Penerapan Himpunan dan Teori Bilangan dalam Desain *Core Loop* dalam Beberapa Aplikasi Permainan

Akbar Al Fattah - 13522036[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13522036@std.stei.itb.ac.id

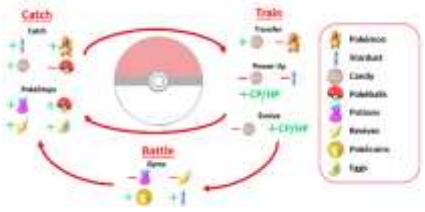
***Abstract*—Matematika diskrit adalah sebuah konsep dasar yang sering digunakan dalam pengembangan game. Konsep matematika diskrit yang digunakan di antaranya adalah himpunan dan teori bilangan. Kedua konsep tersebut digunakan juga dalam beberapa game terkenal, seperti *Pokémon, Grand Theft Auto, Minecraft,* dan lain-lain. Konsep matematika diskrit pada umumnya digunakan sebagai bagian dari mekanisme *game* itu sendiri.**

***Keywords*—Core Loop, Game, Himpunan, Teori Bilangan**

I. PENDAHULUAN

Game development (pengembangan game) adalah sebuah cabang dari pengembangan software (*software development*) yang menghasilkan sebuah aplikasi khusus yang berfungsi sebagai permainan/*game*. *Game development* memerlukan banyak sekali disiplin ilmu, seperti pemrograman, desain visual, desain suara, manajemen, dan desain dari *game* itu sendiri. Namun, yang akan dibahas pada makalah ini adalah dari desain *game*, yang berperan dalam pengalaman pengguna dalam bermain game tersebut.

Di dalam sebuah *game*, diperlukan sebuah *core loop*. *Core loop* di dalam sebuah *game* adalah kerangka besar dari alur permainan yang dilakukan oleh pemain yang dilakukan secara berulang-ulang. *Core loop* juga mencakup segala mekanisme yang ada di dalam sebuah *game*.



Gambar 1: Contoh *Core Loop* di dalam *game Pokemon Go*
Sumber: diambil dari referensi [1]

Penjelasan *core loop* pada makalah ini tidak akan dibahas secara mendetail. Makalah ini hanya akan membahas tentang penerapan konsep matematika diskrit di dalam mekanisme yang terdapat di dalam *core loop* sebuah *game*, terutama konsep himpunan dan teori bilangan.

II. TEORI DASAR

1. Himpunan
   Himpunan (*set*) adalah kumpulan objek yang elemennya harus berbeda semuanya dan tidak boleh ada elemen yang diulang. Contoh dari himpunan adalah
   C = {kucing, a, Amir, 10, paku}.

   Himpunan yang membolehkan ada elemen yang berulang disebut dengan *multiset*. Contoh dari *multiset* adalah
   A = {1, 1, 1, 2, 2, 3}.

   Operasi-operasi dasar yang terdapat dalam himpunan adalah:
   a. Irisan (∩)
      Irisan adalah operasi himpunan yang membentuk himpunan baru yang hanya berisi elemen-elemen yang sama-sama ada di kedua himpunan yang diiris.
      Contoh: A = {1,8,2,9,6}, B = {2,8,5,3},
      $$A \cap B = \{2,8\}$$
   b. Gabungan (∪)
      Gabungan adalah operasi himpunan yang membentuk himpunan baru yang berisi semua elemen yang ada di kedua himpunan, baik yang termasuk ke dalam irisan maupun yang bukan.
      Contoh: A = {1,8,2,9,6}, B = {2,8,5,3},
      $$A \cup B = \{1,8,2,9,6,5,3\}$$
   c. Komplemen (A^c)
      Komponen adalah operasi himpunan yang menghasilkan semua elemen yang bukan merupakan himpunan tersebut.
      Contoh: S = {1,2,3,..,10}, A = {1,2,6,8,9}
      $$A^c = \{3,4,5,7,10\}$$
   d. Selisih (-)
      Selisih adalah operasi himpunan yang merupakan himpunan tersebut dan tidak termasuk ke dalam irisan himpunan yang diselisihi.
      Contoh: A = { 1, 2, 3, ..., 10 }, B = {2,3,7,9}
      A – B = {1,4,5,6,8,10} dan B – A = ∅
   e. Beda setangkup (⊕)
      Beda setangkup adalah operasi himpunan yang mirip dengan gabungan, dengan pengecualian tidak memasukan semua elemen yang merupakan irisan

dari kedua himpunan tersebut.

Contoh: A = { 2, 4, 6 } B = { 2, 3, 5 }

$A \oplus B = \{3,4,5,6\}$

2. Teori Bilangan
   a. Aritmatika modulo

   Pada *game development*, konsep teori bilangan yang paling sering digunakan adalah aritmatika modulo.

   Modulo dalam matematika diskrit adalah sebuah operasi yang akan menghasilkan sisa bagi.

   Notasi operasi modulo adalah a mod b = r. Arti dari operasi ini adalah "r adalah sisa bagi dari a dibagi b".

   a mod b = r juga dapat dimaknai sebagai a = b*q + r dengan q adalah bilangan bulat sembarang.

   b. Kekongruenan modulo

   Aritmatika modulo tidak mengenal persamaan. Aritmatika modulo hanya mengenal kekongruenan yang dapat dianggap sebagai "persamaan modulo" (namun kedua konsep ini sangat berbeda dikarenakan kekongruenan modulo memiliki banyak solusi).

   Definisi dari kekongruenan modulo adalah

   $a \equiv b \ (mod\ m)$ jika dan hanya jika m habis membagi (a - b).

   Contoh: Misalkan $x \equiv 6 \ (mod\ 9)$, jadi, 9 | (x-6) atau $\frac{x-6}{9}$ harus bilangan bulat. Nilai yang memenuhi adalah x = …,-12,-3,6,15,… . $a \equiv b \ (mod\ m)$ juga dapat diartikan sebagai a = b + km dengan k bilangan bulat.

III. Analisis Kasus dan Pembahasan

A. Contoh Desain Game 1: Dual Type di Pokémon

Salah satu *game* yang dikenal oleh banyak orang adalah *Pokémon* yang diciptakan pada bulan Februari 1996 di Jepang dan sampai saat ini seri game *Pokémon* masih dilanjutkan. *Pokémon* adalah sebuah game RPG (*role-playing game*) yang membuat kita untuk menangkap dan beradu *Pokémon* dengan karakter di game mengikuti alur cerita atau dengan pemain lain secara daring.

Ada mekanisme yang sangat berperan penting dalam *game* ini, yaitu konsep *type*/tipe tiap *Pokémon*. Sampai saat makalah ini dibuat, ada 18 *type* yang ada di *Pokémon*, yaitu *Normal, Fighting, Flying, Poison, Ground, Rock, Bug, Ghost, Steel, Fire, Water, Grass, Electric, Psychic, Ice, Dragon, Dark,* dan *Fairy*.

Gambar 2: Daftar *Type* di *Pokémon* yang ada sampai sekarang
Sumber: diambil dari referensi [3]

Yang menarik dari konsep *type* ini adalah setiap *type* memiliki kelemahan (menyebabkan *damage* yang lebih besar) dan ketahanan (menyebabkan *damage* yang lebih kecil) jika diserang dengan *move* dengan *type* yang lain. Berikut adalah tabel *type* kelemahan dan ketahanan di *Pokémon* beserta pengali *damage*nya.

Tabel 1: Tabel Kelemahan dan Ketahanan *type* di *Pokémon* yang berlaku saat ini.
Sumber: diambil dari referensi [3]

Selain itu, satu *Pokémon* dapat memiliki minimal 1 *type* dan maksimum dua *type*. Akibatnya, kelemahan/ketahanan dari suatu *Pokémon* dapat bertumpuk dan berlipat ganda akibat memiliki dua buah *type*. Oleh karena itu, kita dapat menggunakan teori himpunan untuk menentukan kelemahan dari *Pokémon* yang memiliki dua buah *type*.

Pada pembahasan ini, kita akan mencoba menggunakan teori himpunan untuk menentukan kelemahan dan ketahanan dari suatu *Pokémon* yang memiliki dua *type*.

Misalkan kita ingin menentukan kelemahan dan ketahanan dari salah satu *Pokémon* yang bernama Tinkaton.

Gambar 3: Tinkaton dan dua *type* yang dimilikinya
Sumber: bulbapedia.bulbagarden.net, diedit untuk keperluan visualisasi.

Berdasarkan data yang ditentukan oleh desainer *Pokémon*, Tinkaton memiliki dua buah *type*, yaitu *Fairy* dan *Steel.*

Langkah pertama yang kita lakukan untuk mencari himpunan *type* kelemahan dan ketahanan dari Tinkaton adalah kita misalkan himpunan S (S di contoh ini bukan merupakan himpunan semesta) adalah himpunan seluruh *type* yang pengali *damagenya* selain 1 jika *type* Steel adalah *type* yang sedang diserang. (lihat tabel 1, himpunan S adalah semua *type* yang ada di kolom Steel (*defending type*) yang selnya tidak bernilai x1 atau 1), dan SMul adalah *array/multiset* yang merupakan nilai pengali *damage* yang berkorelasi dengan elemen di himpunan S dan harus terurut berdasarkan korelasi dengan elemen di S.

S = {Normal, Fighting, Flying, Poison, Ground, Rock, Bug, Steel, Fire, Grass, Psychic, Ice, Dragon, Fairy}

SMul = {1/2, 2, 1/2, 0, 2, 1/2, 1/2, 1/2, 2, 1/2, 1/2, 1/2, 1/2, 1/2}

Setelah itu, kita lakukan hal yang sama dengan *type Fairy*. Kita buat himpunan F dan *array/multiset* FMul.

F = {Fighting, Poison, Bug, Steel, Dragon, Dark}
FMul = {1/2, 2, 1/2, 2, 0, 1/2}

Setelah kita mendefinisikan himpunan di atas beserta nilai pengalinya, maka kita perlu mencari:

a. S – F (beserta SMul – FMul yang berkorelasi)
b. F – S (beserta FMul - SMul yang berkorelasi)
c. $S \cap F$ (beserta $SMul \cap FMul$ yang berkorelasi, catatan: elemen $SMul \cap FMul$ adalah perkalian dari elemen SMul dan FMul yang berkorelasi)

Didapatkan bahwa:

a. S – F = {Normal, Flying, Ground, Rock, Fire, Grass, Psychic, Ice, Fairy}
   SMul – FMul = {1/2,1/2,2,1/2,2,1/2,1/2,1/2,1/2,1/2}
b. F – S = {Dark}
   FMul – SMul = {1/2}
c. $S \cap F = \{Fighting, Poison, Bug, Steel, Dragon\}$
   $SMul \cap FMul =$ {1,0,1/4,1,0} (hasil perkalian SMul dan FMul)

Kemudian, kita eliminasi elemen di $S \cap F$ yang memiliki nilai 1 di $SMul \cap FMul$, jadi

$$S \cap F = \{Poison, Bug, Dragon\}$$
$$SMul \cap FMul = \{0,1/4,0\}$$

Langkah terakhir adalah kita operasikan
(S – F) ∪ (F – S) ∪ $(S \cap F)$ dan
(SMul – FMul) ∪ (FMul – SMul) ∪ $(SMul \cap FMul)$.

a. (S – F) ∪ (F – S) ∪ $(S \cap F)$ ={Normal, Flying, Ground, Rock, Fire, Grass, Psychic, Ice, Fairy, Dark, Poison, Bug, Dragon}
b. (SMul – FMul) ∪ (FMul – SMul) ∪ $(SMul \cap FMul)$ = {1/2,1/2,2,1/2,2,1/2,1/2,1/2,1/2,1/2, 1/2, 0,1/4,0}

Kita berhasil menentukan *type* apa saja yang menjadi kelemahan dari Tinkaton, dan *type* apa saja yang tidak mempan/kurang efektif untuk menyerang Tinkaton.

Untuk mempermudah dalam membaca ini, kita pisahkan menjadi himpunan TinkatonW (*type* yang menjadi kelemahan Tinkaton, yaitu *type* yang nilai pengali *damage*nya > 1), TinkatonR (*type* yang kurang efektif untuk menyerang Tinkaton, yaitu *type* yang nilai pengali *damage*nya 0<x<1), dan TinkatonI (*type* yang tidak mempan untuk menyerang Tinkaton, yaitu *type* yang nilai pengali *damagenya* = 0)

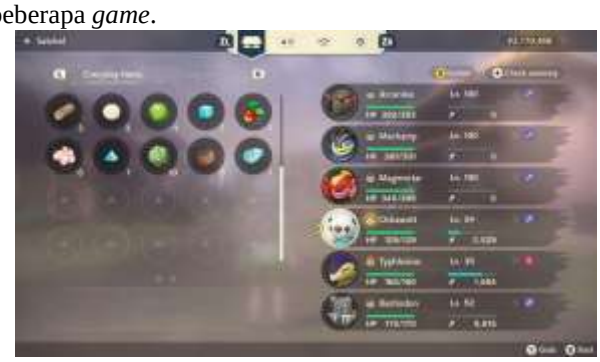
a. TinkatonW = {Ground, Fire}
   TinkatonWMul = {2,2}
b. TinkatonR = {Normal, Flying, Rock, Grass, Psychic, Ice, Fairy, Dark, Bug}
   TinkatonRMul = {1/2,1/2,1/2,1/2,1/2,1/2,1/2,1/2,1/2,1/4}
c. TinkatonI = {Poison, Dragon}
   TinkatonIMul = {0,0}

Kita berhasil menemukan *type* yang menjadi kelemahan Tinkaton, yaitu *Ground* dan *Fire*. Selain itu, kita juga berhasil menemukan *type* yang tidak mempan untuk menyerang Tinkaton, yaitu *Poison* dan *Dragon*, serta *type* yang kurang efektif untuk menyerang Tinkaton.

Cara yang sama juga dapat dilakukan untuk mencari kelemahan dan ketahanan dari suatu *Pokémon* yang lain, terutama untuk *Pokémon* yang memiliki dua *type*, seperti Lucario, Blaziken, Gardevoir, dan lain-lain. Konsep himpunan dari *type* di *Pokémon* juga diterapkan ke dalam *game* aslinya, sehingga dapat membuat alur permainan di *Pokémon* menjadi lebih menarik.

B. Contoh Desain Game 2: Inventory

Inventory/Penyimpanan barang adalah salah satu mekanisme yang sangat umum di dalam *game* yang mengharuskan kita untuk menggunakan barang-barang tertentu sebagai *core loopnya*. Contoh *game* yang memiliki *inventory* adalah *Grand Theft Auto*, *Pokémon*, *Undertale*, *Deltarune*, *Minecraft*, dan masih banyak lagi. Berikut adalah contoh *inventory* dari beberapa *game*.



Gambar 4: Contoh *Inventory* di *game Pokémon: Legends Arceus*

Sumber: diambil dari referensi [4]

Gambar 5: Contoh *Inventory* di *game Minecraft*

Sumber: diambil dari referensi [5]

Perhatikan bahwa *inventory* pada gambar 4 (*game Pokémon: Legends Arceus*) hanya memunculkan gambar benda yang berbeda hanya sekali, namun dilengkapi dengan keterangan jumlah item yang sama. Hal ini menunjukkan bahwa *inventory* pada gambar 4 adalah implementasi dari himpunan (*set*) dengan beberapa modifikasi. Perbedaan dengan himpunan pada matematika diskrit adalah *inventory* dilengkapi dengan keterangan jumlah item yang sama yang disimpan. Persamaan (1) berikut adalah salah satu representasi *inventory* pada gambar 4 dalam bentuk himpunan (catatan: persamaan (1) bukanlah satu-satunya representasi *inventory* yang dapat dibuat, Anda juga bisa merepresentasikannya sebagai 1 buah set dan 1 buah array, atau dengan representasi yang lain).

$$Inventory = \{(Item1,3), (Item2,3), (Item3,1), (Item4,3), (Item5,1), (Item6,6), (Item7,1), (Item8,10) \dots\} \quad (1)$$

Namun, pada gambar 5 (*Minecraft*), *inventory* pada game tersebut ada sedikit perbedaan konsep. Perbedaannya adalah di *Minecraft*, jika banyak item yang sama melebihi batas *stack* item yang sama (pada umumnya 1 *stack* = 64 *item* yang sama. Namun, ada juga yang menyatakan 1 *stack* = 16

item yang sama dan 1 *stack* = 1 *item* yang sama), maka akan dianggap sebagai *item* yang berbeda. Jadi, dapat disimpulkan bahwa *inventory* di *Minecraft* adalah implementasi dari himpunan (*multiset*) yang dimodifikasi. Persamaan (2) berikut adalah representasi dari beberapa item yang ada di gambar 5.

$$Inventory = \{(Cobblestone, 64), (Oak\ Log, 64), (Arrow, 64), (Bread, 64), (Bread, 64), \dots\} \quad (2)$$

Tulisan yang diberi warna merah pada persamaan (2) adalah elemen *item* yang diulang karena melebihi batas *stack* (64 item yang sama per *stack*). Jadi, *inventory* pada *Minecraft* adalah contoh dari *multiset*.

C. Contoh Desain Game 3: Hotbar Sirkular

Hotbar di dalam *game* adalah salah satu mekanisme yang penting dalam *game* yang menuntut kita untuk menggunakan barang. Contoh *hotbar* adalah hotbar pada *Minecraft* yang merupakan baris *item* terbawah di gambar 4, dan *weapon wheel* di *Grand Theft Auto V*.

Gambar 5: *Weapon Wheel* di *Grand Theft Auto V*.
Sumber: https://gta.fandom.com/wiki/Weapon_Wheel

Kedua contoh di atas adalah contoh dari *list* sirkular. Para pengembang *game* sebenarnya bisa saja mengimplementasikan ini sebagai *list*/*array* biasa. Namun, alasan *list* sirkuler lebih sering dipilih adalah demi kenyamanan pemain dalam melakukan navigasi di dalam *hotbar*/*weapon wheel*. Para pemain lebih menyukai mekanisme yang lebih nyaman digunakan. *List* sirkular memungkinkan pemain untuk melakukan navigasi dari akhir list ke awal list lagi dan sebaliknya hanya dengan menekan satu tombol. Hal ini sangat berbeda jika *hotbar* diimplementasikan sebagai *array* biasa, karena jika demikian, pemain harus melakukan perpindahan lagi secara satu per satu dari akhir list ke awal list dan sebaliknya. Akibatnya, *list* sirkular lebih sering digunakan sebagai implementasi dari *hotbar* dibandingkan dengan *array* biasa karena *list* sirkular akan memberikan kenyamanan lebih bagi pengguna.

Untuk mengimplementasikan hal tersebut, kita perlu teori bilangan, lebih tepatnya operasi modulo. Misalkan *hotbar* yang digunakan adalah *hotbar* pada *Minecraft*. Maka kita akan buat variabel baru sebagai berikut:

1. i adalah indeks hotbar yang sedang ditunjuk saat ini
2. N adalah panjang dari *hotbar*. Dalam kasus ini, panjang *hotbar* di *Minecraft* adalah 9.
3. i' adalah indeks *hotbar* setalah dilakukan pemindahan.
4. a adalah besar perpindahan indeks yang dilakukan oleh pemain.

Maka, perpindahan indeks *hotbar* dapat dinyatakan dengan kekongruenan (3)

$$i' \equiv (i + a)\ (mod\ N) \quad (3)$$

Berikut adalah contoh simulasi *hotbar* sirkular di *Minecraft*.

Gambar 6: *Hotbar* di *Minecraft* yang sudah diberi indeks

Sumber: diambil dari referensi [5], diedit untuk keperluan visualisasi

Misalkan indeks *hotbar* sekarang ada di indeks 4 (i = 4). Kemudian, pemain memberikan input berupa *mouse scroll* ke bawah, yang mengakibatkan perpindahan indeks. Setiap kali komputer mendeteksi adanya input *mouse scroll* ke bawah, indeks akan berpindah ke kiri.

Asumsikan input *mouse scroll* ke bawah dari pemain mengakibatkan perpindahan sebanyak 6 indeks ke kiri (a = -6). Maka, indeks yang akan ditunjuk setelah perpindahan ditunjukkan oleh sistem kekongruenan (4).

$$\begin{gathered} i' \equiv (i + a)\ (mod\ N) \\ i' \equiv (4 - 6)\ (mod\ 9) \\ i' \equiv (-2)\ (mod\ 9) \\ i' \equiv 7\ (mod\ 9) \\ i' = 7 \end{gathered} \quad (4).$$

Indeks yang baru akan menunjuk ke indeks 7 di *hotbar*.

IV. KESIMPULAN

Dari makalah ini, dapat disimpulkan bahwa banyak sekali penerapan konsep matematika diskrit yang ada di dalam desain game. Contoh konsep yang digunakan pada *game* adalah himpunan dan teori bilangan. Fungsi dari konsep matematika diskrit pada pengembangan *game* adalah sebagai bagian dari mekanisme permainan di *game* itu sendiri, menambah kenyamanan pemain dalam bermain *game*. Oleh karena itu, matematika diskrit adalah konsep yang fundamental dalam pengembangan *game* dan setiap pengembang game wajib untuk memahami konsep matematika diskrit agar *game* yang dihasilkan menarik untuk dimainkan dan disukai oleh pemain.

REFERENSI

[1] M. Korek, "Core Loop in Game Development." blog.theknightsofunity.com. https://blog.theknightsofunity.com/core-loop-in-game-development/ (accessed Dec. 9, 2023).

[2] The Pokémon Company. "Foundation and Evolution." corporate.pokemon.co.jp/en/. https://corporate.pokemon.co.jp/en/aboutus/history/ (accessed Dec. 10, 2023).

[3] "Type." bulbapedia.bulbagarden.net. https://bulbapedia.bulbagarden.net/wiki/Type (accessed Dec. 10, 2023).

[4] M.B. Koepp, "How to increase inventory space in Pokemon Legends Arceus: All Satchel upgrades & cost." dexerto.com. https://www.dexerto.com/pokemon/how-to-increase-inventory-space-in-pokemon-legends-arceus-all-satchel-upgrades-cost-1750357/ (accessed Dec. 10, 2023).

[5] A. Carnes, "5 tips for managing inventory in Minecraft." sportskeeda.com. https://www.sportskeeda.com/minecraft/5-tips-managing-inventory-minecraft (accessed Dec. 10, 2023).

[6] R. Munir, “Teori Himpunan (Bagian 1)”. https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/01-Himpunan(2023)-1.pdf, (accessed Dec. 10, 2023).
[7] R. Munir, “Teori Himpunan (Bagian 2)”. https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/02-Himpunan(2023)-2.pdf, (accessed Dec. 10, 2023).
[8] R. Munir, “Teori Bilangan (Bagian 1)”. https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/14-Teori-Bilangan-Bagian1-2023.pdf, (accessed Dec. 10, 2023).

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, 10 Desember 2023

Akbar Al Fattah / 13522036