Optimalisasi Strategi Pergerakan dan Pengambilan Keputusan Karakter dalam *Game* Pac-Man menggunakan Program Dinamis

Kartini Copa - 13521026
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13521026@std.stei.itb.ac.id

***Abstract*—Pac-Man adalah permainan klasaik populer di mana karakter utama harus menghindari musuh dan mengumpulkan objek di labirin. Dalam permainan ini, strategi pergerakan dan pengambilan keputusan karakter Pac-Man memainkan peran penting dalam mencapai skor tertinggi. Program dinamis digunakan untuk mengoptimalkan strategi pergerakan karakter Pac-Man dengan mempertimbangkan posisi objek yang dikumpulkan dan posisi musuh. Metode ini memungkinkan karakter Pac-Man untuk membuat keputusan yang cerdas dan adaptif berdasarkan situasi saat ini dalam permainan.**

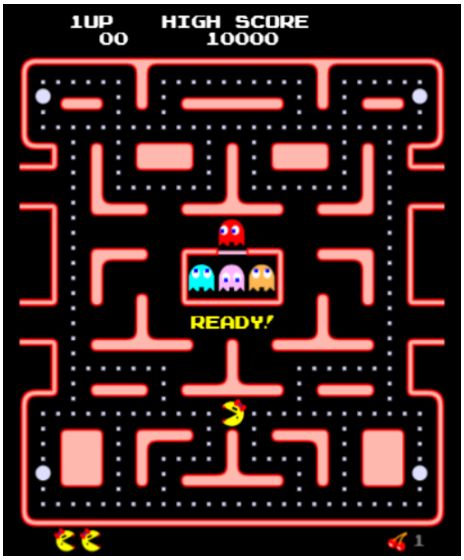
Keywords—dinamis; bottom-up; subproblem; Pac-Man

I. PENDAHULUAN

Permainan video telah menjadi industri yang berkembang pesat, menawarkan pengalaman hiburan yang menarik dan menantang. Dalam beberapa tahun terakhir, kecerdasan buatan dan optimalisasi strategi telah menjadi fokus penelitian yang signifikan dalam pengembangan permainan. Salah satu permainan klasik yang menarik minat para peneliti adalah Pac-Man, yang melibatkan karakter utama yang harus menghindari musuh sambil mengumpulkan objek di dalam labirin. Tujuan dari makalah ini adalah untuk mengoptimalkan strategi pergerakan dan pengambilan keputusan karakter dalam permainan Pac-Man menggunakan pendekatan program dinamis.

Dalam permainan Pac-Man, karakter utama harus menghadapi beberapa tantangan, termasuk menghindari musuh yang bergerak secara acak dan mengumpulkan sebanyak mungkin objek dalam waktu yang terbatas. Untuk mencapai tujuan ini, karakter Pac-Man perlu membuat keputusan yang cerdas dalam memilih rute pergerakan yang optimal dan menghindari tabrakan dengan musuh. Dalam konteks ini, optimalisasi strategi pergerakan dan pengambilan keputusan karakter menjadi sangat penting untuk meningkatkan kinerja permainan dan memberikan pengalaman yang lebih menarik bagi para pemain.

Pendekatan program dinamis merupakan salah satu metode untuk mengoptimalkan strategi pergerakan dan pengambilan keputusan karakter dalam permainan Pac-Man. Program dinamis memungkinkan karakter Pac-Man untuk membuat keputusan yang adaptif berdasarkan situasi permainan saat ini. Dengan mempertimbangkan posisi objek yang dikumpulkan dan posisi musuh, karakter Pac-Man dapat merencanakan jalur pergerakan terbaik yang akan memaksimalkan jumlah objek yang dikumpulkan dan menghindari musuh dengan efektif.



Melalui makalah ini, penulis akan membahas metode dan algoritma yang dapat digunakan dalam optimalisasi strategi pergerakan dan pengambilan keputusan karakter Pac-Man menggunakan program dinamis. Penulis juga akan menguji dan memvalidasi strategi yang diusulkan menggunakan simulator permainan Pac-Man dan skenario permainan yang berbeda. Hasil penelitian ini diharapkan dapat memberikan kontribusi dalam pengembangan lebih lanjut dalam bidang kecerdasan buatan, permainan video, dan aplikasi yang melibatkan karakter yang cerdas dan adaptif.

II. LANDASAN TEORI

A. *Game* Pac-Man

Game Pac-Man merupakan permainan arkade dengan genre *maze* atau labirin. Pemain mengendalikan karakter Pac-Man yang harus menjelajahi labirin untuk mengkonsumsi semua titik makanan yang ada, sambil menghindari musuh yang disebut Ghosts. Tujuan utama dalam permainan ini adalah mengumpulkan sebanyak mungkin titik makanan dan mencetak skor tinggi. Berikut merupakan karakter dan elemen penting dalam permainan Pac-Man

- Pac-Man: Karakter utama dalam permainan, biasanya digambarkan sebagai bulat kuning dengan mulut besar yang terus-menerus bergerak di sepanjang labirin.
- Ghosts: Musuh yang mengejar Pac-Man dalam labirin. Terdapat empat Ghosts dengan warna yang berbeda, yaitu *Blinky* (merah), *Pinky* (merah muda), *Inky* (biru), dan *Clyde* (oranye).
- Power Pellets: Objek khusus yang memberikan kekuatan kepada Pac-Man. Ketika Pac-Man mengkonsumsi Power Pellet, Ghosts menjadi lemah dan dapat dikejar serta dimakan oleh Pac-Man untuk sementara waktu.

Dalam game Pac-Man, musuh (Ghosts) menggunakan algoritma pergerakan yang cerdas untuk mengejar dan mencari Pac-Man. Algoritma ini dirancang untuk meningkatkan tingkat kesulitan permainan dan memberikan tantangan kepada pemain. Mengembangkan taktik pribadi yang efektif dapat membantu pemain dalam menghadapi tantangan yang semakin sulit seiring dengan naiknya level permainan. Terdapat beberapa strategi pergerakan yang umum digunakan dalam permainan Pac-Man, seperti:

- *Chase*: Ghosts berusaha untuk mengejar Pac-Man dengan tujuan menangkapnya.
- *Scatter*: Ghosts menjauh dari Pac-Man dan mencoba mengisi wilayah-wilayah tertentu di labirin.
- *Random*: Gerakan acak yang dapat membuat perilaku Ghosts menjadi tidak dapat diprediksi.

B. Program Dinamis dalam *Game* Pac-Man

Dalam *game* Pac-Man, algoritma program dinamis dapat digunakan untuk mengoptimalkan keputusan pergerakan karakter Pac-Man. Algoritma ini memanfaatkan prinsip pemrograman dinamis, di mana solusi optimal dari suatu masalah besar dapat ditemukan dengan memecahnya menjadi submasalah yang lebih kecil. Salah satu pendekatan yang umum digunakan dalam algoritma program dinamis adalah dengan menggunakan tabel atau matriks yang disebut dengan tabel keuntungan (Q-table). Tabel ini menyimpan informasi tentang keuntungan atau nilai-nilai yang dihubungkan dengan setiap langkah yang mungkin diambil oleh Pac-Man dalam situasi tertentu. Penerapan algoritma program dinamis dengan Q-table ini memungkinkan Pac-Man untuk mengoptimalkan strategi pergerakannya seiring berjalannya permainan. Pac-Man dapat memilih langkah-langkah yang memberikan hasil terbaik berdasarkan penilaian keuntungan yang disimpan dalam Q-table. Dengan demikian, Pac-Man dapat menghindari Ghosts, mengumpulkan titik makanan dengan efisiensi, dan meningkatkan peluangnya untuk mencetak skor tinggi. Langkah-langkah umum dalam menerapkan algoritma program dinamis dalam game Pac-Man adalah sebagai berikut. Pertama, mendefinisikan ruang keadaan (*state space*), Pac-Man berada dalam suatu keadaan atau posisi tertentu dalam labirin. Keadaan ini mencakup informasi seperti posisi Pac-Man, posisi Ghosts, dan keberadaan Power Pellets. Kedua, menentukan tindakan yang mungkin, setiap keadaan memiliki beberapa tindakan yang dapat diambil oleh Pac-Man, misalnya, bergerak ke atas, ke bawah, ke kanan, atau ke kiri. Ketiga, memperbarui tabel keuntungan (Q-table), tabel keuntungan diperbarui berdasarkan keadaan saat ini dan tindakan yang diambil. Pembaruan ini melibatkan penghitungan nilai keuntungan yang diharapkan dari setiap tindakan yang mungkin, dengan mempertimbangkan keadaan berikutnya. Kemudian, memilih tindakan terbaik setelah tabel keuntungan diperbarui, Pac-Man memilih tindakan berdasarkan strategi yang telah ditentukan. Strategi ini dapat mencakup pemilihan tindakan dengan nilai keuntungan tertinggi atau dengan probabilitas tertentu. Langkah selanjutnya adalah melakukan Tindakan, Pac-Man melakukan tindakan yang dipilih dan memperbarui keadaan saat ini. Selanjutnya mengulangi langkah-langkah di atas, proses ini diulang secara iteratif selama permainan berlangsung, dengan tabel keuntungan diperbarui dan tindakan terbaik dipilih pada setiap langkah.

Dengan algoritma program dinamis, Pac-Man dapat belajar dari pengalaman dan meningkatkan keputusan pergerakannya seiring berjalannya permainan. Algoritma ini memungkinkan Pac-Man untuk mengoptimalkan strategi pergerakan, seperti menghindari Ghosts atau memanfaatkan Power Pellets dengan efektif. Penerapan algoritma program dinamis dalam game Pac-Man memberikan kecerdasan buatan pada karakter Pac-Man, sehingga meningkatkan tingkat kesulitan dan tantangan dalam permainan. Berikut merupakan *pseudocode* untuk penerapan algoritma program dinamis dalam *game* Pac-Man.

```
function DynamicProgrammingAlgorithm(PacMan, Ghosts,
PowerPellets):

  stateSpace <- defineStateSpace(PacMan, Ghosts, PowerPellets)

  QTable <- initializeQTable(stateSpace)

  while game is not over:

    currentState <- getCurrentState(PacMan, Ghosts, PowerPellets)

    if isTerminalState(currentState):

      break

    availableActions <- getPossibleActions(currentState)

    bestAction <- bestAction(QTable, currentState,
availableActions)

    executeAction(PacMan, bestAction)

    nextState <- getNextState(PacMan, Ghosts, PowerPellets)

    reward <- calculateReward(currentState, bestAction, nextState)

    updateQTable(QTable, currentState, bestAction, reward,
nextState)

return PacMan's final strategy

end function
```

III. PEMBAHASAN DAN ANALISIS

A. Implementasi Program Dinamis dalam *Game* Pac-Man

Pemetaan persoalan untuk optimalisasi strategi pergerakan dan pengambilan keputusan karakter dalam game Pac-Man menggunakan program dinamis melibatkan langkah-langkah berikut. Inisialisasi posisi awal Pac-Man dan Ghost, serta kondisi lingkungan permainan. Langkah pertama Pac-Man dan perubahan posisi Ghost. Langkah kedua Pac-Man dan perubahan posisi Ghost, dan seterusnya. Pada setiap tahap, variabel keputusan adalah arah pergerakan yang dipilih oleh Pac-Man. Misalnya, atas, bawah, kanan, atau kiri. Pada setiap tahap juga terdapat status mencakup informasi seperti posisi Pac-Man saat ini, posisi Ghost saat ini, posisi titik makanan yang tersisa, kondisi lingkungan sekitar, misalnya dinding atau jalan terbuka. Nilai solusi optimal pada setiap tahap dapat didefinisikan sebagai keuntungan yang diharapkan dari langkah yang diambil oleh Pac-Man pada tahap tersebut. Nilai solusi optimal dapat bergantung pada faktor-faktor seperti jarak ke titik makanan, jarak ke Ghost, dan keadaan lingkungan. Hubungan rekursif harus ditentukan antara nilai solusi optimal pada tahap sekarang dengan nilai solusi optimal pada tahap sebelumnya. Misalnya, nilai solusi optimal pada tahap k dapat dihitung berdasarkan nilai solusi optimal pada tahap k-1 dan informasi pergerakan hantu. Dengan pemetaan ini, langkah-langkah pengembangan algoritma program dinamis dapat diikuti untuk mengoptimalkan strategi pergerakan dan pengambilan keputusan karakter Pac-Man dalam game.

Terdapat beberapa *subproblem* yang lebih kecil yang dapat mempengaruhi keputusan karakter dalam setiap *state* permainan.

- Mencari jalur terpendek dari satu posisi (misalnya, posisi saat ini Pac-Man) ke posisi lain (misalnya, posisi makanan atau Power Pellets). Dengan mencari jalur terpendek, Pac-Man dapat memilih langkah yang paling efisien untuk mencapai tujuan tersebut.

```
def shortest_distance(start, target, maze):
  rows = len(maze)
  cols = len(maze[0])
  distances = [[float('inf')] * cols for _ in range(rows)]
  distances[target[0]][target[1]] = 0
  steps = [(-1, 0), (1, 0), (0, -1), (0, 1)]
  # Menghitung jarak terpendek dengan pendekatan bottom-up
  for r in range(target[0], start[0] - 1, -1):
    for c in range(target[1], start[1] - 1, -1):
      if maze[r][c] != '#':
        for step in steps:
          nr = r + step[0]
          nc = c + step[1]
          if 0 <= nr < rows and 0 <= nc < cols and
          maze[nr][nc] != '#':
       distances[r][c]=min(distances[r][c],distances[nr][nc]+
1)
  return distances

# Implementasi
maze = [
    ['#', '#', '#', '#', '#'],
    ['#', '.', '.', '.', '#'],
    ['#', '.', '#', '.', '#'],
    ['#', '.', '.', '.', '#'],
    ['#', '#', '#', '#', '#']
]
start = (1, 1)
target = (3, 3)

distances = shortest_distance(start, target, maze)
for row in distances:
    print(row)
```

- Mempertimbangkan tindakan Ghost terdekat, *subproblem* ini melibatkan mempertimbangkan perilaku Ghost terdekat dan memilih aksi terbaik untuk menghindari atau mengejar Ghost tersebut. Pac-Man perlu memperhitungkan posisi dan arah pergerakan Ghost terdekat untuk mengambil langkah yang aman dan menghindari penangkapan oleh Ghost.

```
ghost_matrix = [[0 for _ in range(num_columns)]
for _ in range(num_rows)]

def update_ghost_matrix(ghost_position, ghost_direction):
  ghost_matrix[ghost_position[0]][ghost_position[1]] = 1
  for direction in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
    neighbor_position = (ghost_position[0] + direction[0],
ghost_position[1] + direction[1])
    if is_valid_position(neighbor_position):

ghost_matrix[neighbor_position[0]][neighbor_position[1]] = 1

def update_step_values():
  for row in range(num_rows):
    for col in range(num_columns):
      if ghost_matrix[row][col] == 1:
        step_values[row][col] = MAX_VALUE
      else:
        step_values[row][col] = calculate_step_value(row, col)

def get_best_action(pacman_position, available_actions):
  best_action = None
  best_value = MAX_VALUE
  for action in available_actions:
    new_position = get_new_position(pacman_position, action)
    value = step_values[new_position[0]][new_position[1]]
    if value < best_value:
      best_action = action
      best_value = value
  return best_action
```

- Memilih langkah yang optimal berdasarkan kondisi permainan, dengan mempertimbangkan kondisi permainan secara keseluruhan, termasuk jumlah makanan yang tersisa dan status kekuatan Pac-Man. Pac-Man perlu membuat keputusan yang tepat berdasarkan prioritas seperti mengumpulkan makanan, menghindari Ghost, atau memanfaatkan status kekuatan Pac-Man dengan bijak.

```
def calculate_step_value(row, col):
  distance_to_target = calculate_distance_to_target(row, col)
  distance_to_ghost = calculate_distance_to_ghost(row, col)
  if distance_to_ghost <= GHOST_PROXIMITY_THRESHOLD:
    return MAX_VALUE
   elif power_pellet_active:
     return distance_to_ghost
   else:
     return distance_to_target

def get_best_action(pacman_position, available_actions):
  best_action = None
  best_value = MAX_VALUE
  for action in available_actions:
    new_position = get_new_position(pacman_position, action)
    value=calculate_step_value(new_position[0],
new_position[1])
        if value < best_value:
          best_action = action
          best_value = value
  return best_action
```

- Menyesuaikan strategi berdasarkan situasi, *subproblem* ini melibatkan menyesuaikan strategi pergerakan dan pengambilan keputusan Pac-Man berdasarkan perubahan kondisi permainan. Pac-Man perlu dapat beradaptasi dengan perubahan dalam labirin, pergerakan Ghost, atau perubahan status kekuatan Pac-Man untuk mengambil keputusan yang paling efektif.

```
# Fungsi untuk menyesuaikan strategi berdasarkan situasi
def adjust_strategy(current_strategy, maze_changes, ghost_changes,
power_pellet_changes):
  new_strategy = current_strategy
  if maze_changes:
    new_strategy = find_optimal_route()
  if ghost_changes:
     if ghost_nearby():
       new_strategy = avoid_ghost()
     else:
       new_strategy = continue_current_strategy()

  if power_pellet_changes:
    if power_pellet_active():
      new_strategy = maximize_power_pellet()

  return new_strategy
```

Dalam permainan Pac-Man, implementasi program dinamis dengan strategi jangka panjang sangat penting untuk mencapai tujuan akhir. Program dinamis memungkinkan Pac-Man membuat keputusan yang tidak selalu menguntungkan secara langsung, tetapi dapat memberikan hasil yang lebih baik dalam jangka panjang. Misalnya, Pac-Man harus dapat menghindari makanan bernilai rendah meskipun terlihat menggoda, karena menghabiskan waktu dan energi untuk mengumpulkannya mungkin tidak sebanding dengan nilai poin yang diperoleh.

Pac-Man harus siap mengambil risiko untuk mendapatkan Power Pellet. Meskipun dapat berisiko, Power Pellet memberikan keuntungan yang signifikan dengan memberikan kemampuan Pac-Man untuk memakan hantu-hantu. Dalam program dinamis, Pac-Man perlu mempertimbangkan kemungkinan keberhasilan dalam mendapatkan Power Pellet, efeknya terhadap pergerakan hantu, dan potensi poin tambahan yang bisa didapatkan. Pac-Man harus menyeimbangkan antara menghindari risiko yang berlebihan dan memanfaatkan peluang untuk meningkatkan skor secara substansial.

Selain itu, strategi jangka panjang juga melibatkan mempertimbangkan keseimbangan antara mengumpulkan poin dan bertahan hidup dalam jangka waktu yang lebih lama. Meskipun mengumpulkan poin adalah tujuan utama, Pac-Man juga harus memperhatikan tingkat kesulitan permainan dan sumber daya yang tersedia. Misalnya, jika ada hantu-hantu yang mengintai dengan sangat dekat dan tidak ada Power Pellet yang tersedia, Pac-Man harus menghindari risiko yang tidak perlu dan fokus pada bertahan hidup.

Implementasi program dinamis dalam strategi jangka panjang juga memerlukan analisis kemungkinan langkah selanjutnya dan dampaknya terhadap tujuan akhir. Pac-Man harus mempertimbangkan pergerakan hantu-hantu, distribusi dan jenis makanan, serta pergerakan karakter itu sendiri saat membuat keputusan. Dengan mempertimbangkan semua faktor ini, Pac-Man dapat mengoptimalkan langkahnya untuk mencapai tujuan akhir dengan hasil yang maksimal.Keputusan yang diambil Pac-Man harus didasarkan pada evaluasi terus-menerus terhadap kondisi saat ini dan manfaat yang dapat diperoleh dari setiap langkah yang diambil. Dalam keseluruhan, implementasi program dinamis dengan strategi jangka panjang dalam permainan Pac-Man memungkinkan Pac-Man membuat keputusan yang tidak menguntungkan secara langsung, tetapi memiliki dampak positif dalam jangka panjang.

B. Menghitung Nilai Optimal Strategi Pergerakan

Dalam program dinamis untuk mengoptimalkan strategi pergerakan dan pengambilan keputusan karakter Pac-Man, dapat digunakan tabel keuntungan (fk(xk)) untuk menyimpan nilai keuntungan optimal pada setiap langkah atau posisi dalam permainan. Tabel keuntungan ini akan diisi berdasarkan rumus berikut:

$$fk(xk) = \max\{Rk(pk) + fk\text{-}1[xk - ck(pk)]\}$$

Di sini, fk(xk) menggambarkan keuntungan optimal yang dapat diperoleh pada langkah atau posisi saat ini (xk). Rk(pk) adalah keuntungan langsung yang didapatkan dari langkah atau tindakan (pk) pada langkah saat ini. fk-1[xk - ck(pk)] mengacu pada keuntungan optimal dari langkah sebelumnya (fk-1) pada posisi (xk - ck(pk)), di mana ck(pk) adalah perubahan posisi yang diakibatkan oleh langkah (pk). Rumus tersebut mencerminkan pendekatan pemrograman dinamis, di mana nilai keuntungan optimal pada langkah atau posisi ke-k dihitung dengan memilih tindakan atau pergerakan terbaik yang memberikan keuntungan maksimum. Perhitungan tersebut melibatkan perbandingan antara keuntungan yang diperoleh pada langkah atau posisi saat ini (Rk(pk)) dengan nilai keuntungan optimal pada langkah atau posisi sebelumnya setelah melakukan tindakan atau pergerakan pk (fk-1[xk - ck(pk)]). Berikut adalah implementasi menggunakan Bahasa pemrograman Python untuk menghitung table keuntungan optimal.

```
def optimize_strategy(game_positions, rewards, changes):
    num_steps = len(game_positions)
    fk = [0] * num_steps
    for k in range(num_steps):
        max_reward = float('-inf')
        current_position = game_positions[k]
        for action in range(len(rewards[k])):
            previous_position = current_position -
changes[k][action]
            if previous_position < 0:
              previous_reward = 0
            else fk[previous_position]
              total_reward = rewards[k][action] + previous_reward
            if total_reward > max_reward:
                max_reward = total_reward

        fk[current_position] = max_reward
    return fk

# Implementasi
game_positions = [0, 1, 2, 3, 4]
rewards = [[1, 2, 3], [2, 4, 6], [3, 6, 9], [4, 8, 12], [5, 10, 15]]
changes = [[1, 1, 1], [1, 1, 1], [1, 1, 1], [1, 1, 1], [1, 1, 1]]
optimal_strategy =
optimize_strategy(game_positions,rewards,changes)
print(optimal_strategy)
```

Pada dasarnya, program dinamis adalah sebuah pendekatan untuk memecahkan masalah yang melibatkan submasalah yang lebih kecil yang saling terkait. Dalam kasus ini, optimize_strategy digunakan untuk mencari strategi optimal dalam suatu permainan dengan posisi-permainan yang berbeda. Dalam implementasinya, tabel keuntungan fk(xk) dapat diinisialisasi dengan nilai-nilai awal yang sesuai, seperti 0 atau nilai negatif tak terhingga. Kemudian, tabel tersebut diisi

secara iteratif dengan menggunakan rumus di atas, dimulai dari langkah atau posisi awal hingga langkah atau posisi akhir permainan. Dengan menggunakan tabel keuntungan ini, Pac-Man dapat memilih strategi pergerakan yang optimal dengan memeriksa nilai keuntungan pada langkah atau posisi berikutnya dan memilih tindakan atau pergerakan yang memberikan nilai keuntungan maksimum.

Implementasi program dinamis dengan pendekatan *bottom-u*p untuk menghitung nilai optimal strategi pergerakan dalam permainan Pac-Man dapat dilakukan dengan langkah-langkah berikut:

1. Inisialisasi tabel atau matriks dengan nilai awal. Setiap sel dalam tabel akan mewakili state permainan, seperti posisi Pac-Man, dan nilai awalnya dapat diatur berdasarkan kondisi awal permainan.
2. Mulai dari state akhir permainan. Hitung nilai optimal untuk setiap langkah yang dapat diambil dari state tersebut berdasarkan subproblem yang relevan. Misalnya, jika subproblem adalah mencari jalur terpendek, hitung biaya atau nilai untuk setiap langkah berdasarkan jarak ke tujuan. Jika subproblem adalah mempertimbangkan tindakan Ghost terdekat, hitung biaya atau nilai untuk setiap langkah berdasarkan informasi Ghost terdekat. Jika subproblem adalah mempertimbangkan kondisi permainan secara keseluruhan, hitung biaya atau nilai berdasarkan faktor-faktor seperti jumlah makanan yang tersisa, kekuatan Pac-Man, dan sebagainya.
3. Lanjutkan ke state sebelumnya dan ulangi langkah 2 untuk setiap state hingga mencapai state awal permainan. Pada setiap langkah, gunakan informasi yang tersimpan dalam tabel atau matriks untuk menghitung nilai optimal untuk setiap langkah yang dapat diambil. Ini dapat dilakukan dengan membandingkan nilai-nilai dari langkah-langkah yang mungkin dan memilih langkah dengan nilai tertinggi atau paling optimal berdasarkan kriteria yang ditentukan.
4. Setelah selesai menghitung nilai optimal untuk setiap state, tabel atau matriks akan berisi informasi tentang langkah-langkah optimal dari setiap state. Informasi ini dapat digunakan untuk membuat keputusan strategi pergerakan Pac-Man dalam permainan. Pac-Man dapat memilih langkah dengan nilai optimal tertinggi atau mempertimbangkan faktor-faktor lain yang relevan dalam pengambilan keputusan.

Dengan pendekatan *bottom-up*, nilai optimal strategi pergerakan dapat dihitung secara efisien dan lengkap. Pendekatan ini memastikan bahwa semua kemungkinan langkah dan kondisi permainan dipertimbangkan dalam pengambilan keputusan Pac-Man. Dengan demikian, Pac-Man dapat mengambil langkah-langkah yang paling menguntungkan dan adaptif dalam menghadapi perubahan situasi permainan.

```
def calculate_optimal_strategy(game_states):
  num_states = len(game_states)
  optimal_values = [0] * num_states

  for i in range(num_states - 1, -1, -1):
    current_state = game_states[i]
    max_value = float('-inf')

    for action in current_state.get_available_actions():
      next_state = current_state.get_next_state(action)
      value = next_state.get_value()
              + optimal_values[next_state.get_index()]

      if value > max_value:
        max_value = value

    optimal_values[current_state.get_index()] = max_value

  return optimal_values[0]

# Implementasi kelas Gamestate dan metode yang relevan

class GameState:
  def __init__(self, index, value):
    self.index = index
    self.value = value
    self.available_actions = []

 def add_available_action(self, action):
   self.available_actions.append(action)

 def get_available_actions(self):
   return self.available_actions

 def get_next_state(self, action):
   pass
 def get_index(self):
   return self.index

 def get_value(self):
   return self.value

# Inisialisasi tabel atau matriks dengan nilai awal
game_states = [
    GameState(0, 0),
    GameState(1, 1),
    GameState(2, 3),
    GameState(3, 5),
    GameState(4, 2),
    ...
]

# Menambahkan aksi yang tersedia untuk setiap state
game_states[0].add_available_action('Up')
game_states[0].add_available_action('Down')
game_states[1].add_available_action('Left')
game_states[1].add_available_action('Right')
...

# Menghitung nilai optimal strategi pergerakan
optimal_strategy = calculate_optimal_strategy(game_states)
print(optimal_strategy)
```

C. Pengambilan Keputusan

Pengambilan keputusan dalam program dinamis untuk permainan Pac-Man melibatkan pemilihan langkah-langkah yang optimal berdasarkan perhitungan nilai strategi pergerakan. Dalam konteks ini, terdapat beberapa faktor yang harus dipertimbangkan untuk membuat keputusan yang adaptif dalam permainan Pac-Man. Pertama, mencari jalur terpendek menjadi salah satu faktor penting dalam pengambilan keputusan. Program dinamis dapat digunakan untuk menghitung jarak terpendek dari satu posisi ke posisi lain dalam labirin Pac-Man. Dengan mengevaluasi jarak terpendek, Pac-Man dapat memilih langkah yang paling efisien untuk mencapai tujuan seperti mengumpulkan makanan atau menghindari Ghost.

Selain itu, perilaku Ghost juga harus dipertimbangkan dalam pengambilan keputusan. Program dinamis dapat menghitung perilaku Ghost terdekat dan memprediksi tindakan terbaik yang harus diambil oleh Pac-Man. Hal ini memungkinkan Pac-Man untuk menghindari Ghost atau bahkan mengejar Ghost dengan bijaksana, berdasarkan informasi tentang posisi dan arah pergerakan Ghost terdekat. Pac-Man juga perlu mempertimbangkan kondisi permainan secara keseluruhan. Jumlah makanan yang tersisa dapat menjadi faktor penting dalam pengambilan keputusan. Program dinamis dapat menghitung nilai atau biaya berdasarkan faktor-faktor seperti jumlah makanan yang tersisa, status kekuatan Pac-Man, dan sebagainya. Dengan demikian, Pac-Man dapat membuat keputusan yang tepat berdasarkan prioritas seperti mengumpulkan makanan, menghindari Ghost, atau memanfaatkan status kekuatan Pac-Man dengan bijak.

Selanjutnya, program dinamis memungkinkan Pac-Man untuk menyesuaikan strategi berdasarkan situasi permainan. Pac-Man harus dapat beradaptasi dengan perubahan dalam labirin, pergerakan Ghost, atau perubahan status kekuatan Pac-Man untuk mengambil keputusan yang paling efektif. Dengan informasi yang diperoleh dari perhitungan nilai optimal, Pac-Man dapat mengubah strategi pergerakannya sesuai dengan situasi yang berkembang. Selain itu, program dinamis juga memungkinkan Pac-Man untuk mengoptimalkan penggunaan kekuatan khusus seperti Power Pellets. Pac-Man dapat memanfaatkan status kekuatan Pac-Man dengan bijak untuk mengubah dinamika permainan. Program dinamis dapat membantu menghitung keuntungan atau biaya dari menggunakan kekuatan khusus dan membantu Pac-Man dalam membuat keputusan yang optimal.

Dalam pengambilan keputusan program dinamis Pac-Man, strategi jangka panjang juga harus dipertimbangkan. Pac-Man perlu memiliki visi keseluruhan tentang tujuan jangka panjang dalam permainan. Program dinamis dapat membantu mengoptimalkan langkah-langkah Pac-Man dalam mencapai tujuan jangka panjang seperti menghabiskan semua makanan atau mencapai skor tertinggi.

Dalam keseluruhan, pengambilan keputusan program dinamis dalam permainan Pac-Man melibatkan perhitungan nilai optimal dan penyesuaian strategi pergerakan berdasarkan kondisi permainan. Dengan mempertimbangkan faktor-faktor seperti jalur terpendek, perilaku Ghost, kondisi permainan, dan strategi jangka panjang, program dinamis dapat membantu Pac-Man mengambil keputusan yang adaptif dan optimal untuk mencapai tujuan dalam permainan.

Program dinamis berikut digunakan untuk mengoptimalkan strategi pergerakan dan pengambilan keputusan karakter dalam permainan Pac-Man. Fungsi make_decision mengambil input berupa *state* (keadaan permainan saat ini) dan *table* (tabel atau matriks yang berisi informasi langkah optimal).

```
def make_decision(state, table):
  pacman_position = state['pacman_position']
  ghost_positions = state['ghost_positions']
  food_remaining = state['food_remaining']
  pacman_powered_up = state['pacman_powered_up']

  # Mendapatkan informasi dari tabel atau matriks
  optimal_moves = table[pacman_position]

  # Inisialisasi nilai optimal dan langkah terbaik
  optimal_value = float('-inf')
  best_move = None

  for move in optimal_moves:
  # Mendapatkan nilai dan informasi langkah
    value = optimal_moves[move]['value']
    is_safe = optimal_moves[move]['is_safe']
    is_powered_up = optimal_moves[move]['is_powered_up']

    # Evaluasi nilai optimal berdasarkan faktor-faktor permainan
    if is_safe:
      if is_powered_up and not pacman_powered_up:
      # Keputusan mengaktifkan kekuatan Pac-Man jika tersedia
        value += 10
      else:
      # Keputusan menghindari Ghost atau mengumpulkan makanan
        if ghost_positions:
          min_ghost_distance                                    =
min(manhattan_distance(pacman_position, ghost)
for ghost in ghost_positions)
                value -= min_ghost_distance
            else:
                value += food_remaining

        #  Memperbarui  nilai  optimal  dan  langkah  terbaik  jika
ditemukan nilai yang lebih tinggi
        if value > optimal_value:
            optimal_value = value
            best_move = move

    return best_move
```

Analisis pengambilan keputusan dengan program dinamis pada fungsi make_decision dalam permainan Pac-Man dapat dilakukan dengan memperhatikan beberapa factor penting. Informasi Permainan: Program menggunakan informasi permainan yang relevan, seperti posisi Pac-Man (pacman_position), posisi hantu (ghost_positions), jumlah makanan yang tersisa (food_remaining), dan status kekuatan Pac-Man (pacman_powered_up). Informasi ini digunakan untuk mengevaluasi dan memutuskan langkah terbaik. Tabel Optimal: Program menggunakan tabel (table) yang berisi langkah-langkah optimal untuk setiap posisi Pac-Man. Tabel ini memberikan informasi tentang nilai langkah, keamanan langkah, dan apakah langkah tersebut mengaktifkan kekuatan Pac-Man. Tabel ini membantu dalam pengambilan keputusan berdasarkan langkah-langkah yang tersedia. Evaluasi Faktor Permainan: Program melakukan evaluasi faktor-faktor permainan untuk menentukan nilai langkah optimal. Faktor-faktor ini mencakup keamanan langkah, pengaruh kekuatan Pac-Man, jarak terdekat dengan hantu, dan jumlah makanan yang tersisa. Program menggunakan faktor-faktor ini untuk menambah atau mengurangi nilai langkah saat memutuskan langkah terbaik. Pengambilan Keputusan: Program memilih langkah terbaik berdasarkan nilai langkah optimal yang telah dihitung. Program mengupdate nilai optimal (optimal_value) dan langkah terbaik (best_move) jika ditemukan nilai yang lebih tinggi dari nilai optimal saat ini. Akhirnya, program mengembalikan langkah terbaik yang telah dipilih.

Melalui penggunaan program dinamis ini, Pac-Man dapat mengoptimalkan strategi pergerakan dan pengambilan keputusan dalam permainan. Dengan mempertimbangkan faktor-faktor permainan dan memanfaatkan informasi dari tabel optimal, Pac-Man dapat membuat keputusan yang lebih cerdas, seperti menghindari hantu, mengumpulkan makanan, atau mengaktifkan kekuatan Pac-Man jika diperlukan.

IV. Penutup

A. Kesimpulan

Penggunaan program dinamis dalam optimalisasi strategi pergerakan Pac-Man efektif dalam meningkatkan kinerja dan kecerdasan karakter. Dengan mempertimbangkan faktor-faktor seperti posisi permainan, reward, dan perubahan posisi, kita dapat menghitung reward maksimum pada setiap langkah dan memilih tindakan terbaik berdasarkan reward tersebut. Pendekatan program dinamis memberikan fleksibilitas dan adaptabilitas yang tinggi dalam menyesuaikan strategi berdasarkan situasi permainan yang berubah-ubah.

Dalam pengembangan strategi pergerakan Pac-Man, dapat digunakan beberapa teknik seperti perhitungan jarak terpendek, evaluasi situasi seperti keberadaan hantu atau power pellet, dan pembaruan matriks hantu. Teknik-teknik ini memberikan wawasan yang lebih baik tentang lingkungan permainan dan membantu dalam mengambil keputusan yang tepat.

B. Saran

Pendalaman materi tentang algoritma program dinamis diperlukan untuk mendalami pemasalahan pengambilan keputusan pergerakan karakter dalam *game* Pac-Man.

Strategi dapat dikembangkan untuk berbagai jenis kemungkinan dalam *game* Pac-Man lebih spesifik.

Video Link at Youtube

https://youtu.be/v2DtiCcVSEo

Ucapan Terima Kasih

Puji syukur penulis panjatkan kepada Tuhan Yang Maha Esa karena dengan rahmat-Nya, penulis dapat menyelesaikan makalah dengan judul "Optimalisasi Strategi Pergerakan dan Pengambilan Keputusan Karakter dalam Game Pac-Man menggunakan Program Dinamis". Penulis berterima kasih kepada semua pihak yang telah memberikan dukungan dan bantuan selama penulisan makalah ini.Penulis ingin mengucapkan terima kasih yang sebesar-besarnya kepada orang tua, sahabat, dan teman-teman penulis yang selalu memberikan dukungan emosional dan semangat dalam menyelesaikan makalah ini. Tanpa dukungan mereka, penulis tidak akan bisa mencapai hasil yang memuaskan. Penulis juga ingin mengungkapkan rasa terima kasih kepada seluruh dosen IF2211 Strategi Algoritma yang telah memberikan pengajaran dan bimbingan kepada penulis dalam memahami program dinamis. Terima kasih atas pengetahuan dan wawasan yang berharga yang telah diberikan. Semoga hasil makalah ini dapat memberikan manfaat dan menjadi sumbangan kecil bagi pengembangan bidang optimalisasi strategi pergerakan dan pengambilan keputusan dalam *game* Pac-Man.

References

[1] Munir, Rinaldi, 2020. Program Dinamis (Bagian 1). Institut Teknologi Bandung. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Program-Dinamis-2020-Bagian1.pdf

[2] Munir, Rinaldi, 2020. Program Dinamis (Bagian 2). Institut Teknologi Bandung, https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Program-Dinamis-2020-Bagian2.pdf

[3] Insidini Fawwaz, Agus Winarta, Selvianna, Jevon Jose Ramli1, Lenny Marthalina Waruwu. Penerapan Algoritma Dynamic Programming pada Pergerakan Lawan dalam Permainan Police & Thief. https://www.ojs.uma.ac.id/index.php/jite/article/view/2169/1872.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 22 Mei 2023

Kartini Copa - 13521026