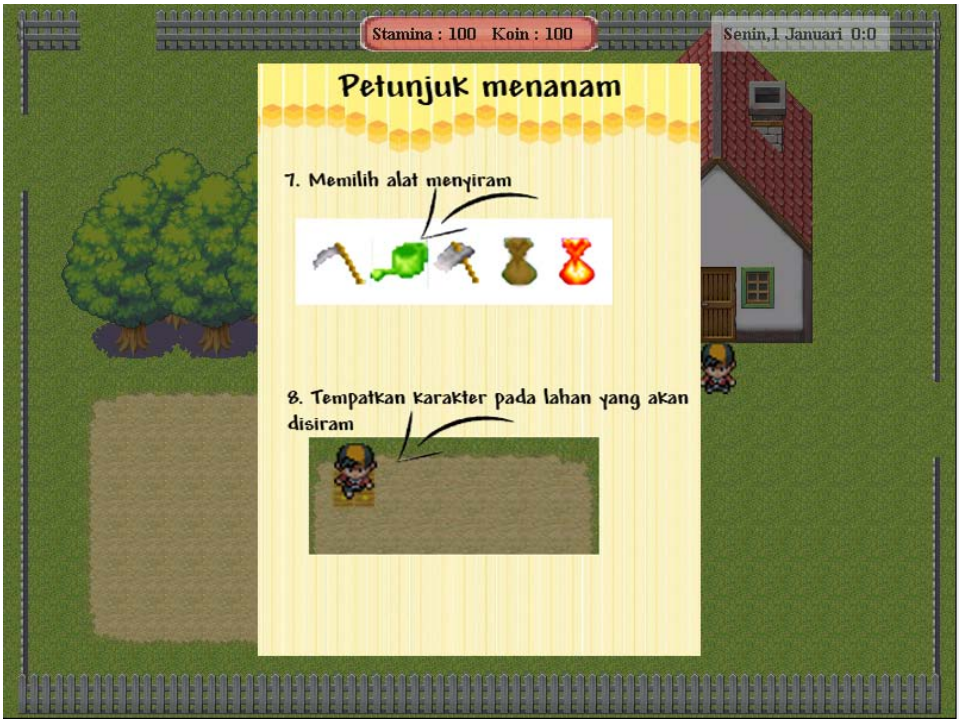
Gambar 4.5 Tampilan Petunjuk Menanam 2

Halaman ini menampilkan petunjuk menanam bagian 2



Gambar 4.6 Tampilan Petunjuk Menanam 3

Halaman ini menampilkan petunjuk menanam bagian 3



Gambar 4.7 Tampilan Petunjuk Menanam 4

Halaman ini menampilkan petunjuk menanam bagian 4



Gambar 4.8 Tampilan Petunjuk Menanam 5

Halaman ini menampilkan petunjuk menanam bagian 5

e. Tampilan Menu Permainan



Gambar 4.9 Tampilan Menu Permainan

Halaman ini menampilkan menu permainan ketika berada pada Arena Permainan. Menu Permainan memiliki 3 pilihan yaitu, status permainan, simpan permainan dan keluar permainan.

f. Tampilan Arena Permainan



Gambar 4.10 Tampilan Arena Permainan map Ladang

Halaman ini menampilkan Arena Permainan map Ladang.



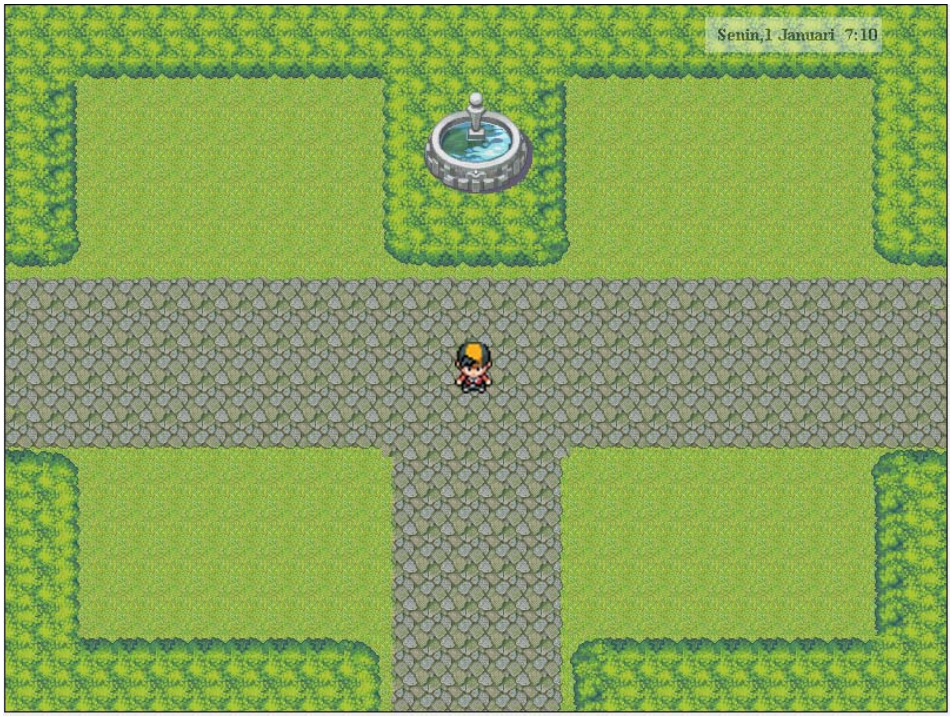
Gambar 4.11 Tampilan Arena Permainan map Pemukiman01

Halaman ini menampilkan Arena Permainan map Ladang.



Gambar 4.12 Tampilan Arena Permainan map Pemukiman02

Halaman ini menampilkan Arena Permainan map Ladang.



Gambar 4.13 Tampilan Arena Permainan map Alun - Alun

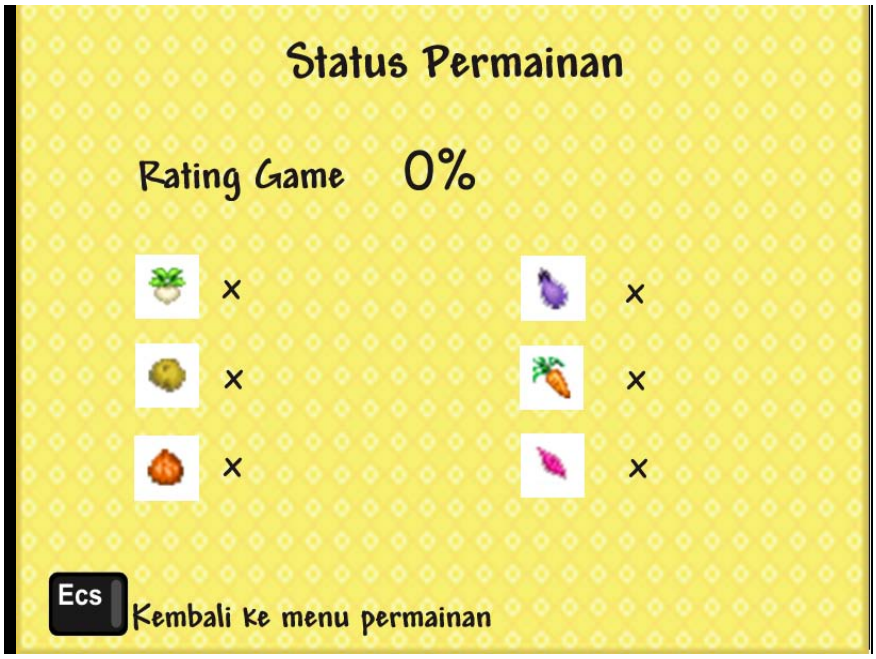
Halaman ini menampilkan Arena Permainan map Ladang.



Gambar 4.14 Tampilan Arena Permainan Pantai

Halaman ini menampilkan Arena Permainan map Ladang.

g. Tampilan Status Permainan



Gambar 4.15 Tampilan Status Permainan

Halaman ini menampilkan status permainan yang sedang berjalan dan menampilkan misi permainan yang telah diselesaikan.

h. Tampilan Simpan Data



Gambar 4.16 Tampilan Simpan Data

Halaman pada ini menampilkan Slot Data yang dapat dipilih untuk menyimpan status permainan yang sedang dijalankan.

i. Tampilan Keluar Permainan



Gambar 4.17 Tampilan Keluar Permainan

Halaman ini menampilkan sebuah pesan yang akan muncul ketika memlik keluar permainan.

j. Tampilan Lanjutkan Permainan



Gambar 4.18 Tampilan Lanjutkan Permainan

Halaman ini menampilkan Slot Data yang dapat dipiilih untuk melanjutkan permainan menggunakan status permainan yang telah disimpan sebelumnya.

k. Tampilan Keluar Aplikasi



Gambar 4.19 Tampilan Keluar Aplikasi

Halaman ini menampilkan Pesan yang akan muncul ketika memilih Keluar Aplikasi.

4.2 Pengujian Sistem

Pengujian sistem adalah bagian penelusuran dalam rangka menemukan kesalahan pada sistem yang dibangun. Tujuan pengujian sistem adalah untuk mencari kesalahan pada sistem yang dibangun sebelum sistem digunakan oleh pengguna.

4.2.1 Rencana Pengujian

Rencana pengujian perangkat lunak ini dibagi menjadi 2 bagian yaitu pengujian *black box* dan pengujian *white box.*

Pengujian *white box* dilakukan untuk menguji kesalahan logik dan asumsi yang tidak tepat pada kemungkinan eksekusi. Pengembangan kode-kode program selalu memungkinkan terjadinya alur program yang tidak dan tereksekusi dan kesalahan *typography* yang sulit ditemukan kalau tidak dijalankan. Pada umumnya pengujian ini dilakukan hanya pada lingkungan developer.

Pengujian *black box* dilakukan untuk mencari fungsi-fungsi program yang tidak benar atau hilang. Kesalahan lain yang mungkin terjadi dalam pembuatan

program adalah kesalahan dalam struktur data, kesalahan antar muka, inisialisasi dan akhir program dan kesalahan performansi.

Strategi pengujian akan dilakukan dengan menggunakan pengujian *Beta*. Pengujian *beta* adalah pengujian pada lingkungan pengguna. Pengujian ini dilakukan dengan menggunakan metode pengumpulan data melalui kuesioner. Pengguna akan mencoba game ini, kemudian pengguna akan memberikan *feedback* pada kuesioner yang telah disediakan . Deskripsi rencana pengujian dapat dilihat dalam Tabel 4.1.

Tabel 4.1 Rencana Pengujian

No.	Komponen yang diuji	Detail Pengujian	Jenis Uji
1.	Menu Utama	Memilih Tombol Mulai Permainan	*Black Box*
		Memilih Tombol Lanjutkan Permainan	*Black Box*
		Memilih Tombol Keluar Permainan	*Black Box*
2.	Kotak Nama	Memasukan Huruf	*Black Box*
		Tekan ENTER	*Black Box*
3.	Intro	Memilih Lewat Intro	*Black Box*
		Setelah gigi bersih	*Black Box*
4.	Arena Permainan	Karakter Bergerak	*Black Box*
		Becocok Tanam	*Black Box*
		Waktu Berjalan	*Black Box*
		Pindah Lokasi	*Black Box*
		Karakter Berbicara dengan NPC	*Black Box*
		Rating Game	*Black Box*
		Memilih Menu Permainan	*Black Box*
5.	Menu Permainan	Memilih Status Permainan	*Black Box*
		Memilih Simpan Data	*Black Box*
		Waktu Berhenti	*Black Box*
		Memilih Keluar Permainan	*Black Box*
6.	Status	Menampilkan Misi Permainan	*Black Box*

	Permainan	Dapat jajanan jelek	*Black Box*
		Karakter menabrak batu	*Black Box*
7.	Simpan Data Permainan	Menampilkan Slot Data Permainan	*Black Box*
		Memilih Slot Data Permainan	*Black Box*
8.	Keluar Permainan	Memilih Pesan Ya	*Black Box*
		Memilih Pesan Tidak	*Black Box*
9.	Lanjutkan Permainan	Menampilkan Slot Data Permainan	*Black Box*
		Memilih Slot Data Permainan	*Black Box*
10.	Keluar Aplikasi	Memilih Peasn Ya	*Black Box*
		Memilih Pesan Tidak	*Black Box*
11.	Algoritma Koloni Semut	Algoritma Koloni Semut	*White Box*

4.2.2 Kasus dan Hasil Pengujian (Black-Box)

Kasus dan hasil uji perangkat lunak yang sudah dibangun dengan menggunakan metode *black box* berdasarkan pengamatan yang dilakukan dan menghasilkan kesimpulan dari pengujian tersebut. Kasus dan Hasil Pengujian dapat dilihat pada tabel 4.2.

Tabel 4.2 Kasus dan Hasil Uji

Kasus dan hasil uji (data normal)				
No.	**Kasus /diuji**	**Skenario Uji**	**Hasil yang diharapkan**	**Hasil Pengujian**
1.	Menu Utama	Memilih Tombol Mulai Permainan	Menampilkan halaman Kotak Nama dan Virtal Keyboard	[√] berhasil [ ] gagal
		Memilih Tombol Lanjutkan Permainan	Menampilkan Slot Data Permainan	[√] berhasil [ ] gagal
		Memilih Tombol Keluar Aplikasi	Menampilkan Pesan Keluar Aplikasi	[√] berhasil [ ] gagal

2.	Kotak Nama	Memasukan Huruf	Huruf Tampil pada Kotak Nama	[√] berhasil [ ] gagal
		Tekan ENTER	Masuk ke dalam Intro	[√] berhasil [ ] gagal
3.	Intro	Memilih Lewat Intro	Mempercepat Intro	[√] berhasil [ ] gagal
4.	Arena Permainan	Karakter Bergerak	Karakter Dapat bergerak ke arah kanan, kiri, atas dan bawah	[√] berhasil [ ] gagal
		Becocok Tanam	Mencangkul, menanam, menyiram	[√] berhasil [ ] gagal
		Waktu Berjalan	Hari, Jam, Menit bertambah	[√] berhasil [ ] gagal
		Pindah Lokasi	Ganti Lokasi Arena Permainan	[√] berhasil [ ] gagal
		Karakter Berbicara dengan NPC	Memunculkan Textbox	[√] berhasil [ ] gagal
		Rating Game	Rating game bertambah dari karakter memetik hasil panen	[√] berhasil [ ] gagal
		Memilih Menu Permainan	Menampilkan Menu Permainan	[√] berhasil [ ] gagal
5.	Menu Permainan	Memilih Status Permainan	Menampilkan Status Permainan	[√] berhasil [ ] gagal

		Memilih Simpan Permainan	Menampilkan Slot Data Permainan	[√] berhasil [ ] gagal
		Waktu Berhenti	Hari, Jam, Menit berhenti	[√] berhasil [ ] gagal
		Memilih Keluar Permainan	Menampilkan Pesan Keluar Permainan	[√] berhasil [ ] gagal
6.	Status Permainan	Menampilkan Misi Permainan	Tampil Misi Permainan yang sudah diselesaikan	[√] berhasil [ ] gagal
7.	Simpan Data Permainan	Menampilkan Slot Data Permainan	Tampil Slot Data Permainan	[√] berhasil [ ] gagal
		Memilih Slot Data Permainan	Status Permainan tersimpan	[√] berhasil [ ] gagal
8.	Keluar Permainan	Memilih Pesan Ya	Kembali ke Menu Utama	[√] berhasil [ ] gagal
		Memilih Pesan Tidak	Tetap pada Menu Permainan	[√] berhasil [ ] gagal
9.	Lanjutkan Permainan	Menampilkan Slot Data Permainan	Tampil Slot Data Permainan	[√] berhasil [ ] gagal
		Memilih Slot Data Permainan	Status Permainan dapat dilanjutkan	[√] berhasil [ ] gagal
10.	Keluar Aplikasi	Memilih Peasn Ya	Keluar dari Aplikasi	[√] berhasil [ ] gagal

		Memilih Pesan Tidak	Tetap pada Menu Utama	[√] berhasil [ ] gagal

4.2.3 Kasus dan Pengujian (White-Box)

Pengujian *white box* bertujuan untuk mengetahui kinerja logika yang dibuat pada sebuah perangkat lunak apakah berjalan dengan baik atau tidak. Pengujian *white box* akan digunakan pada algoritma koloni semut. Pengujian *white box* dilakukan dengan beberapa tahap, diantaranya :

1. Pengubahan *Sour Code* ke dalam *flowchart* kemudian menjadi *flowgraph*

Tabel 4.3 *Source Code* Algoritma Koloni Semut

	Class AntHill
	`import greenfoot.*;`
	`public class AntHill extends Actor`
	`{`
	`    static boolean active = false;`
	`    /** Number of ants that have come out so far. */`
1	`    static int ants = 0;`
	`    /** Total number of ants in this hill. */`
	`    private int maxAnts = 40;`
	`    /** Counter to show how much food have been collected so far. */`
	`    private Counter foodCounter;`
	`    /**`
	`     * Constructor for ant hill with default number of ants (40).`
	`     */`
	`    public AntHill()`
	`    {`
	`        setImage("ACO/anthill.png");`
	`    }`
	`    /**`
	`     * Construct an ant hill with a given number of ants.`
	`     */`
	`    public AntHill(int numberOfAnts)`
	`    {`
2	`        maxAnts = numberOfAnts;`
	`    }`
	`    /**`
	`     * Act: If there are still ants left inside, see whether one should come out.`
	`     */`
	`    public void act()`

```
    {
        sebarSemut();
    }

    /**
     * Record that we have collected another bit of food.
     */
    public void countFood()
    {
12      food++;
13      if (food == 40)
        {
14          getWorld().addObject(new  Hama(Field.AH1),  getX(),
getY());
            food = 0;
            if(target == 0)
            {
                active = true;
            }
        }
    }

    public void sebarSemut()
    {
2       if(ants < maxAnts)
        {
            if(Greenfoot.getRandomNumber(100)  <  10  &&  active
== true)
            {
3               getWorld().addObject(new       Ant(Field.AH1),
4   getX(), getY());
5               ants++;
            }
        }
    }
}
```

Class Ant

```
import greenfoot.*;
public class Ant extends Creature
{
    /** Every how many steps can we place a pheromone drop. */
1   private static final int MAX_PH_LEVEL = 18;

    /**  How  long  do  we  keep  direction  after  finding
pheromones. */
    private static final int PH_TIME = 30;

    /** Indicate whether we have any food with us. */
    private boolean carryingFood = false;

    /** How much pheromone do we have right now. */
    private int pheromoneLevel = MAX_PH_LEVEL;

    /** How well do we remember the last pheromone - larger
number: more recent */
    private int foundLastPheromone = 0;
```

```
    /**
     * Create an ant with a given home hill. The initial speed
is zero (not moving).
     */
    public Ant(AntHill home)
    {
        setImage("ACO/ant.gif");
        setHomeHill(home);
    }

    /**
     * Do what an ant's gotta do.
     */
    public void act()
    {
        colLeft();
        colRight();
        colUp();
        colDown();
        checkGetFood());
        removeAnt();
    }

    /**
     * Walk around in search of food.
     */
    private void searchForFood()
    {
        if (foundLastPheromone > 0) { // if we can still
remember...
            foundLastPheromone--;
            walkAwayFromHome();
        }
6       else if (smellPheromone()) {
7           walkTowardsPheromone();
        }
        else {
            randomWalk();
        }
8       checkFood();
    }

    /**
     * Are we home? Drop the food if we are, and start heading
back out.
     */
    private void checkHome()
    {
        if (atHome()) {
12          dropFood();
        }
    }

    /**
     * Are we home?
```

```
     */
    private boolean atHome()
    {
        if (getHomeHill() != null) {
            return (Math.abs(getX() - getHomeHill().getX()) <
4) && (Math.abs(getY() - getHomeHill().getY()) < 4);
        }
        else {
            return false;
        }

    }

    /**
     * Is there any food here where we are? If so, take some!.
     */
    public void checkFood()
    {
        if(Soil01.PEST == true){
            Soil01          soil01          =          (Soil01)
getOneIntersectingObject(Soil01.class);
        }
        if(soil01 != null){
            carryingFood = true;
            setImage("ACO/ant-with-food.gif");
        }
        if(Soil02.PEST == true){
            Soil02          soil02          =          (Soil02)
getOneIntersectingObject(Soil02.class);
        }
        if(soil02 != null){
            carryingFood = true;
            setImage("ACO/ant-with-food.gif");
        }
        if(Soil03.PEST == true){
            Soil03          soil03          =          (Soil03)
getOneIntersectingObject(Soil03.class);
        }
        if(soil03 != null){
            carryingFood = true;
            setImage("ACO/ant-with-food.gif");
        }
        if(Soil04.PEST == true){
            Soil04          soil04          =          (Soil04)
getOneIntersectingObject(Soil04.class);
        }
        if(soil04 != null){
            carryingFood = true;
            setImage("ACO/ant-with-food.gif");
        }
        if(Soil05.PEST == true){
            Soil05          soil05          =          (Soil05)
getOneIntersectingObject(Soil05.class);
        }
        if(soil05 != null){
            carryingFood = true;
```

```
            setImage("ACO/ant-with-food.gif");
        }
        if(Soil06.PEST == true){
            Soil06          soil06          =          (Soil06)
getOneIntersectingObject(Soil06.class);
        }
        if(soil06 != null){
            carryingFood = true;
            setImage("ACO/ant-with-food.gif");
        }
        if(Soil07.PEST == true){
            Soil07          soil07          =          (Soil07)
getOneIntersectingObject(Soil07.class);
        }
        if(soil07 != null){
            carryingFood = true;
            setImage("ACO/ant-with-food.gif");
        }
        if(Soil08.PEST == true){
            Soil08          soil08          =          (Soil08)
getOneIntersectingObject(Soil08.class);
        }
        if(soil08 != null){
            carryingFood = true;
            setImage("ACO/ant-with-food.gif");
        }
        if(Soil09.PEST == true){
            Soil09          soil09          =          (Soil09)
getOneIntersectingObject(Soil09.class);
        }
        if(soil09 != null){
            carryingFood = true;
            setImage("ACO/ant-with-food.gif");
        }
        else if(Soil10.PEST == true){
            Soil10          soil10          =          (Soil10)
getOneIntersectingObject(Soil10.class);
        }
        if(soil10 != null){
            carryingFood = true;
            setImage("ACO/ant-with-food.gif");
        }
        else if(Soil11.PEST == true){
            Soil11          soil11          =          (Soil11)
getOneIntersectingObject(Soil11.class);
        }
        if(soil11 != null){
            carryingFood = true;
            setImage("ACO/ant-with-food.gif");
        }
        else if(Soil12.PEST == true){
            Soil12          soil12          =          (Soil12)
getOneIntersectingObject(Soil12.class);
        }
        if(soil12 != null){
            carryingFood = true;
```

```
            setImage("ACO/ant-with-food.gif");
        }
        else if(Soil13.PEST == true){
            Soil13          soil13          =          (Soil13)
getOneIntersectingObject(Soil13.class);
        }
        if(soil13 != null){
            carryingFood = true;
            setImage("ACO/ant-with-food.gif");
        }
        else if(Soil14.PEST == true){
            Soil14          soil14          =          (Soil14)
getOneIntersectingObject(Soil14.class);
        }
        if(soil14 != null){
            carryingFood = true;
            setImage("ACO/ant-with-food.gif");
        }
        else if(Soil15.PEST == true){
            Soil15          soil15          =          (Soil15)
getOneIntersectingObject(Soil15.class);
        }
        if(soil15 != null){
            carryingFood = true;
            setImage("ACO/ant-with-food.gif");
        }
        else if(Soil16.PEST == true){
            Soil16          soil16          =          (Soil16)
getOneIntersectingObject(Soil16.class);
        }
        if(soil16 != null){
            carryingFood = true;
            setImage("ACO/ant-with-food.gif");
        }
    }

    /**
     * Take some food from a fool pile.
     */
    private void takeFood(Food food)
    {
        carryingFood = true;
        food.takeSome();
        setImage("ACO/ant-with-food.gif");
    }

    /**
     * Drop our food in the ant hill.
     */
    private void dropFood()
    {
        carryingFood = false;
        getHomeHill().countFood();
        setImage("ACO/ant.gif");
    }
```

```
    /**
     * Check whether we can drop some pheromone yet. If we
can, do it.
     */
    private void handlePheromoneDrop()
    {
        if (pheromoneLevel == MAX_PH_LEVEL) {
            Pheromone ph = new Pheromone();
            getWorld().addObject(ph, getX(), getY());
            pheromoneLevel = 0;
        }
        else {
            pheromoneLevel++;
        }
    }

    /**
     * Check whether we can smell pheromones. If we can,
return true, otherwise return false.
     */
    public boolean smellPheromone()
    {
        Actor ph = getOneIntersectingObject(Pheromone.class);
        return (ph != null);
    }

    /**
     * If we can smell some pheromone, walk towards it. If
not, do nothing.
     */
    public void walkTowardsPheromone()
    {
        Actor ph = getOneIntersectingObject(Pheromone.class);
        if (ph != null) {
            headTowards(ph);
            walk();
            if (ph.getX() == getX() && ph.getY() == getY()) {
                foundLastPheromone = PH_TIME;
            }
        }
    }

    public void checkGetFood()
    {
        if (carryingFood) {
            walkTowardsHome();
            handlePheromoneDrop();
            checkHome();
        }
        else {
            searchForFood();
        }
    }

    public void removeAnt()
    {
```

```
        if(AntHill.active == false){

getWorld().removeObjects(getWorld().getObjects(Ant.class));
        }
    }
    public void colRight()
    {
        if(canSee(AntsBlockRight.class))
        {
            setLocation(getX()-10,getY());
        }
    }

    public void colLeft()
    {
        if(canSee(AntsBlockLeft.class))
        {
            setLocation(getX()+10,getY());
        }
    }

    public void colDown()
    {
        if(canSee(AntsBlockDown.class))
        {
            setLocation(getX(),getY()-10);
        }
    }

    public void colUp()
    {
        if(canSee(AntsBlockUp.class))
        {
            setLocation(getX(),getY()+10);
        }
    }
}
```

Class Creature

```
import  greenfoot.*;    //  (World,  Actor,  GreenfootImage,
Greenfoot and MouseInfo)

public class Creature  extends Actor
{
    /** The maximum movement speed of the ant. */
    private static final int SPEED = 3;

    /** Current movement. Defined as the offset in x and y
direction moved in each step. */
    private int deltaX;
    private int deltaY;

    /** The home ant hill. */
    private AntHill home;

    /**
     * Crtae a new creature with neutral movement (movement
```

```
speed is zero).
     */
    public Creature()
    {
        deltaX = 0;
        deltaY = 0;
    }

    /**
     * Set the home hill of this creature.
     */
    public void setHomeHill(AntHill homeHill)
    {
        home = homeHill;
    }

    /**
     * Get the home hill of this creature.
     */
    public AntHill getHomeHill()
    {
        return home;
    }

    /**
     * Walk around randomly (random direction and speed).
     */
    public void randomWalk()
    {
        if (randomChance(50)) {
            deltaX = adjustSpeed(deltaX);
            deltaY = adjustSpeed(deltaY);
        }
        walk();
    }

    /**
     * Try to walk home. Sometimes creatures get distracted or
encounter small obstacles, so
     * they occasionally head in a different direction for a
moment.
     */
    public void walkTowardsHome()
    {
        if(home == null) {
            //if we do not have a home, we can not go there.
            return;
        }
        if (randomChance(2)) {
            randomWalk();  // cannot always walk straight. 2%
chance to turn off course.
        }
        else {
            headRoughlyTowards(home);
            walk();
        }
```

```
    }

    /**
     * Try to walk away from home. (Goes occasionally off
course a little.)
     */
    public void walkAwayFromHome()
    {
        if(home == null) {
            //if we do not have a home, we can not head away
from it.
            return;
        }
        if (randomChance(2)) {
            randomWalk();  // cannot always walk straight. 2%
chance to turn off course.
        }
        else {
            headRoughlyTowards(home);    // first head towards
home...
            deltaX = -deltaX;             // ...then turn 180
degrees
            deltaY = -deltaY;
            walk();
        }
    }

    /**
     * Adjust the walking direction to head towards the given
co-ordinates.
     */
    public void headTowards(Actor target)
    {
        deltaX = capSpeed(target.getX() - getX());
        deltaY = capSpeed(target.getY() - getY());
    }

    /**
     * Walk forward in the current direction with the current
speed.
     * (Does not change direction or speed.)
     */
    public void walk()
    {
        setLocation(getX() + deltaX, getY() + deltaY);
        setRotation((int) (180 * Math.atan2(deltaY, deltaX) /
Math.PI));
    }

    /**
     * Adjust the walking direction to head somewhat towards
the given co-ordinates. This does not
     * always head in the same direction. The heading is
slightly random (but likely to be somewhat
     * towards the target) to make it look more natural.
     */
```

```
    private void headRoughlyTowards(Actor target)
    {
        int distanceX = Math.abs(getX() - target.getX());
        int distanceY = Math.abs(getY() - target.getY());
        boolean    moveX    =    (distanceX    >    0)    &&
(Greenfoot.getRandomNumber(distanceX    +    distanceY)    <
distanceX);
        boolean    moveY    =    (distanceY    >    0)    &&
(Greenfoot.getRandomNumber(distanceX    +    distanceY)    <
distanceY);

        deltaX      =      computeHomeDelta(moveX,      getX(),
target.getX());
        deltaY      =      computeHomeDelta(moveY,      getY(),
target.getY());
    }

    /**
     * Compute and return the direction (delta) that we should
steer in when
     * we're on our way home.
     */
    private  int  computeHomeDelta(boolean  move,  int  current,
int home)
    {
        if (move) {
            if (current > home)
                return -SPEED;
            else
                return SPEED;
        }
        else
            return 0;
    }

    /**
     * Adjust  the  speed  randomly  (start  moving,  continue  or
slow down). The
     * speed returned is in the range [-SPEED .. SPEED].
     */
    private int adjustSpeed(int speed)
    {
        speed = speed + Greenfoot.getRandomNumber(2 * SPEED -
1) - SPEED + 1;
        return capSpeed(speed);
    }

    /**
     * Make sure the speed returned is in the range [-SPEED ..
SPEED].
     */
    private int capSpeed(int speed)
    {
        if (speed < -SPEED)
            return -SPEED;
        else if (speed > SPEED)
```

```
            return SPEED;
        else
            return speed;
    }

    /**
     * Return 'true' in exactly 'percent' number of calls. That is: a call
     * randomChance(25) has a 25% chance to return true.
     */
    private boolean randomChance(int percent)
    {
        return Greenfoot.getRandomNumber(100) < percent;
    }

    public boolean canSee(Class clss)
    {
        Actor actor = getOneObjectAtOffset(0, 0, clss);
        return actor != null;
    }
}
```

Class Pheromone

```
import greenfoot.*;   // (World, Actor, GreenfootImage, and Greenfoot)
import java.awt.Color;

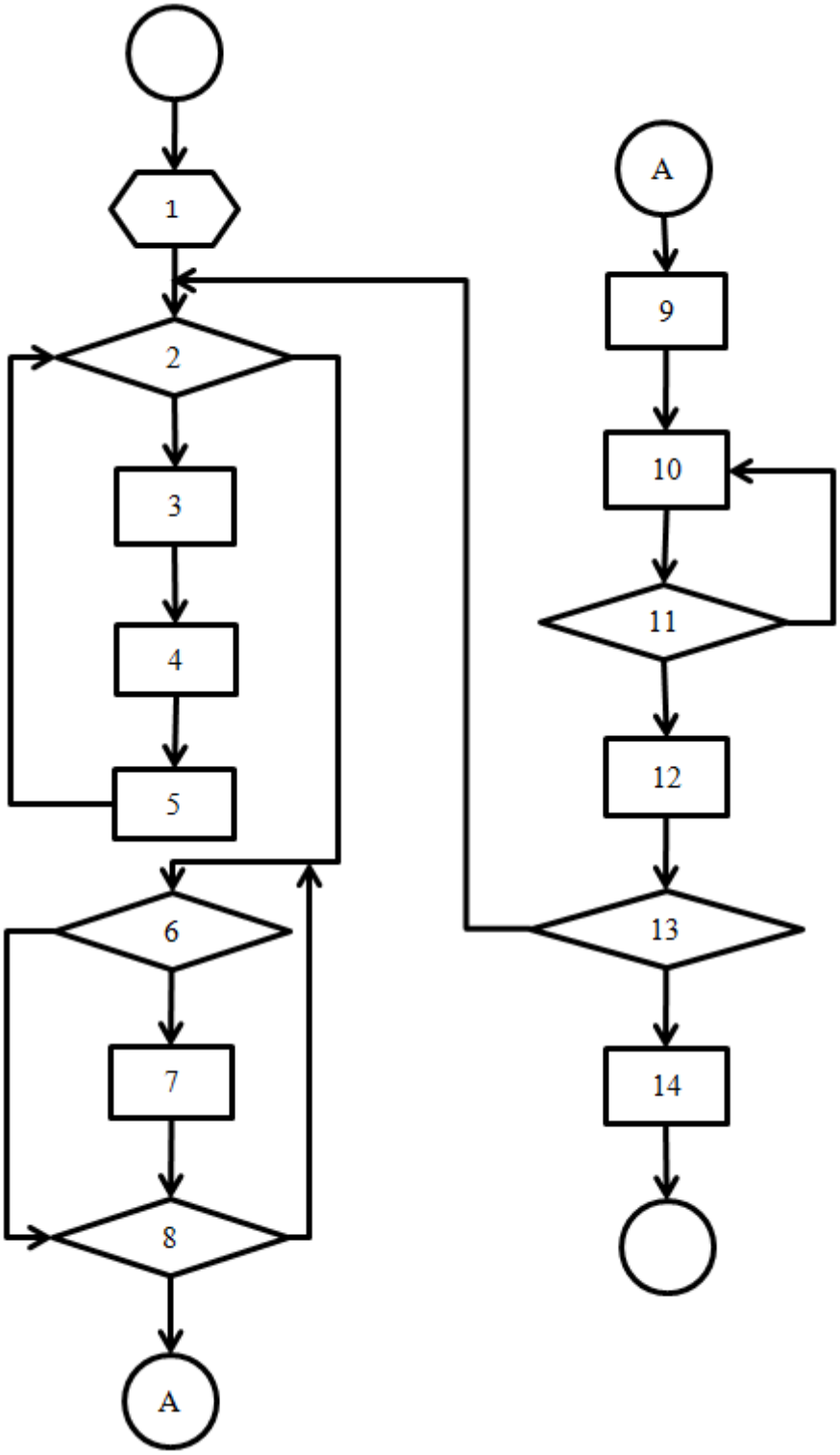
public class Pheromone extends Actor
{
    private final static int MAX_INTENSITY = 180;
    private int intensity;

    /**
     * Create a new drop of pheromone with full intensity.
     */
    public Pheromone()
    {
        intensity = MAX_INTENSITY;
        setImage("ACO/Pheromone.png");
        updateImage();
    }

    /**
     *  The pheromone decreases the intensity. When the intensity reaches zero, it disappears.
     */
    public void act()
    {
        intensity -= 1;
        if (intensity <= 0) {
            getWorld().removeObject(this);
        }
        else {
            if ((intensity % 4) == 0) {       // every four steps...
                updateImage();
            }
```

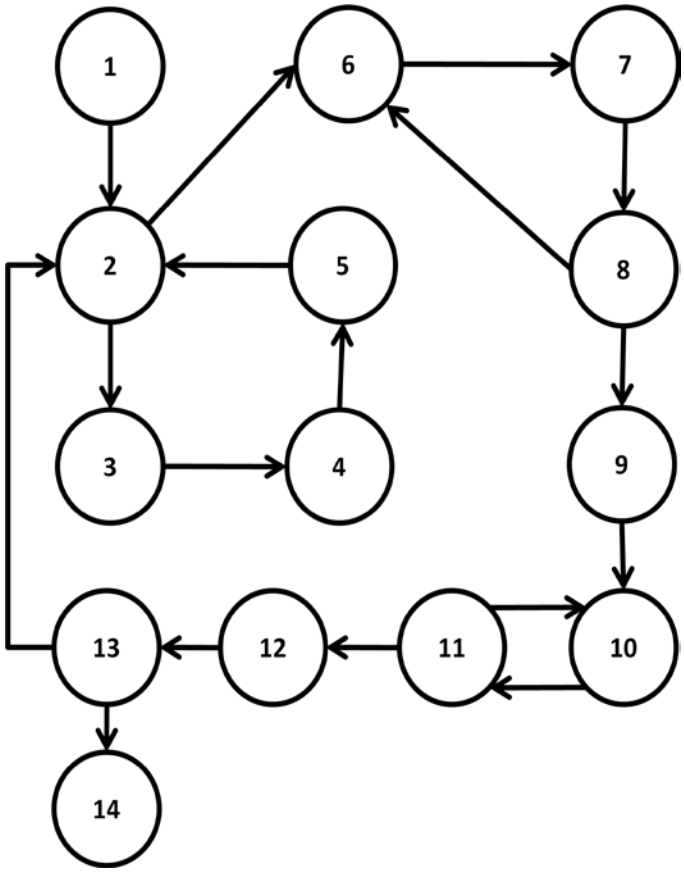
```
        }
    }

    /**
     *  Make  the  image.  The  size  and  transparency  are
proportional to the intensity.
     */
    private void updateImage()
    {
        int size = intensity / 3 + 5;
        GreenfootImage  image  =  new  GreenfootImage(size  +  1,
size + 1);
        int alpha = intensity / 3;
        image.setColor(new Color(255, 255, 255, alpha));
        image.fillOval(0, 0, size, size);
        image.setColor(Color.DARK_GRAY);
        image.fillRect(size / 2, size / 2, 2, 2);   // small
dot in the middle
        image.setTransparency(0);
        setImage(image);
    }
}
```



Gambar 4.20 ***Flowchart*** **Pengujian Algoritma Koloni Semut**

Flowgraph yang merupakan hasil dari perubahan *flowchart* dapat dilihat pada gambar 4.21.



Gambar 4.21 *Flowgraph* Pengujian Algoritma Koloni Semut

Keterangan : = Menggambarkan kondisi

= Menggambarkan aksi

a. Tahap Pengujian Algoritma Koloni Semut

1. *Cyclomatic Complexity* V(G)

   Edge = 17, Node = 14

   V(G) = Edge – Node + 2

   V(G) = 17 – 14 + 2

   V(G) = 3 + 2

   V(G) = 5

2. *Independent Path*

   Path 1 = 1-2-3-4-5-2-6-7-8-9-10-11-12–13-14

   Path 2 = 1-2-6-7-8-9-10-11-12-13-14

   Path 3 = 1-2-3-6-7-8-6-….

   Path 4 = 1-2-6-7-8-9-10-11-10-….

Path 5 = 1-2-6-7-8-9-10-11-12-13-2-….

3. *Predicate Node*

   V(G) = Jumlah node yang memiliki lebih dari 1 jalur + 1

   V(G) = 4 + 1

   V(G) = 5

4. *Graph Matriks*

   Graph Matriks pengujian pada Algoritma Koloni Semut dapat dilihat pada tabel 4.4.

Tabel 4.4 *Graph Matriks* Algoritma Koloni Semut

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
1		1													0
2			1			1									1
3				1											0
4					1										0
5		1													0
6							1								0
7								1							0
8						1			1						1
9										1					0
10											1				0
11										1		1			1
12													1		0
13		1												1	1
14															
HASIL															4

   V(G) = Jumlah *Graph Matriks* + 1

   V(G) = 4 + 1

   V(G) = 5

5. Kesimpulan

   Berdasarkan hasil pengujian dihasilkan nilai *Cyclomatic Complexity* yang sama yaitu 5, maka dapat disimpulkan bahwa pada proses pencarian dengan algoritma Koloni Semut berjalan dengan baik, karena setiap pengujian menghasilkan nilai yang sama.

4.2.4 Pengujian Beta

Pengujian *beta* merupakan pengujian yang dilakukan secara objektif, dimana pengujian langsung melibatkan para pengguna aplikasi *game* secara langsung dengan menggunakan kuesioner mengenai pernyataan para pengguna terhadap aplikasi *game* yang dibangun. Pengujian dilakukan terhadap 30 responden dengan usia minimal 12 tahun dan setiap responden diberikan 7 pernyataan untuk kemudian dilakukan penilaian dengan memilih 5 pilihan yang telah disediakan untuk setiap pernyataannya.

Data hasil kuesioner yang telah diberikan dapat dicari presentasinya dari masing-masing jawaban dengan menggunakan rumus [12] :

$$P = \frac{\sum s}{skorideal} \times 100\%$$

dimana,

P = Nilai presentase yang dicari

$\sum s$ = Jumlah skor yang didapatkan dari setiap pilihan jawaban dikalikan dengan nilai yang telah ditetapkan sebelumnya

skorideal = Nilai tertinggi pada point pilihan dikalikan dengan jumlah responden

Data yang didapat dari hasil kuisioner terhadap 30 orang responden adalah sebagai berikut :

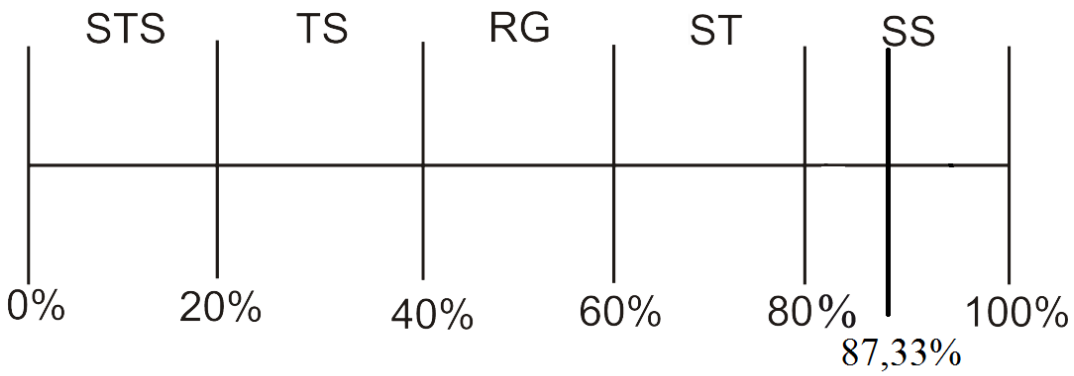
1. Cara pengoprasian aplikasi dari menu sampai arena permainan mudah dimengerti. Hasil kuesioner untuk pernyataan ke-1 dapat dilihat pada tabel 4.5.

Tabel 4.5 Perhitungan Pernyataan Nomor 1

Pernyataan	Jawaban	Nilai	Responden	Skor
1	Sangat Setuju	5	11	55
	Setuju	4	19	76
	Ragu-Ragu	3	0	0
	Tidak Setuju	2	0	0
	Sangat Tidak Setuju	1	0	0
JUMLAH			**30**	**131**

P = (131 / 150) x 100% = 87,33%

Dapat disimpulkan bahwa penilaian terhadap cara pengoprasian aplikasi dari menu sampai arena permainan adalah 87,33% dari yang diharapkan 100%, atau dapat dikategorikan sebagai sangat setuju, untuk lebih jelasnya dapat dilihat pada gambar 4.22.



Gambar 4.22 Interpretasi Skor Pernyataan ke-1

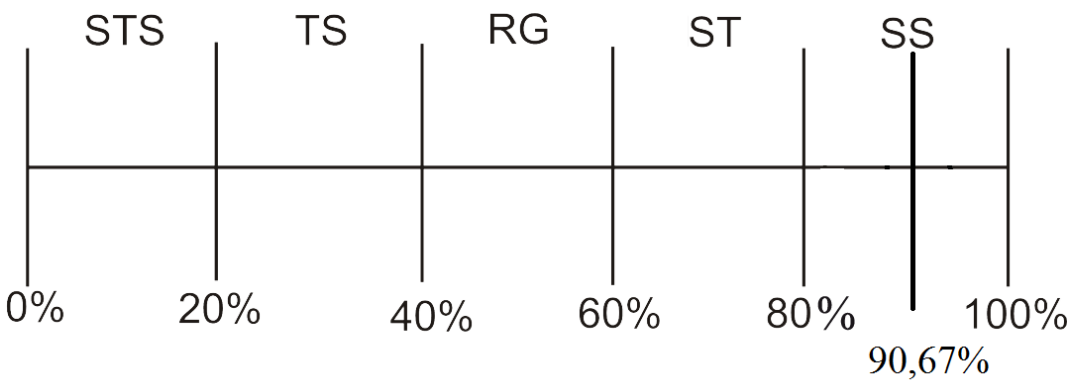
2. Tampilan dan suara sudah baik. Hasil kuesioner untuk pernyataan ke-2 dapat dilihat pada tabel 4.6.

Tabel 4.6 Perhitungan Pernyataan Nomor 2

Pernyataan	Jawaban	Nilai	Responden	Skor
2	Sangat Setuju	5	16	80
	Setuju	4	14	56
	Ragu-Ragu	3	0	0
	Tidak Setuju	2	0	0
	Sangat Tidak Setuju	1	0	0
JUMLAH			**30**	**136**

Persentase = (136 / 150) x 100% = 90,67%

Dapat disimpulkan bahwa penilaian terhadap tampilan dan suara sudah baik adalah 90,67% dari yang diharapkan 100%, atau dapat dikategorikan sebagai sangat setuju, untuk lebih jelasnya dapat dilihat pada gambar 4.23.



Gambar 4.23 Interpretasi Skor Pernyataan ke-2

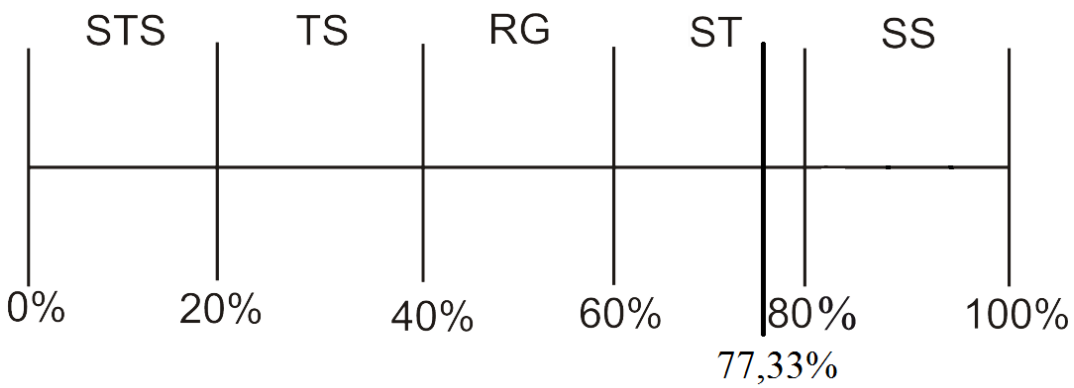
3. Kemunculan hama sulit ditebak. Hasil kuesioner untuk pernyataan ke-3 dapat dilihat pada tabel 4.7.

Tabel 4.7 Perhitungan Pernyataan Nomor 3

Pernyataan	Jawaban	Nilai	Responden	Skor
3	Sangat Setuju	5	4	20
	Setuju	4	18	72
	Ragu-Ragu	3	8	24
	Tidak Setuju	2	0	0
	Sangat Tidak Setuju	1	0	0
JUMLAH			**30**	**116**

Persentase = (116 / 150) x 100% = 77,33%

Dapat disimpulkan bahwa penilaian terhadap kemunculan hama sulit ditebak adalah 77,33% dari yang diharapkan 100%, atau dapat dikategorikan sebagai sangat setuju, untuk lebih jelasnya dapat dilihat pada gambar 4.24.



Gambar 4.24 Interpretasi Skor Pernyataan ke-3

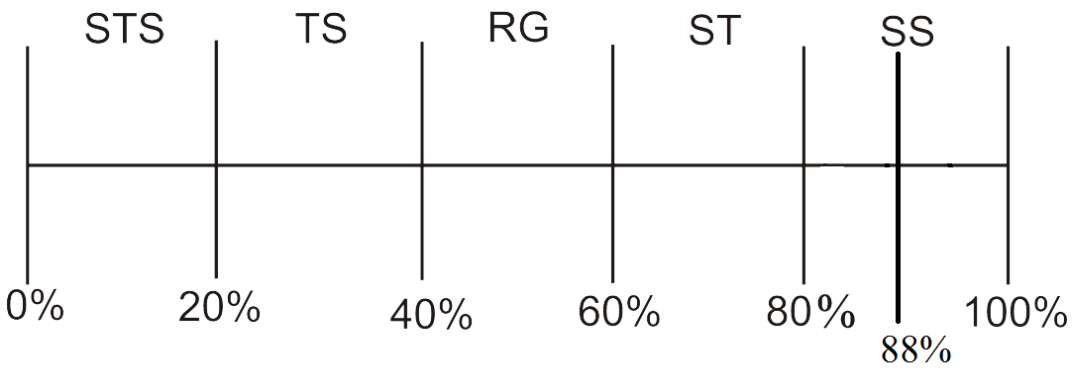
4. Cara membasmi hama mudah. Hasil kuesioner untuk pernyataan ke-4 dapat dilihat pada tabel 4.8.

Tabel 4.8 Perhitungan Pernyataan Nomor 4

Pernyataan	Jawaban	Nilai	Responden	Skor
4	Sangat Setuju	5	12	60
	Setuju	4	18	72
	Ragu-Ragu	3	0	0
	Tidak Setuju	2	0	0
	Sangat Tidak Setuju	1	0	0
JUMLAH			**30**	**132**

P = (132 / 150) x 100% = 88%

Dapat disimpulkan bahwa penilaian terhadap cara membasmi hama mudah. adalah 88% dari yang diharapkan 100%, atau dapat dikategorikan sebagai sangat setuju, untuk lebih jelasnya dapat dilihat pada gambar 4.25.



Gambar 4.25 Interpretasi Skor Pernyataan ke-4

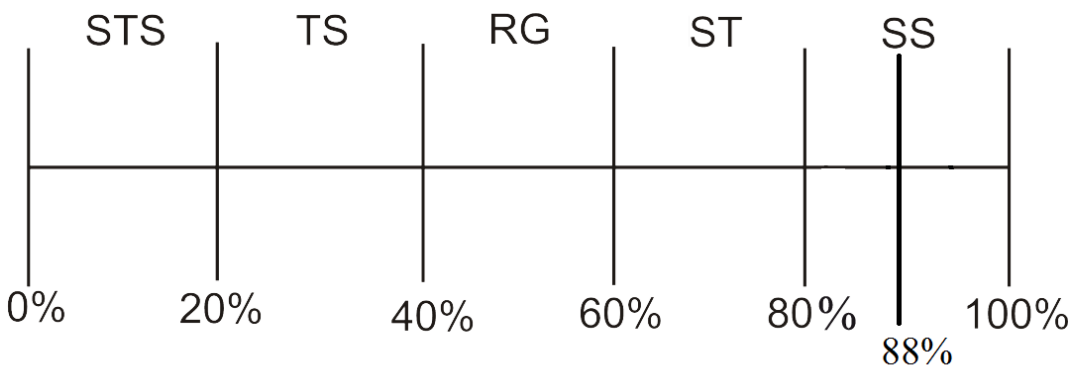
5. Hal dalam bercocok tanam sudah baik.. Hasil kuesioner untuk pernyataan ke-5 dapat dilihat pada tabel 4.9.

Tabel 4.9 Perhitungan Pernyataan Nomor 5

Pernyataan	Jawaban	Nilai	Responden	Skor
5	Sangat Setuju	5	12	60
	Setuju	4	18	72
	Ragu-Ragu	3	0	0
	Tidak Setuju	2	0	0
	Sangat Tidak Setuju	1	0	0
JUMLAH			**30**	**132**

Persentase = (132 / 150) x 100% = 88%

Dapat disimpulkan bahwa penilaian terhadap Hal dalam bercocok tanam sudah baik. adalah 88% dari yang diharapkan 100%, atau dapat dikategorikan sebagai sangat setuju, untuk lebih jelasnya dapat dilihat pada gambar 4.26.



Gambar 4.26 Interpretasi Skor Pernyataan ke-5

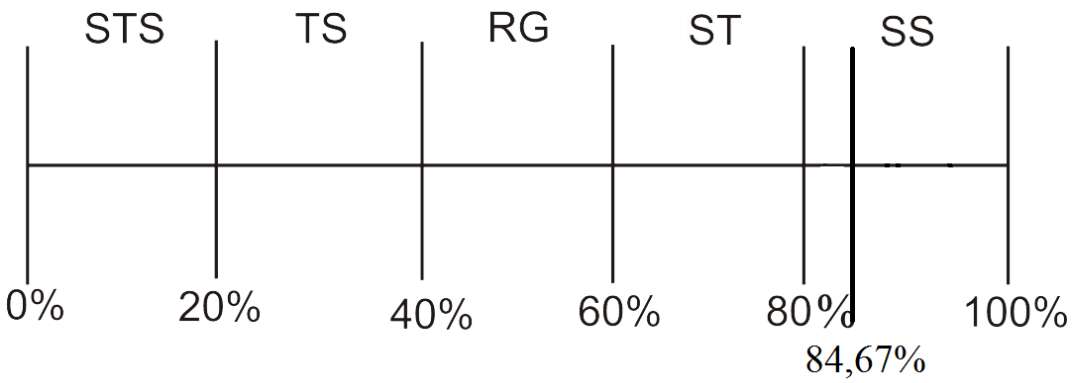
6. Membutuhkan manajemen waktu yang baik dalam bermain *game*. Hasil kuesioner untuk pernyataan ke-6 dapat dilihat pada tabel 4.10.

Tabel 4.10 Perhitungan Pernyataan Nomor 6

Pernyataan	Jawaban	Nilai	Responden	Skor
6	Sangat Setuju	5	8	40
	Setuju	4	21	84
	Ragu-Ragu	3	1	3
	Tidak Setuju	2	0	0
	Sangat Tidak Setuju	1	0	0
JUMLAH			**30**	**127**

Persentase = (127 / 150) x 100% = 84,67%

Dapat disimpulkan bahwa penilaian terhadap membutuhkan manajemen waktu yang baik dalam bermain *game* adalah 84,67% dari yang diharapkan 100%, atau dapat dikategorikan sebagai sangat setuju, untuk lebih jelasnya dapat dilihat pada gambar 4.27.



Gambar 4.27 Interpretasi Skor Pernyataan ke-6

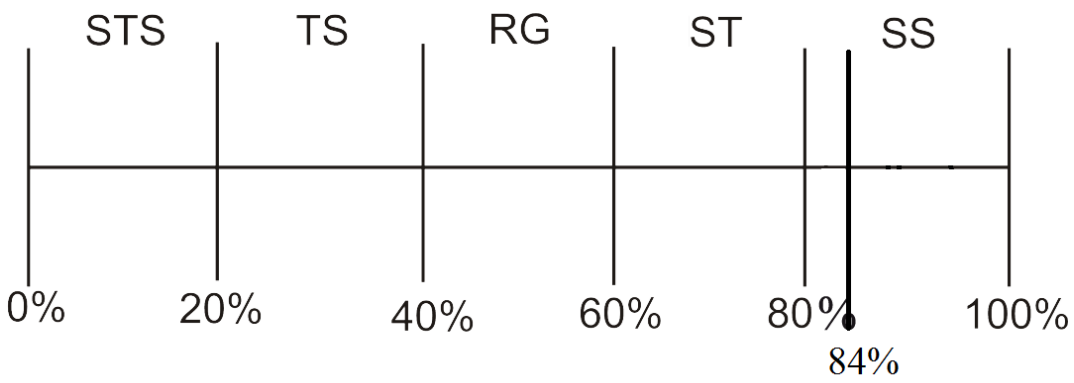
7. Cuaca yang berbeda berpengaruh dalam *game* maupun pemain. Hasil kuesioner untuk pernyataan ke-7 dapat dilihat pada tabel 4.11.

Tabel 4.11 Perhitungan Pernyataan Nomor 7

Pernyataan	Jawaban	Nilai	Responden	Skor
6	Sangat Setuju	5	8	40
	Setuju	4	20	80
	Ragu-Ragu	3	2	6
	Tidak Setuju	2	0	0
	Sangat Tidak Setuju	1	0	0
JUMLAH			**30**	**126**

Persentase = (126 / 150) x 100% = 84%

Dapat disimpulkan bahwa penilaian terhadap cuaca yang berbeda berpengaruh dalam *game* maupun pemain adalah 84% dari yang diharapkan 100%, atau dapat dikategorikan sebagai sangat setuju, untuk lebih jelasnya dapat dilihat pada gambar 4.28.



Gambar 4.28 Interpretasi Skor Pernyataan ke-7

4.2.4.2 Kesimpulan Pengujian Beta

Berdasarkan pengujian beta yang telah didapat dari hasil kuesioner yang telah dijawab oleh para responden, maka bisa diambil kesimpulan bahwa *game Farming Life* ini :

1. Konsep bercocok tanam telah berhasil diterapkan.
2. Game sudah cukup bervariatif.
3. NPC Hama cukup berjalan dengan baik.
4. Cukup dapat meningkatkan daya pikir pemain karena lebih banyak berpikir.
5. Menyenangkan dari tampilan.