Pemilihan Langkah Maksimal dalam Permainan Persona 3 dengan Dynamic Programming

Makalah Ini Dibuat untuk memenuhi Tugas IF2211 Strategi Algoritma

Francisco Kenandi Cahyono
Teknik Informatika
Institut Teknologi Bandung
Bandung, Indonesia
13515140@std.stei.itb.ac.id

***Abstract*—Permainan elektronik atau yang dikenal dengan *video game* merupakan permainan yang banyak dimainkan oleh berbagai kalangan sejak diciptakan pada abad ke 20. Sejak saat itu, popularitasnya mengalahkan permainan konvensional yang melibatkan mainan fisik. Penggunaan keilmuan informatika tentu terlibat secara langsung dalam pengembangan sebuah permainan elektronik. Dalam konteks proyek ini, implementasi dari kecerdasan buatan atau acap kali disebut dengan *artificial intelligence* mengusik pikiran penulis untuk memerhatikan AI pada permainan persona 3 dan menggunakan *dynamic programming* yang sudah dipelajari pada kelas IF 2211 Strategi Algoritma untuk mendekati AI yang ada pada permainan.**

Keywords—Artificial Intelligence, Dynamic Programming, Role Playing Game, Permainan

I. PENDAHULUAN

Persona 3 adalah sebuah permainan bergenre *Japanese Role Playing Game* (JRPG) yang dirilis oleh ATLUS pada tahun 2006. Dalam permainan Persona 3, pemain menjadi seorang siswa SMA yang harus tergabung dalam program ekstrakurikuler Specialized Extracurricular Execution Squad (SEES), sekelompok siswa dengan kemampuan khusus yang meneliti fenomena misterius bernama *Dark Hour*, sebuah fenomena bertambahnya satu jam dalam satu hari setelah 00.00. Uniknya, hanya makhluk berkemampuan khusus yang dapat mengalami *Dark Hour* ini, sedangkan makhluk biasa akan terlindungi dalam peti mati. Setiap terjadi *Dark Hour*, sekolah mereka akan menghilang dan sebuah menara misterius bernama Tartarus muncul. Dengan kemampuan memanggil kekuatan dari persona, manifestasi kepribadian masing-masing individu, pemain dan teman-temannya yang tergabung dalam SEES mengeksplorasi Tartarus dan memecahkan misteri yang menyelimuti *Dark Hour*.

Permainan yang ber-genre JRPG terutama Persona 3 memiliki *gameplay* berupa *turn-based battle*. Melakukan pertarungan dengan memilih aksi bergantian antara musuh dan pemain. Pemain memiliki 7 pilihan dalam setiap pertarungan : menyerang secara fisik dengan senjatanya masing-masing, menggunakan kemampuan persona, menggunakan *item*, mengganti persona, diam, lari, dan memberi arahan kepada tim. Karakter lain tidak dapat mengganti persona dan memberi arahan. Setiap anggota tim memiliki skill persona masing-masing dengan efek memberi damage, menambah HP, atau mengganti status (menambah *damage*, meningkatkan *defense*, menambah kecepatan). Keunikan dari game ini ialah selain battle dan eksplorasi di *dark hour*, pemain juga dapat menjalani simulasi kehidupan sehari-hari seperti pergi ke sekolah, jalan-jalan keliling kota, atau menjalin relasi pertemanan yang dapat memengaruhi persona yang dimiliki karakter-karakter yang ada pada *game*. Keunikan lain yang terdapat pada game ini ialah pemain tidak mengontrol karakter lain selain karakter utama, karakter lain akan menggunakan AI tersendiri yang sesuai dengan komando dari pemain.

Untuk memahami lebih lanjut cara bermain dari permainan ini dapat melihat video pada tautan ini : https://www.youtube.com/watch?v=Oykxz8_MeoE

II. Dasar Teori

A. Program Dinamis

"The 1950s were not good years for mathematical research. We had a very interesting gentleman in Washington named Wilson. He was secretary of Defense, and he actually had a pathological fear and hatred of the word 'research'. I'm not using the term lightly; I'm using it precisely. His face would suffuse, he would turn red, and he would get violent if people used the term 'research' in his presence. You can imagine how he felt, then, about the term 'mathematical'. The RAND Corporation was employed by the Air Force, and the Air Force had Wilson as its boss, essentially. Hence, I felt I had to do something to shield Wilson and the Air Force from the fact that I was really doing mathematics inside the RAND Corporation. What title, what name, could I choose? "

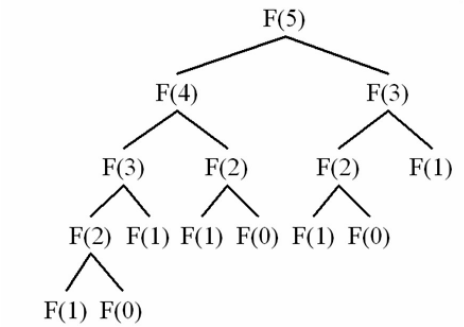
- Richard Bellman, on the origin of his term 'dynamic programming' (1984) –

Program dinamis adalah sebuah metode pemecahan masalah dengan menguraikan masalah tersebuh menjadi beberapa tahapan (*stage*). Program dinamis bukanlah sebuah algoritma spesifik, namun berupa sebuah metode pemecahan masalah (Mirip dengan hubungan antara Algoritma *quick sort* dan metode *Divide and Conquer*). Walaupun bernama "program" dinamis, metode ini tidak selalu diidentifikasikan dengan pemrograman, melainkan lebih dikenal dengan pengerjaannya dengan tabel.

Program dinamis merupakan metode yang cocok untuk menyelesaikan permasalahan optimasi. Tidak seperti rekursi biasa, program dinamis mencegah adanya perhitungan ulang (seperti pada program bilangan fibonacci).

```
RecFibo(n):
  if (n < 2)
      return n
  else
      return RecFibo(n − 1) + RecFibo(n − 2)
```

Berikut adalah algoritma fibonacci dengan pendekatan rekursif biasa, dan bila alur program divisualisasikan dengan nilai input berupa 5:



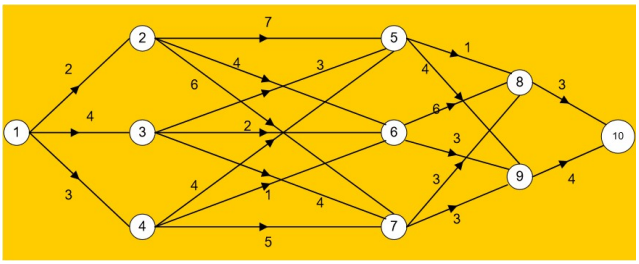
Dapat terlihat bahwa ada beberapa kalkulasi yang terulang kembali. Inilah salah satu hal yang ingin diperbaiki oleh *dynamic programming*. Pendekatan *dynamic programming* mengusahakan bahwa pemanggilan fungsi tidak perlu lebih dari sekali oleh karena itu pemanggilan fungsi disimpan pada suatu tempat penyimpanan.

```
IterFibo(n):
  F[0] ← 0
  F[1] ← 1
  for i ← 2 to n
      F[i] ← F[i − 1] + F[i − 2]
  return F[n]
```

Masih pada persoalan fibonacci, akan tetapi karena ukuran memori juga menjadi pertimbangan dan semakin besar angka yang diinginkan semakin "mahal" memori, maka persoalan fibonacci ini dapat diselesaikan dengan hanya menyimpan 2 hasil kalkulasi terdahulu karena (secara logika) lebih hemat memori dan memang dua hasil kalkulasi terdahulu yang dibutuhkan.

```
IterFibo2(n):
  prev ← 1
  curr ← 0
  for i ← 1 to n
      next ← curr + prev
      prev ← curr
      curr ← next
  return curr
```

Selain dengan fungsi rekursif, dynamic programming dapat diimplementasikan secara manual dengan menggunakan tabel, sesuai dengan sifat alami yang dimiliki oleh program dinamis. Berikut adalah salah satu contoh pemecahan soal dengan menggunakan program dinamis dan diselesaikan dengan tabulasi. Persoalan yang ingin dipecahkan adalah mencari rute terpendek dari suatu kota ke kota lain.



Pada contoh kasus, k (tahap) adalah proses memilih simpul tujuan berikutnya. Status (s) yang berhubungan dengan masing-masing tahap adalah simpul-simpul di dalam graf. x_k adalah peubah keputusan pada tahap k, $c_{s,xk}$ adalah bobot sisi dari s ke x_k. $f_k(s)$ adalah nilai minimum dari $f_k(x_k,s)$ dan $f_k(x_k,s)$ adalah total bobot lintasan dari x_k ke s.

Tahap 1:

$f_1(s) = c_{x1s}$

Dengan catatan: x_k* adalah nilai x_k yang meminimumkan f_s.

s	Solusi Optimum	
	$f_1(s)$	x_1*
2	2	1
3	4	1
4	3	1

Tahap 2:

$f_2(s) = \min\{ c_{x2s} + f_1(x_2)\}$

$s\backslash x_2$	$f_2(x_2,s) = c_{x2},s + f_1(x_2)$			Solusi Optimum	
	2	3	4	$f_2(s)$	x_2*
5	9	7	7	7	3 atau 4
6	6	6	4	4	4
7	8	8	8	8	2,3,4

Tahap 3:

$f_3(s) = \min\{ c_{x3s} + f_2(x_3)\}$

$s\backslash x_2$	$f_2(x3,s) = c_{x3},s + f_2(x_3)$			Solusi Optimum	
	5	6	7	$f_3(s)$	x_3*
8	8	10	11	8	5
9	11	7	11	7	6

Tahap 4:

$f_4(s) = \min\{ c_{x4s} + f_3(x_4)\}$

$s\backslash x_1$	$f_1(x_4,s) = c_{x4},s + f_3(x_4)$		Solusi Optimum	
	8	9	$f_4(s)$	x_4*
10	11	11	11	8 atau 9

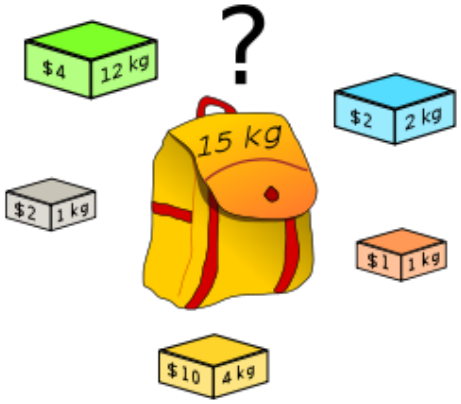
Dapat diturunkan bahwa terdapat tiga lintasan terpendek dari titik 1 ke titik 10 yakni:

- 1 -> 3 -> 5 -> 8 -> 10
- 1 -> 4 -> 5 -> 8 -> 10
- 1 -> 4 -> 6 -> 9 -> 10

Persoalan-persoalan lain yang dapat diselesaikan dengan dynamic program diantaranya adalah *Knapsack problem, capital budgeting, Travelling salesperson problem, Longest increasing subsequences, edit distance problem,* dan *chain matrix multiplication*.

B. *Knapsack Problem*

Knapsack problem merupakan sebuah permasalahan yang cukup terkenal di kalangan keilmuan informatika. Bentuk paling sederhana dari knapsack problem adalah kasus ketika terdapat sebuah wadah (*sack*) yang hanya mampu menahan beban tertentu. Setiap benda memiliki beban dan nilainya masing-masing. Tujuan akhir dari masalah ini adalah mencari kombinasi barang dengan limitasi berupa kapasitas kontainer namun dengan nilai yang maksimum.



Persoalan *knapsack* memiliki beberapa variasi diantaranya adalah *Bounded knapsack problem. Bounded knapsack problem* merupakan sebuah kasus ketika benda dapat dipilih lebih dari sekali dengan suatu batasan. Bila terdapat batasan jumlah m_j dari setiap barang bertipe j, maka persoalan ini dapat diformulasikan menjadi

$$\begin{aligned} \text{maximize} \quad & \sum_{j=1}^{n} p_j x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} w_j x_j \le c, \\ & x_j \in \{0, 1, \ldots, m_j\}, \quad j = 1, \ldots, n. \end{aligned}$$

Integer knapsack problem atau *unbounded knapsack problem* tidak memberi batasan jumlah pemilihan apapun. Dengan kata lain *unbounded knapsack problem* merupakan persoalan yang lebih *general* dari *bounded knapsack problem* karena tidak ada yang membatasi jumlah barang yang dapat diambil (walau ada batasan dari kapasitas *knapsack* itu sendiri). U*nbounded knapsack problem* dapat diformulasikan menjadi seperti berikut.

$$\begin{aligned} \text{maximize} \quad & \sum_{j=1}^{n} p_j x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} w_j x_j \le c, \\ & x_j \ge 0 \text{ integer}, \quad j = 1, \ldots, n. \end{aligned}$$

Multiple-choice knapsack problem adalah variasi *knapsack problem* saat benda-benda yang ada dapat dibagi menjadi k jenis dan setiap jenis harus diambil satu. Formulasi dari permasalahan tersebut adalah

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{k} \sum_{j \in N_i} p_{ij} x_{ij} \\ \text{subject to} \quad & \sum_{i=1}^{k} \sum_{j \in N_i} w_{ij} x_{ij} \le c, \\ & \sum_{j \in N_i} x_{ij} = 1, \qquad i = 1, \ldots, k, \\ & x_{ij} \in \{0, 1\}, \qquad i = 1, \ldots, k, \quad j \in N_i. \end{aligned}$$

Bila terdapat n benda dan m buah wadah dengan kapasitas yang berbeda-beda maka persoalan ini masuk kedalam variasi *multiple knapsack problem* dengan formulasi sebagai berikut.

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{m} \sum_{j=1}^{n} p_j x_{ij} \\ \text{subject to} \quad & \sum_{j=1}^{n} w_j x_{ij} \le c_i, \quad i = 1, \ldots, m, \\ & \sum_{i=1}^{m} x_{ij} \le 1, \qquad j = 1, \ldots, n, \\ & x_{ij} \in \{0, 1\}, \qquad i = 1, \ldots, m, \ j = 1, \ldots, n. \end{aligned}$$

Dengan p_j adalah keuntungan (profit), dan w_j adalah beban (*weight*) yang dimasukan kedalam satu atau lebih knapsack dengan kapasistas c. Pada teori-teori diatas nilai diasumsikan nilai p_j, w_j, dan c adalah bilangan bulat positif.

Persoalan knapsack dapat diselesaikan dengan fungsi rekurens biasa dengan contoh program sebagai berikut.

```
Basis:
B[k-1,w] bila w_k > w

Rekurens :
Max{B[k-1,w],B[k-1],w-w_k+b_k}
```

Namun kembali kepada kelemahan fungsi rekursif biasa, pada kasus ini ia akan menyimpan atau mengkalkulasi status yang sama berulang-ulang sehingga menyebabkan penyimpanan memori yang berlebihan dan waktu proses yang lebih lama, terutama bila jumlah status yang harus diolah banyak.

Oleh sebab itu alternatif yang lebih baik dibanding pemecahan oleh fungsi rekurens biasa adalah dengan program dinamis. *Knapsack problem* merupakan permasalahan yang memang sering dipecahkan oleh program dinamis karena ia merupakan persoalan optimasi. Berikut adalah salah satu contoh program penyelesaian masalah *knapsack* dengan program dinamis.

```
KnapSack(v,w,n,W)
{
    for(w = 0 to W) V[0,w] = 0;
        for(i = 1 to n)
            for(w = 0 to W)
      if(w[i] <=w)
          V[i,w] = max{V[i-1,w],
          v[i]+V[i-1,w-w[i]]};
      Else
          V[i,w] = V[i-1,w];
    Return V[n,W];
}
```

III. RANCANGAN IMPLEMENTASI PROGRAM

Pada makalah ini penulis tertarik meneliti penggunaan program dinamis untuk *artificial intelligence* dari permainan Persona 3. Penulis tertarik karena yang pertama, permainan persona 3 menggunakan sistem permainan yang cukup unik dengan membatasi pemain hanya dapat mengontrol pemeran utama saat sedang tahap pertarungan padahal dalam ada empat karakter. Keberadaan karakter lain yang digerakan oleh AI inilah yang dapat diteliti lebih lanjut bila dilambangkan dengan menggunakan dynamic programming. Alasan kedua dari peneliti adalah persoalan optimasi *damage* pada permainan Role Playing Game yang menganut sistem *turn based* memiliki pola yang mirip dengan persoalan knapsack sehingga relatif secara umum cukup *feasible* untuk diamati.

A. Batasan Simulasi

Penulis menggunakan beberapa batasan dalam mengamati persoalan ini :

1. Pada simulasi, permainan berupa 1 musuh melawan 1 karakter.

2. Simulasi berakhir bila musuh berhasil dikalahkan, atau dengan kata lain *Hit Points* (HP) musuh bernilai kurang dari sama dengan 0.
3. Diasumsikan musuh hanya menyerang satu kali dalam setiap giliran dan serangan musuh berupa serangan yang bukan menjadi kelemahan pemain.
4. Pemain maupun musuh tidak dapat menaikan atau menurunkan *stat*.
5. Diasumsikan pemain (dalam hal ini sistem simulasi) mengetahui stat dari musuh seperti kelemahannya dan HP nya.
6. Diasumsikan pemain dan musuh hanya dapat menggunakan serangan biasa dan *skill*.
7. Diasumsikan pemain dan musuh tidak dapat kabur, menggunakan item, dan mengganti algoritma (pada permainan asli ada beberapa jenis mode yang dapat diganti oleh pemain).
8. Diasumsikan jenis taktik yang digunakan kepada simulasi adalah "*Act Freely*" (taktik yang paling bebas, tidak ada batasan).
9. Tidak ada serangan musuh yang menyebabkan *status effect*.

B. Rencana Simulasi

Berikut adalah "coretan" program yang mungkin akan digunakan dalam bahasa java. Program merupakan salah satu contoh algoritma *unbounded knapsack problem* yang telah diubah sedikit agar dapat memenuhi permasalahan. (credits: http://rosettacode.org/wiki/Knapsack_problem/Unbounded#Java)

```
public class UnboundedKnapsack {

   protected Skill []  skills = {
                        new Skill("agidyne", 237,  237),
                        new Skill("bufudyne"  , 214,  214),
                        new Skill("ziodyne"   , 199,  199)
                        };
   protected final int    n = skills.length; // the number of
items
   protected Item      Enemy = new Skill("EnemyHP"   ,
0, 30000);
   protected Item      best = new Skill("best"   ,   0,  0);
   protected int  []  maxSkill = new int [n]; // maximum
number of skills
   protected int  []    iSkill = new int [n]; // current
indexes of skills
   protected int  [] bestAm = new int [n]; // best amounts

   public UnboundedKnapsack() {
      // initializing:
      for (int i = 0; i < n; i++) {
         maxSkill [i] = (int)(Enemy.getHP() /
skills[i].getDamage())
      } // for (i)

      // calc the solution:
      calcWithRecursion(0);

      // Print out the solution:
      NumberFormat nf =
NumberFormat.getInstance();
      System.out.println("Maximum damage achievable
is: " + best.getValue());
      System.out.print("This is achieved by carrying (one
solution): ");
      for (int i = 0; i < n; i++) {
         System.out.print(bestAm[i] + " " +
skills[i].getName() + ", ");
      }
      System.out.println();

   }

   // calculation the solution with recursion method
   // skill : the number of skill in the "skills" array
   public void calcWithRecursion(int skill) {
      for (int i = 0; i <= maxSkill[skill]; i++) {
         iSkill[skill] = i;
         if (skill < n-1) {
            calcWithRecursion(skill+1);
         } else {
            int    currDam = 0;   // current Damage
            double currHP = 0.0; // current HP
            for (int j = 0; j < n; j++) {
               currDam += iSkill[j] * skills[j].getDamage();
               currHP += iSkill[j] * skills[j].getHP();
            }

            if (currDamage > best.getDamage()
               &&
               currHP <= Enemy.getHP()
            )
            {
               best.setDamage (currDam);
               best.setHP(currHP);
               for (int j = 0; j < n; j++) bestAm[j] = iSkill[j];
            } // if (...)
         } // else
      } // for (i)
   } // calcWithRecursion()

   // the main() function:
   public static void main(String[] args) {
      new UnboundedKnapsack();
   } // main()

} // class
```

IV. KESIMPULAN

Program dinamis merupakan salah satu metode pemecahan masalah yang paling efektif untuk memecahkan masalah knapsack baik secara waktu maupun ruang. Selain itu pada kasus ini persoalan knapsack dapat digunakan dalam pendekatan *artificial intelligence* pada game.

V. PENGEMBANGAN LEBIH LANJUT

Untuk pengembangan lebih lanjut, program simulasi dapat diubah sedemikian rupa sehingga batasan-batasan yang ada pada bagian III dapat dieliminasi dan program simulasi tidak hanya menggambarkan damage maksimal saja, namun mampu menyimulasikan elemen-elemn penting pertarungan dalam permainan. Program masih mungkin dikembangkan lebih lanjut sehingga mampu:

1. Melibatkan lebih dari satu karakter dan juga lebih dari satu musuh
2. Program mampu melakukan eliminasi pohon ruang status bila HP pemain sudah kurang dari sama dengan 0 atau dengan kata lain pemain sudah mati.
3. Musuh mampu menyerang pemain setiap pemain menyerang dan musuh dapat menyerang dengan lebih dari satu jenis serangan walau pengaturan jenis serangan musuh bukan merupakan urusan dari algoritma ini.
4. Pemain maupun musuh dapat menaikan atau menurunkan *stat*.
5. Pemain selain menyerang, memiliki *skill* untuk menyembuhkan diri juga.

UCAPAN TERIMA KASIH

Penulis ingin mengucapkan terima kasih yang sebesar-besarnya kepada Tuhan Yang Maha Esa atas segala karunia Nya penulis dimampukan untuk menulis makalah IF 2211 Strategi Algoritma. Penulis juga ingin berterima kasih kepada pengajar K02 Strategi Algoritma Dr. Nur Ulfa Maulidevi ST,M.Sc. atas segala bimbingan dan kesabarannya dalam menyalurkan ilmu. Terima kasih juga untuk dosen IF2211 yang lain Dr. Ir. Rinaldi Munir,MT. dan ibu Masayu Leyla Khodra, S.T. Yang terakhir, penulis ingin berterima kasih kepada orang tua dan teman-teman K02 dan Informatika 2015 atas dukungan dan doanya.

Referensi

[1] Slide IF2211 Strategi Algoritma

[2] "Shin Megami Tensei: Persona 3". *En.wikipedia.org*. N.p., 2017. Web. 13 May 2017.http://megamitensei.wikia.com/wiki/Persona_3

[3] http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/Dynamic/dynamicIntro.htm

[4] https://www.youtube.com/watch?v=Oykxz8_MeoE

[5] http://jeffe.cs.illinois.edu/teaching/algorithms/notes/05-dynprog.pdf

[6] https://people.eecs.berkeley.edu/~vazirani/algorithms/chap6.pdf

[7] http://www.diku.dk/users/pisinger/95-1.pdf

[8] http://www.es.ele.tue.nl/education/5MC10/Solutions/knapsack.pdf

[9] http://rosettacode.org/wiki/Knapsack_problem/Unbounded#Java

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, 18 Mei 2017

ttd

Francisco Kenandi Cahyono

13515140