

MA2151-Simulasi dan Komputasi Matematika

Proyek Akhir

Predator-Prey dengan menggunakan CA dan ABM



Anggota Kelompok:

1. Ashanti Fairuza (10121022)
2. Arrofiatuz Zahra (10121032)
3. Rahma Okta Feriska (10121050)
4. Amanda Risky Allawiyah (10121058)
5. Muhammad Fathan Assadad (10121076)

**FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
INSTITUT TEKNOLOGI BANDUNG**

2022

A. Analisis Masalah

Diberikan suatu permasalahan sebagai berikut :

1. a. Kembangkan sebuah simulasi dengan visualisasi menggunakan teknik *Cellular Automata* (CA) di mana sebuah sel dapat menjadi predator, mangsa, atau kosong. Gunakan delapan sel tetangga terdekat. Inisialisasi 50×50 grid dengan probabilitas kepadatan populasi tertentu untuk sel dari setiap jenis. Predator dan mangsa tidak dapat meninggalkan lingkungan (terapkan kondisi batas yang sesuai). Setiap langkah simulasi memiliki dua fase: perubahan keadaan dan gerakan.

Aturan perubahan keadaan adalah sebagai berikut:

- Jika mangsa "bertemu" (yaitu, memiliki tetangga) predator, predator memakan mangsanya (yaitu, situs mangsa menjadi kosong). Jika lebih dari satu predator ditemui, tetangga predator yang acak dipilih untuk makan. Pemangsa hanya dapat memakan satu mangsa setiap kali makan dan jika pemangsa makan pada langkah waktu t , ia hanya dapat makan lagi pada langkah waktu $t + 2$
- Predator mati (yaitu, situs predator menjadi kosong) ketika sudah terlalu lama (yaitu, A langkah waktu) tanpa makanan di mana $A \leftarrow 3 + N \bmod 3$, N adalah dua digit terakhir dari NIM terbesar di kelompok

Aturan untuk gerakan adalah sebagai berikut :

- Setiap individu berbelok ke arah acak (utara, timur, selatan, atau barat) yang tidak ditempati individu lain. Jika lebih dari satu individu berbelok ke sel yang sama, satu individu acak dipilih untuk pindah sementara yang lain tetap
- Jika tidak ada tetangga yang tersedia, individu tidak akan bergerak.

b. Jalankan simulasinya beberapa kali untuk variasi peluang populasi lainnya dan bahas hasilnya

2. a. Dari modul 11.2 dan 11.4, kita telah melihat simulasi *Cellular Automata* (CA) sangat mirip dengan simulasi *Agent-Based Modeling* (ABM). Sementara iterasi simulasi CA menyapu setiap sel grid, iterasi simulasi ABM memperbarui status masing-masing agen. Menariknya, hampir setiap masalah yang dapat diselesaikan dengan metode CA dapat diselesaikan dengan menggunakan teknik ABM. Kembangkan simulasi dengan visualisasi menggunakan teknik ABM untuk masalah di 1.

b. Jalankan simulasi nya beberapa kali untuk variasi peluang populasi lainnya dan bahas hasilnya

3. Bandingkan hasil yang didapat menggunakan teknik CA dengan hasil yang didapat menggunakan teknik ABM

Dalam pemodelan predator-prey ini, kami menggunakan dua model algoritma yaitu *Cellular Automata* dan *Agent Based Model*. Kami memvariasikan masing-masing nilai probabilitas populasi prey dan populasi predator dengan variasi sebagai berikut.

Variasi	Probabilitas Prey	Probabilitas Predator
1	0.05	0.01
2	0.1	0.05
3	0.15	0.09

Dari kedua simulasi ini (CA dan ABM) serta dari variasi nilai probabilitas, akan di lihat bagaimana perbedaan kedua teknik yang digunakan serta pengaruh tingkat probabilitas prey dan predator terhadap simulasi.

B. Rancangan Model

Simulasi hubungan *Predator-Prey* dapat dimodelkan dengan teknik *cellular automata* (CA) dan *agent based modelling* (ABM). Untuk tiap iterasi pada *cellular automata* (CA) setiap sel diperbarui, sedangkan pada ABM hanya diperbarui untuk setiap agen. Inti dari rancangan model simulasi ini adalah inisialisasi kondisi matriks lingkungan, aturan makan predator, dan pergerakan dari masing-masing individu. Untuk aturan makan sendiri adalah predator akan memakan prey yang ada di sekitarnya pada ketetanggaan Moore. Namun, jika terdapat lebih dari 1 predator bertemu, akan dipilih secara random predator yang akan memakan prey tersebut. Tiap predator hanya dapat makan sekali dan dapat makan kembali pada *timestep* $t + 2$. Untuk aturan pergerakan masing-masing individu adalah random disesuaikan dengan kondisi lingkungan jika tidak terdapat individu lain. Jika terdapat individu yang menuju sel sama, maka akan dipilih random untuk individu yang pindah dan tetap di tempat. Perpindahan tidak dapat dilakukan jika tidak ada sel yang kosong.

Berikut ini adalah rancangan model untuk masing-masing teknik.

A. Teknik *Cellular Automata* (CA)

Alur pembuatan model predator-prey dengan menggunakan teknik *cellular automata* adalah sebagai berikut

1. Membuat inisialisasi matriks lingkungan dengan fungsi *Init_Env_Grid*. Awalnya, matriks lingkungan dibuat kosong dengan ukuran 50×50 . Kemudian, secara random dilakukan pengisian entri untuk tiap sel-sel matriks. Untuk prey ditunjukkan dengan nilai 1 dan predator dengan nilai 2. Fungsi *Init_Env_Grid* akan me-return kondisi lingkungan yang sudah diisi oleh prey atau predator
2. Membuat inisialisasi matriks kondisi perut dari predator dengan fungsi *Init_Consume_Grid*. Untuk posisi pada matriks lingkungan yang terdapat predator, maka pada posisi yang sama di matriks kondisi perut akan ditunjukkan dengan nilai 1. Fungsi *Init_Consume_Grid* akan me-return kondisi perut dari predator.
3. Membuat kondisi batas dengan fungsi *Absorbing_Boundary*. Digunakan fungsi tersebut karena simulasi yang digunakan adalah simulasi tertutup yang membuat individu dalam sel matriks tidak keluar lingkungan. Nilai untuk *ghost cell* sendiri adalah -1 .
4. Membuat fungsi *Get_Neighbor_Moore* dan *Get_Neighbor_VonNeumann* untuk mengumpulkan tetangga dari suatu *site*
5. Membuat fungsi makan untuk predator dengan fungsi *Eating*. Pada fungsi ini akan di-return matriks lingkungan yang baru dan kondisi perut predator di setiap sel.
6. Membuat fungsi *Sensing* yang digunakan sebagai *planning* menunjukkan arah masing-masing individu untuk berjalan. Adapun pada fungsi ini akan me-return matriks lingkungan *sense* yang sudah diperluas.
7. Membuat fungsi *walking* untuk berjalan masing-masing individu. Fungsi ini akan me-return matriks lingkungan baru dan matriks kondisi perut baru.
8. Fungsi-fungsi yang telah dibuat akan digabungkan dalam 1 simulasi *cellular automata* dengan banyak iterasi adalah 50 kali. Selanjutnya, akan dilakukan variasi terhadap *prob_prey* dan *prob_predator* untuk 3 simulasi.

Variabel-variabel yang digunakan untuk simulasi dengan teknik *cellular automata* (CA) adalah sebagai berikut

n : ukuran baris dan kolom matriks

empty	: variabel berisi nilai 0 yang menunjukkan jika tidak ada predator/prey
prey	: variabel berisi nilai 1 yang menunjukkan prey
predator	: variabel berisi nilai 2 yang menunjukkan predator
prob_prey	: peluang terdapat prey
prob_predator	: peluang terdapat predator
SimLength	: banyak iterasi
Environment	: matriks lingkungan
Stomach_Condition	: matriks kondisi perut
Environment_Arr	: Array lingkungan
Stomach_Arr	: Array kondisi perut

B. Teknik *Agent Based Modelling* (ABM)

Alur pembuatan model predator-prey dengan menggunakan teknik *agent based modelling* mirip dengan teknik *cellular automata* (CA) yang membedakan adalah perbaruan matriks untuk setiap iterasi. Pada ABM, pembaruan kondisi sel matriks hanya dilakukan untuk agent. Berikut adalah alur pembuatan model dengan teknik *Agent Based Modelling* (ABM).

1. Membuat fungsi *Lingkungan*. Fungsi *Lingkungan* akan me-return matriks lingkungan yang entri-entrinya sudah diisi dengan nilai-nilai yang menggambarkan kondisi pada matriks tersebut. Untuk prey ditandai dengan nilai 1 dan predator dengan nilai 2
2. Membuat fungsi *extension* untuk memperluas matriks yang kemudian untuk *ghost cell* sendiri digunakan *absorbing boundary condition*. Digunakan *absorbing boundary condition* karena simulasi yang dilakukan merupakan simulasi tertutup. Fungsi tersebut akan me-return *Lingkungan_ext* yaitu matriks lingkungan yang sudah diperluas
3. Membuat fungsi *pick* yang digunakan untuk memeriksa apakah terdapat prey di sekitarnya. Digunakan ketetanggaan Moore untuk memeriksa apakah terdapat prey yang bisa dimakan atau tidak. Fungsi ini akan me-return indeks posisi dari predator.
4. Membuat fungsi *makan*. Fungsi *makan* tersebut digunakan untuk predator memangsa prey. Fungsi ini akan me-return matriks *Lingkungan_new*.
5. Membuat fungsi *sensing* untuk memeriksa apakah terdapat posisi untuk masing-masing individu dapat berpindah. Pada fungsi ini akan di-return indeks posisi masing-masing individu untuk pindah
6. Membuat fungsi *movement* yang mengatur pergerakan dari masing-masing individu. Pada fungsi ini akan direturn matriks *Lingkungan_new*
7. Membuat fungsi *check* yang digunakan untuk mengecek apakah terdapat predator yang tidak makan selama $(3 + 76 \bmod 3)$. Jika ada, maka predator tersebut akan mati.
8. Pada algoritma utama, dilakukan inisialisasi prey dan predator secara random dengan probabilitas tertentu. Kemudian, fungsi-fungsi yang telah dibuat digabungkan dalam satu simulasi. Adapun banyak iterasi yang dilakukan adalah sebanyak 50 kali.

Variabel-variabel yang digunakan untuk teknik *agent based modelling* (ABM) adalah sebagai berikut

Prey	: list posisi prey pada matriks lingkungan
------	--

Predator : list posisi predator pada matriks lingkungan
 nilai : nilai pada ghost cell
 n : banyak iterasi yang dilakukan
 Lingkungan : matriks lingkungan yang belum diperluas
 Lingkungan_ext : matriks lingkungan yang sudah diperluas

C. Solusi

Algoritma yang telah kami buat ini mengikuti indeks python.

1. Algoritma Predator-Prey dengan menggunakan Cellular Automata (CA)

Fungsi Inisiasi Lingkungan

```
Init_Env_Grid(n, prob_predator, prob_prey)
Initialize : n, prob_predator, prob_prey
Environment ← [[0 for j in range(n)] for i in range(n)]
for i ← 0 to n-1 do
  for j ← 0 to n-1 do
    rand ← np.random.random()
    if rand < prob_predator then
      Environment[i][j] ← 2
    else if rand < prob_prey then
      Environment[i][j] ← 1
    else
      Environment[i][j] ← 0
  return Environment
```

Fungsi Inisiasi Matriks Kondisi Perut

```
Init_Consume_Grid(Environment, n)
Initialize : Environment, n
Stomach_Condition ← [[-1 for j in range(n)] for i in range(n)]
for i ← 0 to n-1 do
  for j ← 0 to n-1 do
    if Environment[i][j] == 2 :
      Stomach_Condition[i][j] ← 1
  return Stomach_Condition
```

Fungsi Absorbing Boundary Condition

```
Absorbing_Boundary(Grid)
Initialize : Grid
n ← length(Grid)
Extended_Grid ← [[-0.01 for j in range(n+2)] for i in range(n+2)]
for i ← 0 to n-1 do
  for j ← 0 to n-1 do
```

```

    Extended_Grid[i+1][j+1] ← Grid[i][j]
return Extended_Grid

```

Fungsi Ketetanggaan Moore

```

Get_Neighbor_Moore(site, i, j)
Initialize : site, i, j
    return [site[i-1][j], site[i-1][j+1], site[i][j+1], site[i+1][j+1], site[i+1][j], site[i+1][j-1],
site[i][j-1], site[i-1][j-1]]

```

Fungsi Ketetanggaan Von Neumann

```

Get_Neighbor_VonNeumann(site,i,j)
Initialize : site, i, j
    return [site[i-1][j], site[i][j+1], site[i+1][j], site[i][j-1]]

```

#Fungsi Makan

```

Eating (Environment, Stomach_Condition, predator, prey, empty, A)
Initialize : Environment, Stomach_Condition, predator, prey, empty, A
n ← length(Environment)
for i←0 to n-1
    for j←0 to n-1
        if Stomach_Condition[i][j] >= 0 then
            Stomach_Condition[i][j] += 1
New_Environment ← np.copy(Environment)
for i←1 to n
    for j←1 to n
        if Environment[i-1][j-1] == prey then
            New_Stomach_Condition ← np.copy(Stomach_Condition)
            Extended_Environment ← Absorbing_Boundary(Environment)
            Extended_Stomach_Condition ← Absorbing_Boundary(Stomach_Condition)
            Neighbor_Type ← Get_Neighbor_Moore(Extended_Environment, i, j)
            Neighbor_Hunger ← Get_Neighbor_Moore(Extended_Stomach_Condition, i, j)
            Want_To_Eat ← []
            for a←0 to 7 do
                if Neighbor_Type[a] == predator and Neighbor_Hunger[a] > 1 then
                    Want_To_Eat.append(a)
            if length(Want_To_Eat) > 0 then
                Selected_Predator ← np.random.choice(Want_To_Eat)
                if Selected_Predator == 0 then
                    Stomach_Condition[i-2][j-1] ← 0
                else if Selected_Predator == 1 then
                    Stomach_Condition[i-2][j] ← 0
                else if Selected_Predator == 2 then
                    Stomach_Condition[i-1][j] ← 0
                else if Selected_Predator == 3 then

```

```

    Stomach_Condition[i][j] ← 0
else if Selected_Predator == 4 then
    Stomach_Condition[i][j-1] ← 0
else if Selected_Predator == 5 then
    Stomach_Condition[i][j-2] ← 0
else if Selected_Predator == 6 then
    Stomach_Condition[i-1][j-2] ← 0
else if Selected_Predator == 7 then
    Stomach_Condition[i-2][j-2] ← 0
    New_Environment[i-1][j-1] ← empty
else if Environment[i-1][j-1] == predator then
    if Stomach_Condition[i-1][j-1] == A then
        New_Environment[i-1][j-1] ← empty
        Stomach_Condition[i-1][j-1] ← -1
    else
        New_Environment[i-1][j-1] ← Environment[i-1][j-1]
else if Environment[i-1][j-1] == empty then
    New_Environment[i-1][j-1] ← empty
return New_Environment, Stomach_Condition

```

Fungsi Sensing

```

Sensing (Environment, empty)
Initialize : Environment, empty
Stay ← 0
North ← 1
East ← 2
South ← 3
West ← 4
n ← length(Environment)
Extended_Environment_Sense ← [[-0.01 for i in range(n+2)] for i in range(n+2)]
Extended_Environment ← Absorbing_Boundary(Environment)
for i ← 1 to n
    for j ← 1 to n
        if Extended_Environment[i][j] != empty then
            Neighbor_List ← Get_Neighbor_VonNeumann(Extended_Environment, i, j)
            Empty_Spot ← []
            for a ← 0 to 3 do
                if Neighbor_List[a] == empty then
                    Empty_Spot.append(a+1)
            if length(Empty_Spot) > 0 then
                Selected_Spot ← np.random.choice(Empty_Spot)
                Extended_Environment_Sense[i][j] ← Selected_Spot
            else
                Extended_Environment_Sense ← 0

```

```

else
    Extended_Environment_Sense[i][j] ← 0
for i ← 1 to n
    for j ← 1 to n
        if Extended_Environment[i][j] == empty then
            Neighbor_List ← Get_Neighbor_VonNeumann(Extended_Environment, i, j)
            Same_Spot ← []
            if Neighbor_List[0] == South then
                Same_Spot.append(Neighbor_List[0])
            if Neighbor_List[1] == West then
                Same_Spot.append(Neighbor_List[1])
            if Neighbor_List[2] == North then
                Same_Spot.append(Neighbor_List[2])
            if Neighbor_List[3] == East then
                Same_Spot.append(Neighbor_List[3])
            if length(Same_Spot) > 1 then
                Chosen_One ← np.random.choice(Same_Spot)
                if Chosen_One == South then
                    for a ← 0 to a ← (length(Same_Spot)-1) do
                        if Same_Spot[a] == West then
                            Extended_Environment_Sense[i][j+1] ← Stay
                        else if Same_Spot[a] == North then
                            Extended_Environment_Sense[i+1][j] ← Stay
                        else if Same_Spot[a] == East then
                            Extended_Environment_Sense[i][j-1] ← Stay
                else if Chosen_One == West then
                    for a ← 0 to a ← (length(Same_Spot)-1) do
                        if Same_Spot[a] == South then
                            Extended_Environment_Sense[i-1][j] ← Stay
                        else if Same_Spot[a] == North then
                            Extended_Environment_Sense[i+1][j] ← Stay
                        else if Same_Spot[a] == East then
                            Extended_Environment_Sense[i][j-1] ← Stay
                else if Chosen_One == East then
                    for a ← 0 to a ← (length(Same_Spot)-1) do
                        if Same_Spot[a] == South then
                            Extended_Environment_Sense[i-1][j] ← Stay
                        else if Same_Spot[a] == West then
                            Extended_Environment_Sense[i][j+1] ← Stay
                        else if Same_Spot[a] == North then
                            Extended_Environment_Sense[i+1][j] ← Stay
                else
                    for a ← 0 to (length(Same_Spot)-1) do
                        if Same_Spot[a] == South then

```



```

        Extended_Environment_Sense[i-1][j] ← Stay
    else if Same_Spot[a] == West then
        Extended_Environment_Sense[i][j+1] ← Stay
    else if Same_Spot[a] == East then
        Extended_Environment_Sense[i][j-1] ← Stay
return Extended_Environment_Sense

```

Fungsi Jalan Predator

```

Walking (Environment, empty)
Initialize : Environment, empty
Stay ← 0
North ← 1
East ← 2
South ← 3
West ← 4
n ← length(Environment)
New_Environment ← np.copy(Environment)
New_Stomach_Condition ← np.copy(Stomach_Condition)
Extended_Environment ← Absorbing_Boundary(Environment)
Sense_Grid ← Sensing(Environment, empty)

for i←1 to n
    for j←1 to n
        if Extended_Environment[i][j] != empty then
            if Sense_Grid[i][j] == North then
                New_Environment[i-2][j-1] ← Environment[i-1][j-1]
                New_Stomach_Condition[i-2][j-1] ← Stomach_Condition[i-1][j-1]
                New_Environment[i-1][j-1] ← empty
                New_Stomach_Condition[i-1][j-1] ← -1
            else if Sense_Grid[i][j] == East then
                New_Environment[i-1][j] ← Environment[i-1][j-1]
                New_Stomach_Condition[i-1][j] ← Stomach_Condition[i-1][j-1]
                New_Environment[i-1][j-1] ← empty
                New_Stomach_Condition[i-1][j-1] ← -1
            else if Sense_Grid[i][j] == South then
                New_Environment[i][j-1] ← Environment[i-1][j-1]
                New_Stomach_Condition[i][j-1] ← Stomach_Condition[i-1][j-1]
                New_Environment[i-1][j-1] ← empty
                New_Stomach_Condition[i-1][j-1] ← -1
            else if Sense_Grid[i][j] == West then
                New_Environment[i-1][j-2] ← Environment[i-1][j-1]
                New_Stomach_Condition[i-1][j-2] ← Stomach_Condition[i-1][j-1]
                New_Environment[i-1][j-1] ← empty
                New_Stomach_Condition[i-1][j-1] ← -1

```

```
return New_Environment, New_Stomach_Condition
```

Program Utama Predator Pray dengan CA

```
n ← 50
empty ← 0
prey ← 1
predator ← 2
N ← 76 #NIM terbesar
A ← 3 + (N % 3)
prob_prey ← 0.3      #Prob prey dan prob predator divariasikan
prob_predator ← 0.15
SimLength ← 50
```

```
Environment ← Init_Env_Grid(n, prob_predator, prob_prey)
Stomach_Condition ← Init_Consume_Grid(Environment, n)
Environment_Arr ← np.zeros((SimLength+1, n, n))
Environment_Arr[0] ← Environment
Stomach_Arr ← np.zeros((SimLength+1, n, n))
Stomach_Arr[0] ← Stomach_Condition
```

```
for t←1 to SimLength do
    Environment, Stomach_Condition ← Eating(Environment, Stomach_Condition, predator, prey,
        empty, A)
    Environment, Stomach_Condition ← Walking(Environment, empty)
    Environment_Arr[t] ← Environment
    Stomach_Arr[t] ← Stomach_Condition
```

```
for t←1 to SimLength do
    Total_Predator ← 0
    for i←0 to n-1
        for j←0 to n-1
            if Environment_Arr[t][i][j] == predator then
                Total_Predator += 1
    if Total_Predator == 0 then
        print("Semua Predator mati pada iterasi ke", t)
        break
```

Menampilkan Animasi

```
fig1 ← plt.figure()
#Fungsi init
def init1():
    plt.clf()
    return None
```

```

#Fungsi iterasi animasi
def animate1(i):
    plt.clf()
    frame ← Environment_Arr[i]
    ax ← sns.heatmap(frame, cmap = 'coolwarm', vmin= 0, vmax = 2)
    return None
#Animasi
anim1 ← animation.FuncAnimation(fig1, animate1, frames=50, init_func=init1)
#Membuat animasi
from matplotlib import rc
from IPython.display import HTML
rc('animation', html←'jshtml')
anim1

```

2. Algoritma Predator-Prey dengan menggunakan Agent Based Model (ABM)

#Fungsi Inisiasi Lingkungan

Lingkungan(Prey, Predator)

Initialize : Prey, Predator

```

Lingkungan ← [[0 for j in range(50)] for i in range(50)]
for i← 0 to (length(Prey))-1
    for j← 0 to 49
        for k← 0 to 49
            if (j,k) in Prey then
                Lingkungan[j][k] ← 1 #Mengisi sel matriks dengan Prey (1)
for i← 0 to (length(Predator))-1
    for j← 0 to (49)
        for k← 0 to(49)
            if (j,k) in Predator then
                Lingkungan[j][k] ← 2
return Lingkungan

```

#Fungsi Memperluas Matriks Lingkungan

extension(Lingkungan)

Initialize : Lingkungan

```

baris ← length(Lingkungan)
kolom ← length(Lingkungan[0])
Lingkungan_ext ← [[0 for i in range(kolom+2)] for i in range(baris+2)]
for i←1 to baris
    for j←1 to kolom
        Lingkungan_ext[i][j] ← Lingkungan[i-1][j-1]
return Lingkungan_ext

```

#Fungsi Ghost Cell (Absorbing Boundary Condition)

```
absorbing(Lingkungan_ext, nilai)
Initialize : Lingkungan_ext, nilai
baris ← length(Lingkungan_ext)
kolom ← length(Lingkungan_ext[0])
for j=1 to (kolom-2) do
    Lingkungan_ext[0][j] ← nilai
    Lingkungan_ext[baris-1][j] ← nilai
for i=0 to baris-1 do
    Lingkungan_ext[i][0] ← nilai
    Lingkungan_ext[i][kolom-1] ← nilai
return Lingkungan_ext
```

#Fungsi Mengecek apakah predator bisa memakan prey atau tidak

```
pick(i, j, N, E, S, W, NE, SE, NW, SW)
Initialize : i, j, N, E, S, W, NE, SE, NW, SW
neighbor_list ← [N, E, S, W, NE, SE, NW, SW]
pos_prey_list ← [i for i in range(length(neighbor_list)) if neighbor_list[i] == 1]
pos_predator_list ← [i for i in range(length(neighbor_list)) if neighbor_list[i] == 2]
if pos_prey_list == [] then
    return i, j
else
    r ← np.random.randint(0, length(pos_prey_list))
    if pos_prey_list[r] == 0 then
        if pos_predator_list == [] then
            return i-1, j
        else
            w ← np.random.randint(1,3)
            if w == 1 then
                return i-1, j
            else
                return i, j
    else if pos_prey_list[r] == 1 then
        if pos_predator_list == [] then
            return i, j+1
        else
            w ← np.random.randint(1,3)
            if w == 1 then
                return i, j+1
            else
                return i, j
    else if pos_prey_list[r] == 2 then
        if pos_predator_list == [] then
            return i+1, j
```

```

else
    w ← np.random.randint(1,3)
    if w == 1 then
        return i+1, j
    else
        return i, j
else if pos_prey_list[r] == 3 then
    if pos_predator_list == [] then
        return i, j-1
    else
        w ← np.random.randint(1,3)
        if w == 1 then
            return i, j-1
        else
            return i, j
else if pos_prey_list[r] == 4 then
    if pos_predator_list == [] then
        return i-1, j+1
    else
        w ← np.random.randint(1,3)
        if w == 1 then
            return i-1, j+1
        else
            return i, j
else if pos_prey_list[r] == 5 then
    if pos_predator_list == [] then
        return i+1, j+1
    else
        w ← np.random.randint(1,3)
        if w == 1 then
            return i+1, j+1
        else
            return i, j
else if pos_prey_list[r] == 6 then
    if pos_predator_list == [] then
        return i-1, j-1
    else
        w ← np.random.randint(1,3)
        if w == 1 then
            return i-1, j-1
        else
            return i, j
else then
    if pos_predator_list == [] then

```

```

    return i+1, j-1
else
    w ← np.random.randint(1,3)
    if w == 1 then
        return i+1, j-1
    else
        return i,j

```

#Fungsi Makan bagi Predator

```

makan(Lingkungan_ext)
Initialize : Lingkungan_ext
row ← length(Lingkungan_ext)
col ← length(Lingkungan_ext[0])
Lingkungan_new ← [[j for j in Lingkungan_ext[i]] for i in range(row)]
for i←1 to row-2
    for j←1 to col-2
        if Lingkungan_ext[i][j] == 2 then
            N ← Lingkungan_ext[i-1][j]
            E ← Lingkungan_ext[i][j+1]
            S ← Lingkungan_ext[i+1][j]
            W ← Lingkungan_ext[i][j-1]
            NE ← Lingkungan_ext[i-1][j+1]
            SE ← Lingkungan_ext[i+1][j+1]
            NW ← Lingkungan_ext[i-1][j-1]
            SW ← Lingkungan_ext[i+1][j-1]
            i_new, j_new ← pick(i,j, N, E, S, W, NE, SE, NW, SW)
            if last_eat[i][j] < 1 then
                Lingkungan_new[i_new][j_new] ← 1
                Lingkungan_new[i][j] ← 2
                last_eat[i][j] += 1
            else
                Lingkungan_new[i_new][j_new] ← 2
                Lingkungan_new[i][j] ← 0
                last_eat[i_new][j_new] ← 0
                last_eat[i][j] ← 0
            if i_new == i and j_new == j then
                last_eat[i_new][j_new] += 1
return Lingkungan_new

```

#Fungsi sensing apakah terdapat sel di sekitarnya yang bisa dilalui

```

sensing(i, j, site, N, E, S, W)
Initialize : i, j, site, N, E, S, W
site ← False

```

```

N ← True
E ← True
S ← True
W ← True
while (site == False) do
  if (N == True or E == True or W == True or S == True) then
    rand_direct ← np.random.random()
    if (rand_direct < 0.25) then
      if Lingkungan_ext[i-1][j] == 2 or Lingkungan_ext[i-1][j] == 1 then
        N ← False
      else
        site ← True
        return i-1, j
    else if (rand_direct < 0.5) then
      if Lingkungan_ext[i][j+1] == 2 or Lingkungan_ext[i][j+1] == 1 then
        E ← False
      else
        site ← True
        return i, j+1
    else if (rand_direct < 0.75) then
      if Lingkungan_ext[i+1][j] == 2 or Lingkungan_ext[i+1][j] == 1 then
        S ← False
      else
        site ← True
        return i+1, j
    else
      if Lingkungan_ext[i][j-1] == 2 or Lingkungan_ext[i][j-1] == 1 then
        W ← False
      else
        site ← True
        return i, j-1
    else
      return i, j

```

#Fungsi Jalan untuk Individu

```

movement(Lingkungan_ext)
Initialize : Lingkungan_ext
row ← length(Lingkungan_ext)
col ← length(Lingkungan_ext[0])
Lingkungan_new ← [[j for j in Lingkungan_ext[i]] for i in range(row)]
for i ← 1 to row-2
  for j ← 1 to col-2
    site ← Lingkungan_ext[i][j]
    N ← Lingkungan_ext[i-1][j]

```

```

E ← Lingkungan_ext[i][j+1]
S ← Lingkungan_ext[i+1][j]
W ← Lingkungan_ext[i][j-1]
if Lingkungan_ext[i][j] != 0 then
  if Lingkungan_ext[i][j] == 1 then
    i_new, j_new ← sensing(i, j, site, N, E, S, W)
    Lingkungan_new[i_new][j_new] ← 1
    Lingkungan_new[i][j] ← 0
  else if Lingkungan_ext[i][j] == 2 then
    i_new, j_new ← sensing(i, j, site, N, E, S, W)
    Lingkungan_new[i_new][j_new] ← 2
    Lingkungan_new[i][j] ← 0
    last_eat[i_new][j_new] ← last_eat[i][j]
    last_eat[i][j] ← 0
return Lingkungan_new

```

#Fungsi untuk mengecek apakah ada predator yang tidak makan selama (3+(76%3))

timestep

```

check(Lingkungan_ext)
Initialize : Lingkungan_ext
row ← length(Lingkungan_ext)
col ← length(Lingkungan_ext[0])
for i ← 1 to row-2
  for j ← 1 to col-2
    if Lingkungan_ext[i][j] == 2 and last_eat[i][j] == 3+(76%3) then
      Lingkungan_ext[i][j] ← 0
return Lingkungan_ext

```

#Algoritma Utama

#Posisi Prey dan Predator secara random

```

Prey ← []
Predator ← []
i ← 0
r ← np.random.randint(1,3)
x ← np.random.randint(0,50)
y ← np.random.randint(0,50)
if r == 1 then
  Prey.append((x,y))
else
  Predator.append((x,y))
while i < 2500 do
  rand ← np.random.random()
  x_new ← np.random.randint(0,50)
  y_new ← np.random.randint(0,50)

```



```

if rand< 0.15 then          #Nilai Prob_Prey divariasikan
  if x_new != x and y_new != y then
    Prey.append((x_new,y_new))
    x ← x_new
    y ← y_new
  else
    i ← i + 1
else if rand <0.24 then    #Nilai Prob_Predator divariasikan
  if x_new != x and y_new != y then
    Predator.append((x_new,y_new))
    x ← x_new
    y ← y_new
  else
    i ← i + 1

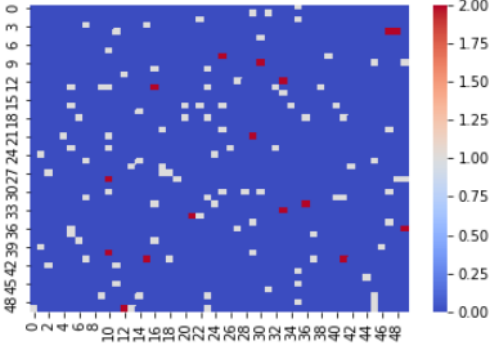
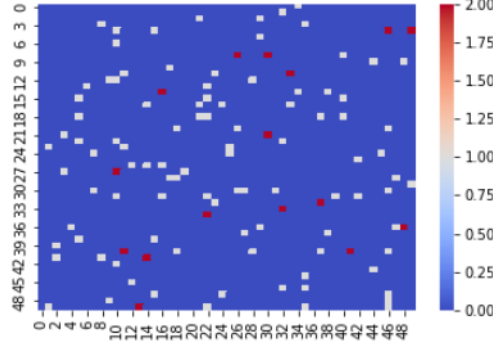
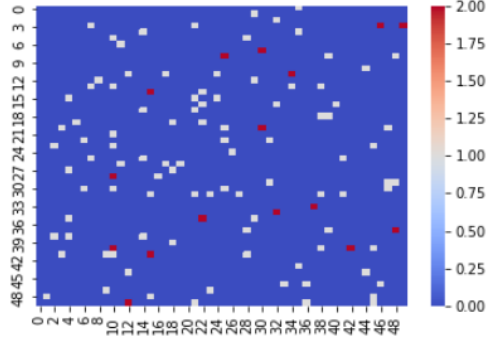
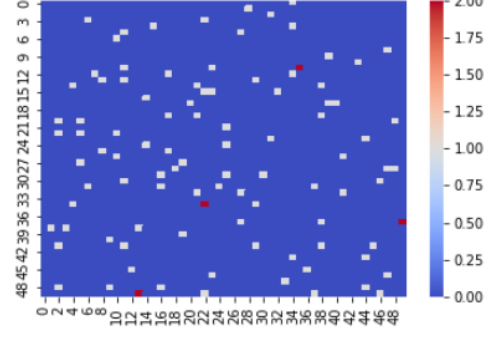
else
  i ← i + 1
#Konstanta
nilai ← -1
n ← 50
Lingkungan ← Lingkungan(Prey, Predator)
Lingkungan_ext ← extension(Lingkungan)
Lingkungan_list ← []
last_eat ← [[0 for j in range(52)] for i in range(52)]
for t=0 to n-1 do
  Lingkungan_ext ← absorbing(Lingkungan_ext, nilai)
  Lingkungan_ext ← makan(Lingkungan_ext)
  Lingkungan_ext ← movement(Lingkungan_ext)
  Lingkungan_ext ← check(Lingkungan_ext)
  Lingkungan_list.append(Lingkungan_ext)
#Kondisi Lingkungan Tiap Iterasi
for i=0 to (length(Lingkungan_list))-1 do
  ax ← sns.heatmap(dataLingkungan_list[i], vmin=-1, vmax=2)
  plt.title("Iterasi ke-"+str(i))
  plt.show

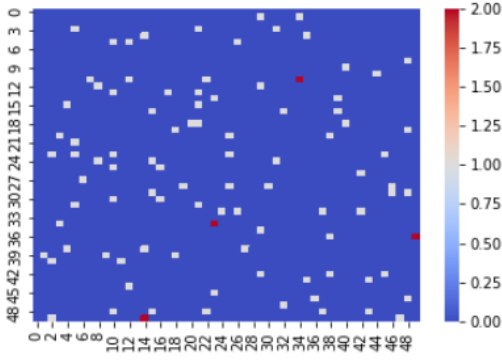
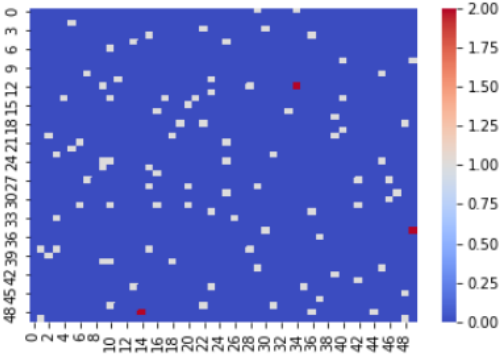
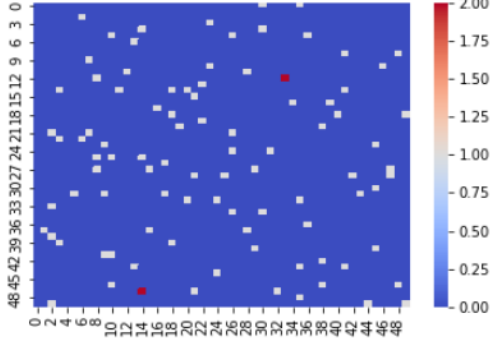
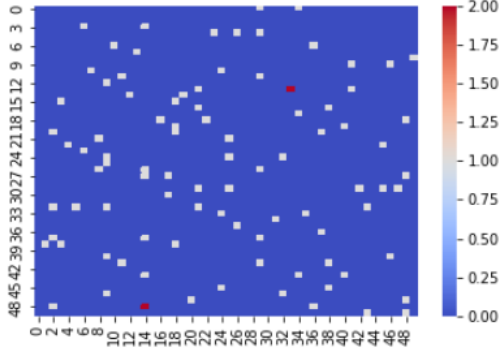
```

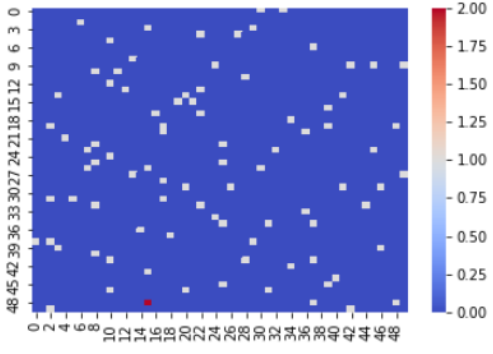
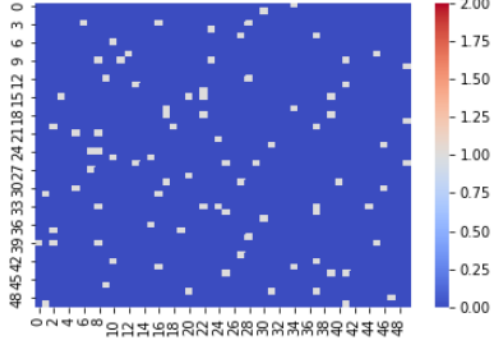
D. Hasil dan Kesimpulan

1. Predator-Prey dengan Cellular Automata (CA)

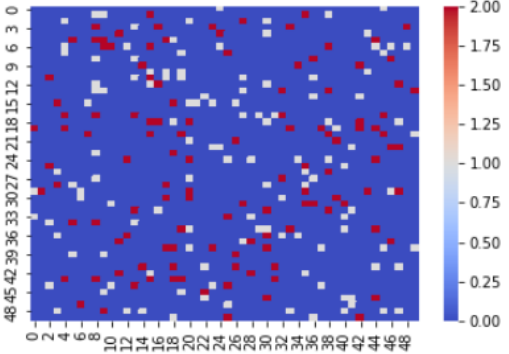
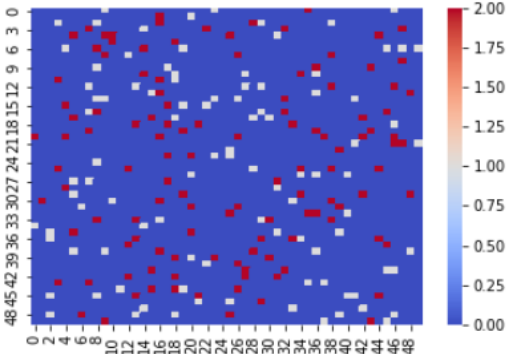
Tabel 1.1 Simulasi Predator-Prey dengan $prob_prey \leftarrow 0.05$ dan $prob_predator \leftarrow 0.01$

No	Iterasi Ke-	Grafik
1	0	
2	1	
3	2	
4	3	

5	4	
6	5	
7	6	
8	7	

9	8	
10	9	

Tabel 1.2 Simulasi Predator-Prey dengan $prob_prey \leftarrow 0.1$ dan $prob_predator \leftarrow 0.05$

No	Iterasi Ke-	Grafik
1	0	
2	1	

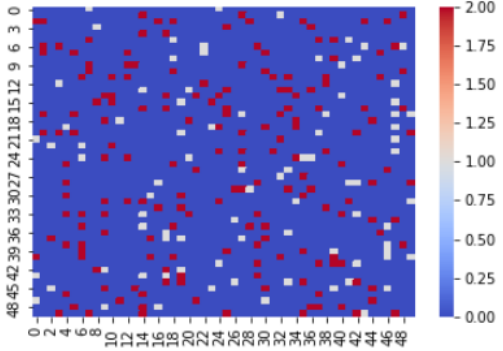
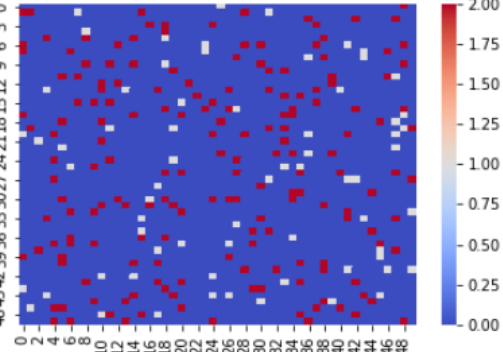
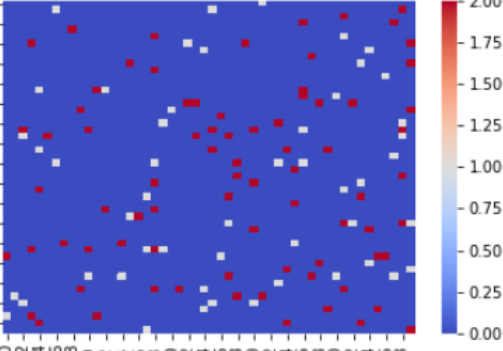
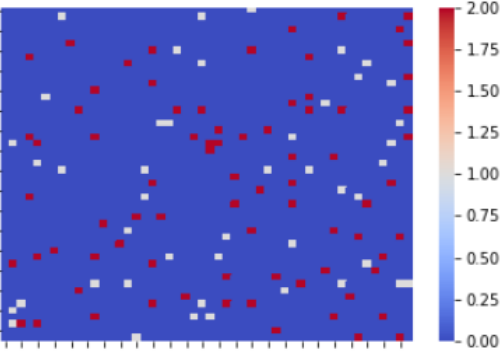
3	2	
4	3	
5	4	
6	5	

7	6	
8	7	
9	8	
10	9	

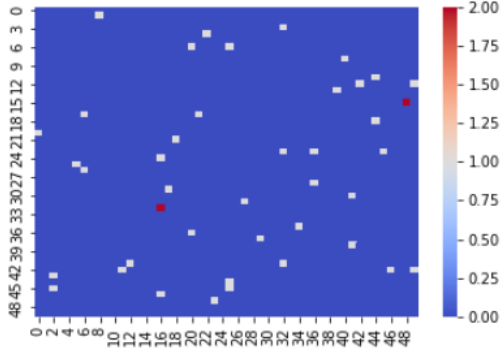
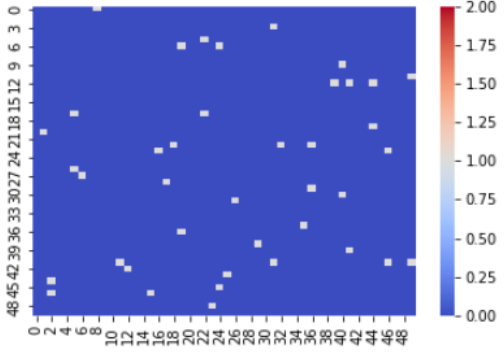
11	10	
12	11	
13	12	

Tabel 1.3 Simulasi Predator-Prey dengan $prob_prey \leftarrow 0.15$ dan $prob_predator \leftarrow 0.09$

No	Iterasi Ke-	Grafik
1	0	

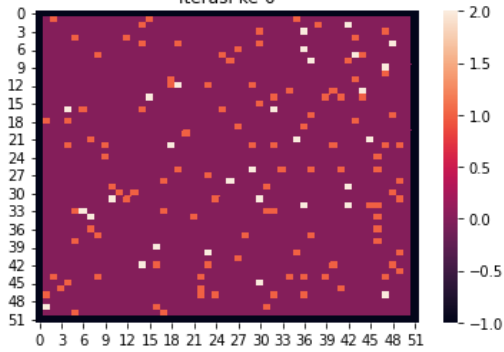
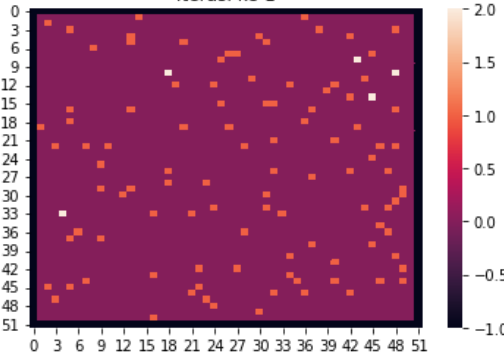
2	1	
3	2	
4	3	
5	4	

6	5	
7	6	
8	7	
9	8	

10	9	
11	10	

2. Predator-Prey dengan *Agent Based Model* (ABM)

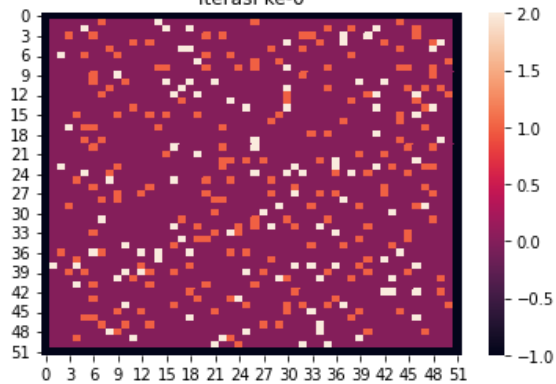
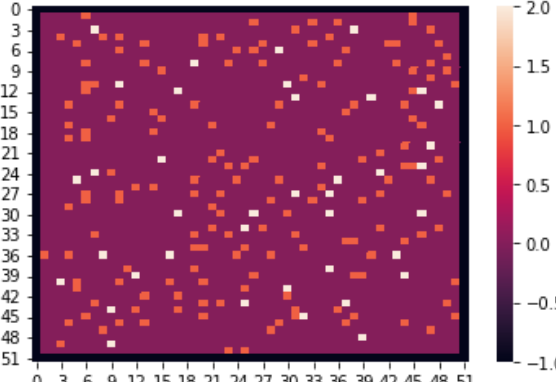
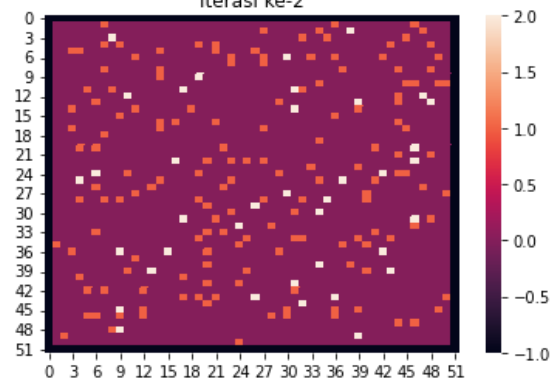
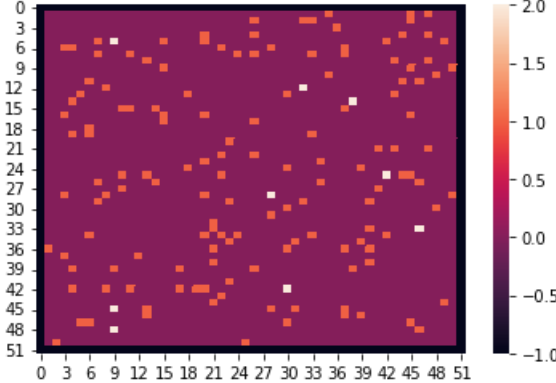
Tabel 2.1 Simulasi Predator-Prey dengan $prob_prey \leftarrow 0.05$ dan $prob_predator \leftarrow 0.01$

No	Iterasi Ke-	Grafik
1.	0	
2.	1	

3.	2	<p>Iterasi ke-2</p>
4.	3	<p>Iterasi ke-3</p>
5.	4	<p>Iterasi ke-4</p>
6.	5	<p>Iterasi ke-5</p>

\

Tabel 2.2 Simulasi Predator-Prey dengan $\text{prob_prey} \leftarrow 0.1$ dan $\text{prob_predator} \leftarrow 0.05$

No	Iterasi Ke-	Grafik
1.	0	
2.	1	
3.	2	
4.	3	

5.	4.	<p>Iterasi ke-4</p>
6.	5	<p>Iterasi ke-5</p>
7.	6	<p>Iterasi ke-6</p>
8.	7	<p>Iterasi ke-7</p>

Tabel 1.3 Simulasi Predator-Prey dengan $prob_prey \leftarrow 0.15$ dan $prob_predator \leftarrow 0.09$

No	Iterasi Ke-	Grafik
1.	0	<p>Heatmap titled "Iterasi ke-0" showing a random distribution of values on a 51x51 grid. The color scale ranges from -1.0 (dark purple) to 2.0 (light orange).</p>
2.	1	<p>Heatmap titled "Iterasi ke-1" showing a random distribution of values on a 51x51 grid. The color scale ranges from -1.0 (dark purple) to 2.0 (light orange).</p>
3.	2	<p>Heatmap titled "Iterasi ke-2" showing a random distribution of values on a 51x51 grid. The color scale ranges from -1.0 (dark purple) to 2.0 (light orange).</p>
4.	3	<p>Heatmap titled "Iterasi ke-3" showing a random distribution of values on a 51x51 grid. The color scale ranges from -1.0 (dark purple) to 2.0 (light orange).</p>

5.	4	<p>Iterasi ke-4</p>
6.	5	<p>Iterasi ke-5</p>
7.	6	<p>Iterasi ke-6</p>
8.	7	<p>Iterasi ke-7</p>

9.	8	<p>Iterasi ke-8</p>
10.	9	<p>Iterasi ke-9</p>
11.	10	<p>Iterasi ke-10</p>
12.	11	<p>Iterasi ke-11</p>

Kesimpulan Predator-Prey dengan CA :

1. Pada simulasi dengan CA, dengan menggunakan kepadatan *Prey* 0.05 dan kepadatan *Predator* 0.01, *Predator* sudah hilang pada iterasi ke-9. Artinya pada iterasi ke-9 semua *Predator* sudah mati.
2. Pada simulasi dengan CA, dengan menggunakan kepadatan *Prey* 0.1 dan kepadatan *Predator* 0.05, *Predator* sudah hilang pada iterasi ke-12. Artinya pada iterasi ke-12 semua *Predator* sudah mati.
3. Pada simulasi dengan CA, dengan menggunakan kepadatan *Prey* 0.15 dan kepadatan *Predator* 0.09, *Predator* sudah hilang pada iterasi ke-10. Artinya pada iterasi ke-10 semua *Predator* sudah mati.
4. Kepadatan *Prey* dan *Predator* tidak berbanding lurus dengan lamanya *Predator* menghilang dari simulasi (*Predator* mati). Pada saat variasi pertama ke variasi kedua, waktu *Predator* mati semakin lama. Akan tetapi, saat variasi kedua ke variasi ketiga, waktu *Predator* mati semakin cepat.

Kesimpulan Predator-Prey dengan ABM :

1. Pada simulasi ABM pertama, digunakan kepadatan *Prey* sebesar 0,05 dan kepadatan *Predator* sebesar 0,01. Pada iterasi ketiga sudah tidak ada *Predator*, artinya semua *Predator* sudah mati.
2. Pada simulasi ABM pertama, digunakan kepadatan *Prey* sebesar 0,1 dan kepadatan *Predator* sebesar 0,5. Pada iterasi ketujuh sudah tidak ada *Predator*, artinya semua *Predator* sudah mati.
3. Pada simulasi ABM pertama, digunakan kepadatan *Prey* sebesar 0,15 dan kepadatan *Predator* sebesar 0,09. Pada iterasi ke-11 sudah tidak ada *Predator*, artinya semua *Predator* sudah mati.
4. Kepadatan *Predator* dan *Prey* berbanding lurus dengan lama waktu yang dibutuhkan semua *Predator* untuk mati. Semakin besar kepadatan *Prey* dan *Predator* semakin lama pula *Predator* menghilang dari simulasi.

Perbandingan simulasi Predator-Prey dengan CA dan ABM :

1. Variasi kepadatan yang digunakan pada percobaan menggunakan CA dan ABM ini memberikan hasil yang berbeda mengenai lama waktu yang dibutuhkan *Predator* untuk menghilang dari simulasi. Hal ini sebenarnya bukan pengaruh dari penggunaan CA ataupun ABM. Hal ini terjadi karena penempatan *Predator* dan *Prey* dilakukan secara random sehingga lama waktu yang dibutuhkan *Predator* untuk mati bergantung pada penempatan *Prey* di sekitarnya.
2. Secara umum algoritma CA dan ABM memberikan hasil yang sama. Perbedaan CA dan ABM hanya terletak pada perubahan keadaan setiap iterasi. Pada CA, setiap iterasi mengubah semua kondisi setiap sel di grid. Sedangkan pada ABM, setiap iterasi mengubah keadaan tiap agen. Pada hal ini agen yang digunakan yaitu *predator* dan *prey*.