Penerapan Procedural Content Generation Dengan Optimasi Unbounded Knapsack Problem Dalam Game Tower Defense

Implementation of Procedural Content Generation using Unbounded Knapsack Problem Optimization in a Tower Defense Game

Michael Argento[1], Hanny Haryanto[2]
[1]Program Studi Teknik Informatika, Fakultas Ilmu Komputer, Universitas Dian Nuswantoro
E-mail: [1]111202214567@mhs.dinus.ac.id, [2]hanny.haryanto@dsn.dinus.ac.id

Abstrak

Permintaan konten baru pada game tower defense sulit dipenuhi apabila wave musuh dibuat secara manual karena membutuhkan waktu, sulit diskalakan, dan berisiko menghasilkan tingkat kesulitan yang tidak konsisten. Penelitian ini menerapkan Procedural content generation dengan optimasi Unbounded knapsack problem pada sistem wave spawner dalam prototype game tower defense berjudul Castle Vanguard. Metode yang digunakan memodelkan musuh sebagai item, HP sebagai weight, difficulty value sebagai value, dan total HP per wave sebagai constraint. Permasalahan tersebut diselesaikan menggunakan Dynamic programming untuk menghasilkan komposisi musuh secara otomatis dengan prinsip exact HP fill. Sistem juga menerapkan tier-aware generation, monopoly guard, anti-repetition, dan boss wave untuk menjaga variasi komposisi musuh. Dengan demikian, penerapan Procedural content generation dengan optimasi Unbounded knapsack problem dapat digunakan untuk membentuk wave musuh yang otomatis, terukur, bervariasi, dan mudah diskalakan.

Kata kunci: Procedural content generation, Unbounded knapsack problem, Dynamic programming, Tower defense, Wave spawner

Abstract

The demand for new content in tower defense games is difficult to meet if enemy waves are generated manually, as this is time-consuming, difficult to scale, and risks producing inconsistent difficulty levels. This research applies procedural content generation with optimization of the unbounded knapsack problem to the wave spawner system in a tower defense game prototype titled Castle Vanguard. The method used models enemies as items, HP as weight, difficulty value as value, and total HP per wave as a constraint. This problem is solved using dynamic programming to automatically generate enemy compositions based on the exact HP fill principle. The system also implements tier-aware generation, monopoly guard, anti-repetition, and boss waves to maintain variation in enemy compositions. Thus, the application of procedural content generation with Unbounded Knapsack Problem optimization can be used to create enemy waves that are automatic, measurable, varied, and easily scalable.

Keywords: Procedural content generation, Unbounded knapsack problem, Dynamic programming, Tower defense, Wave spawner

1. PENDAHULUAN

Tower defense merupakan genre game strategi yang berfokus pada alokasi sumber daya dan penempatan tower untuk mempertahankan area dari serangan musuh. Dalam bentuk dasarnya, player membeli dan mengatur tower untuk menghadapi kelompok musuh yang muncul secara bertahap. Kesederhanaan mekanik dasar yang dipadukan dengan kebutuhan penyusunan strategi membuat tower defense menjadi salah satu genre yang sesuai untuk penerapan metode komputasional pada proses pembentukan konten [1].

Salah satu komponen penting dalam game tower defense adalah wave spawner. Sistem ini menentukan komposisi musuh yang muncul pada setiap wave, termasuk tipe, jumlah, dan waktu kemunculan. Apabila seluruh wave disusun secara manual, kebutuhan konten baru akan meningkatkan beban pengembangan dan menyulitkan skalabilitas. Hendrikx et al. menjelaskan bahwa produksi elemen game secara manual mahal dan sulit diskalakan ketika kebutuhan terhadap konten baru terus meningkat [2].

Procedural Content Generation (PCG) merupakan pendekatan pembentukan konten game secara algoritmik. PCG dapat digunakan untuk menghasilkan peta, level, aturan, misi, maupun elemen permainan lainnya [2], [3]. Pada sistem wave spawner, PCG memungkinkan komposisi musuh dibentuk secara otomatis. Namun, pembangkitan otomatis tetap memerlukan mekanisme kontrol karena generator harus menghasilkan konten yang memenuhi batasan yang ditetapkan oleh designer sekaligus tetap sesuai untuk dimainkan [2].

Penelitian terdahulu telah menerapkan beberapa pendekatan PCG pada game tower defense. Liu et al. membangkitkan level tower defense secara otomatis, termasuk jalur musuh, penempatan tower, dan wave, serta menghasilkan level dengan karakteristik kesulitan yang menyerupai rancangan designer [4]. Berbeda dengan pendekatan tersebut, penelitian ini membatasi ruang lingkup pada wave spawner dan menggunakan total HP musuh sebagai constraint utama pembentukan wave.

Kraner et al. menggunakan Genetic Algorithm (GA) untuk menghasilkan elemen tower defense dan memperoleh kurva kesulitan yang stabil melalui optimasi komposisi wave [5]. Dias et al. mengembangkan PCG berbasis aturan untuk menghasilkan variasi musuh melalui kombinasi atribut, tetapi peningkatan variasi juga menambah tantangan balancing [6]. Hind dan Harvey menerapkan NEAT dengan curriculum untuk menghasilkan konten dinamis yang menyesuaikan keadaan pertahanan player, sehingga aspek Dynamic Difficulty Adjustment (DDA) diterapkan pada pemilihan komposisi musuh [7].

Penelitian ini mengambil pendekatan berbeda dengan membentuk wave sebagai permasalahan Unbounded Knapsack Problem (UKP). Karakteristik unbounded sesuai dengan wave spawner karena satu tipe musuh dapat dipilih lebih dari satu kali. Dalam model penelitian, musuh direpresentasikan sebagai item, HP sebagai weight, difficulty value sebagai value, dan kapasitas HP wave sebagai constraint. Konsep tersebut sejalan dengan karakteristik UKP yang memungkinkan pemilihan banyak item dari setiap jenis selama kapasitas terpenuhi [9].

Dynamic Programming (DP) digunakan sebagai metode penyelesaian UKP. Berbeda dengan pendekatan reaktif terhadap performa player, kapasitas HP dalam penelitian ini ditentukan oleh progres wave sehingga sistem diarahkan untuk menjaga parameter pembentukan wave tetap konsisten pada setiap iterasi permainan. Selain optimasi utama, sistem menerapkan tier-aware generation, within-tier monopoly guard, anti-repetition, dan boss wave untuk mengurangi kecenderungan komposisi yang monoton.

Kontribusi penelitian ini adalah penerapan UKP-DP sebagai inti PCG pada wave spawner dengan prinsip exact HP fill, yaitu total HP musuh yang dihasilkan harus sama dengan kapasitas HP wave. Pendekatan tersebut diimplementasikan pada prototype game Castle Vanguard menggunakan Unity dan C#. Pengujian dilakukan terhadap pemenuhan constraint, variasi musuh, batas dominasi tipe musuh, serta pengalaman bermain melalui Game Experience Questionnaire (GEQ).

2. METODE PENELITIAN

2.1 Perancangan Sistem Wave Spawner

Penelitian menggunakan pendekatan Constraint-based Procedural Content Generation. Inti metode adalah membentuk komposisi musuh pada setiap wave sebagai permasalahan UKP dengan total HP sebagai kapasitas utama. UKP kemudian diselesaikan menggunakan DP untuk memperoleh kombinasi jenis dan jumlah musuh yang memenuhi kapasitas tersebut sekaligus mempertimbangkan difficulty value.

Prototype yang digunakan adalah Castle Vanguard, yaitu game tower defense 2D dengan area pertahanan berupa grid 7×7. Musuh normal dibagi menjadi tipe Basic, Fast, Regen, dan Split. Tier utama memiliki HP 10, 50, dan 200, sedangkan Split memiliki pengecualian tier 0 sebesar 5 HP ketika pecah. Boss muncul pada wave kelipatan 10 dan ditambahkan setelah komposisi musuh normal dibentuk.

Tabel 1 Pemetaan elemen game ke model UKP

Elemen game	Representasi UKP	Peran
Tipe musuh	Item	Kandidat yang dapat dipilih berulang
Health Point (HP)	Weight	Mengisi kapasitas total HP wave
Difficulty value	Value	Merepresentasikan tingkat kesulitan kandidat
Total HP wave	Constraint	Menentukan kapasitas pembentukan wave

2.2 Penentuan Kapasitas HP Wave

Kapasitas HP dilambangkan dengan Cw dan merepresentasikan total HP musuh normal yang diizinkan pada wave ke-w. Wave pertama dimulai dengan kapasitas dasar 50 HP. Rumus kapasitas awal pada skripsi dinyatakan sebagai berikut.

$$C_1 = B,\ \ B = 50 \tag{1}$$

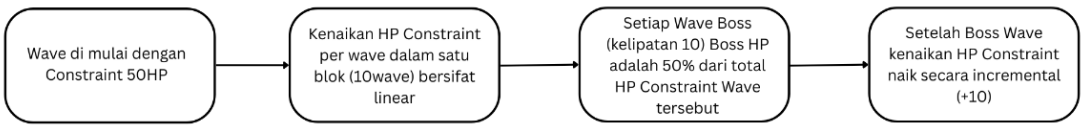
Peningkatan kapasitas dibagi berdasarkan blok 10 wave. Nomor blok kesulitan dan besar kenaikan HP pada wave ke-w dihitung dengan persamaan berikut.

$$b_w = \lfloor \frac{(w-1)}{10} \rfloor + 1,\ \ \Delta_w = 10\ \cdot b_w \tag{2}$$

Kapasitas wave berikutnya kemudian dihitung secara rekursif.

$$C_w = C_{w-1} + \Delta_w,\ \ w \geq 2 \tag{3}$$

Berdasarkan formulasi tersebut, wave 1 memiliki kapasitas 50 HP. Wave 2 sampai 10 bertambah 10 HP per wave, wave 11 sampai 20 bertambah 20 HP per wave, wave 21 sampai 30 bertambah 30 HP per wave, dan pola peningkatan dilanjutkan pada blok berikutnya.



Gambar 1 Alur Kurva Kesulitan

2.3 Pemodelan UKP dan Difficulty Value

Setelah kapasitas HP ditentukan, setiap kandidat musuh ke-i memiliki HP hi, difficulty value vi, dan jumlah kemunculan xi. Total difficulty pada wave ke-w dihitung dengan persamaan yang terdapat pada skripsi sebagai berikut.

$$D_w = \sum_{i=1}^{n} v_i x_i \quad (4)$$

Dalam formulasi penelitian, kombinasi musuh dicari agar total difficulty berada sedekat mungkin dengan target difficulty Tw, sementara total HP harus tepat sama dengan kapasitas wave. Constraint exact HP fill yang digunakan pada skripsi adalah sebagai berikut.

$$dengan\ constraint \sum_{i=1}^{n} h_i x_i = C_{w,} \quad x_i \in \mathbb{Z}_{\geq 0} \quad (5)$$

Tanda sama dengan pada constraint menunjukkan penggunaan exact HP fill. Total HP musuh yang dipilih harus sama dengan kapasitas wave, bukan hanya kurang dari atau sama dengan kapasitas. Karena xi merupakan bilangan bulat non-negatif, setiap kandidat dapat dipilih lebih dari satu kali, sesuai karakteristik UKP [9].

2.4 Penyelesaian UKP Menggunakan Dynamic Programming

Dynamic Programming memecah permasalahan menjadi submasalah dan menyimpan hasil state agar solusi dapat dibangun secara bertahap [10], [11]. Pada implementasi penelitian, DP tidak dinyatakan dalam rumus tambahan, tetapi diwujudkan melalui struktur state dua dimensi berdasarkan capacity dan difficulty. Array reachable menyimpan apakah pasangan capacity-difficulty dapat dicapai, sedangkan prevCap, prevDiff, dan lastChoice menyimpan jejak keputusan untuk proses backtracking.

Proses dimulai dengan state reachable[0,0] = true. Untuk setiap state yang dapat dicapai, sistem mengevaluasi semua kandidat musuh. Kandidat hanya diteruskan apabila penambahan weight tidak melampaui capacityUnits dan difficulty tidak melebihi batas state yang disediakan. Setelah seluruh state dihitung, sistem memilih bestDiff pada capacityUnits berdasarkan difficulty band dan jaraknya terhadap targetDifficulty. Komposisi akhir kemudian direkonstruksi dengan backtracking melalui prevCap, prevDiff, dan lastChoice.

Jika solusi pada kapasitas penuh tidak ditemukan, exactFillPossible bernilai false dan wave tidak menggunakan hasil tersebut. Pada tier-aware generation, kegagalan exact fill pada pembagian tier menyebabkan sistem kembali menggunakan BuildGlobalProceduralWave() sebagai fallback. Dengan demikian, pembentukan wave tetap memiliki jalur penyelesaian global ketika pembagian tier tidak dapat memenuhi kapasitas secara tepat.

Psuedocode 1. Proses Penyelesaian UKP

```
SolveUkpCountsExactHp(candidates, capacityUnits, targetDifficulty,
minDifficulty, maxDifficulty, useBand)
  Inisialisasi reachable, prevCap, prevDiff, lastChoice, dan unitCount
  reachable[0,0] = true
  Untuk setiap cap dari 0 sampai capacityUnits:
    Untuk setiap diff dari 0 sampai maxDifficultyPossible:
      Jika state tidak reachable, lanjutkan
      Untuk setiap kandidat i:
        Hitung nextCap dan nextDiff
        Jika masih dalam batas, simpan state dan pilihan terakhir
  Pilih bestDiff pada capacityUnits berdasarkan band dan targetDifficulty
  Lakukan backtracking untuk memperoleh counts setiap kandidat
  Tetapkan exactFillPossible = true apabila kapasitas dapat dipenuhi
  Kembalikan counts
```

2.5 Logika Statistik Musuh

Difficulty value tidak hanya berasal dari HP dasar, tetapi juga mempertimbangkan statistik dan ability kandidat. Parameter yang paling relevan terhadap proses pembentukan wave diringkas pada Tabel 2. Nilai tersebut diambil dari logika statistik musuh yang digunakan pada prototype.

Tabel 2 Parameter statistik yang digunakan dalam pembentukan wave

Parameter	Nilai	Keterangan
HP Difficulty value	Setiap 10 HP = 1 poin	Nilai kesulitan berdasarkan HP
Speed Difficulty value	Kenaikan 0,1 speed = 1 poin	Tambahan nilai pada musuh yang lebih cepat
Regen	10% Max HP per detik	Kemampuan pemulihan HP pada Regen
Effective HP	Max HP + Total HP Modifier	Memperkirakan daya tahan akibat regen atau split
Enemy Modifier	Ability atau stats tambahan	Tambahan nilai untuk Speed, Regen, Split, atau Boss
Total Difficulty value	HP Difficulty + Speed Difficulty + Enemy Modifier	Nilai akhir kesulitan kandidat
Boss HP	50% HP Constraint wave	HP boss mengikuti kapasitas wave

2.6 Tier-aware Generation dan Within-tier Monopoly Guard

UKP global dapat menghasilkan komposisi yang terpusat pada kelompok HP tertentu. Untuk mengurangi kondisi tersebut, sistem menerapkan tier-aware generation. Kapasitas HP wave dibagi menjadi budget tier 10 HP, 50 HP, dan 200 HP. Pembagian dilakukan berdasarkan fase early, mid, dan late, kemudian disesuaikan agar setiap budget tetap merupakan kelipatan HP pada tier yang bersangkutan.

Setiap tier diselesaikan menggunakan UKP secara terpisah. Kandidat tier 10 hanya berisi musuh 10 HP, tier 50 hanya berisi musuh 50 HP, dan tier 200 hanya berisi musuh 200 HP. Hasil ketiga tier kemudian digabungkan. Penyesuaian antar tipe dilakukan di dalam tier yang sama, sehingga pergantian Basic_50 menjadi Fast_50, misalnya, tidak mengubah total HP dan tidak merusak exact fill.

Within-tier monopoly guard digunakan untuk mencegah satu tipe musuh mendominasi jumlah musuh pada tier tertentu. Anti-repetition digunakan sebagai mekanisme tambahan untuk mengurangi pola wave yang terlalu berulang. Dengan kombinasi tersebut, optimasi tetap mempertahankan constraint total HP sambil memperluas variasi tipe musuh yang muncul.

2.7 Boss Wave

Boss muncul pada wave kelipatan 10. Kondisi kemunculan boss pada skripsi dinyatakan sebagai berikut.

$$w \bmod p = 0, \quad p = 10 \tag{6}$$

Boss tidak menggantikan musuh normal, tetapi ditambahkan setelah komposisi normal wave selesai dibentuk. HP boss mengikuti persentase kapasitas HP wave dengan mempertimbangkan HP minimum boss.

$$H_{boss,w} = max(H_{min}, \rho C_w) \tag{7}$$

Karena boss digabungkan dengan wave normal, total HP pada boss wave dinyatakan sebagai berikut.

$$H_{total,w} = C_w + H_{boss,w} \tag{8}$$

Pada implementasi Castle Vanguard, persentase HP boss menggunakan 50% dari HP constraint wave. Boss kemudian dimasukkan setelah jadwal spawn musuh normal selesai sehingga mekanisme boss tidak mengubah proses optimasi normal wave.

2.8 Rancangan Pengujian

Pengujian dilakukan dari sisi teknis dan pengalaman bermain. Pengujian teknis menggunakan wave 17, 60, 100, dan 123 untuk mewakili beberapa tahap perkembangan permainan. Data diperoleh dari Unity Console ketika sistem membentuk wave.

Parameter teknis meliputi capacityHP, usedHP, selisih HP, ExactFill, pembagian budget tier, batas maksimal dominasi tipe musuh, UniqueVariants, dan UniqueEnemyTypes. Sistem

dinyatakan memenuhi constraint apabila usedHP sama dengan capacityHP dan selisih HP bernilai 0.

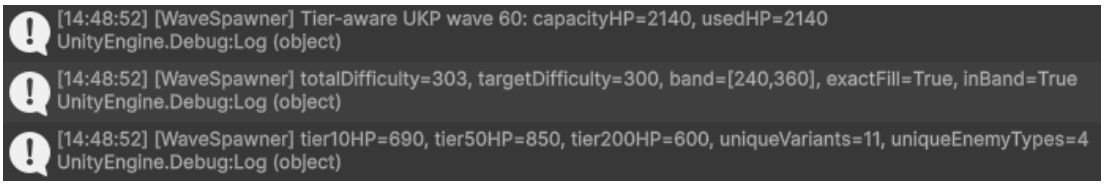
Pengujian pengalaman bermain menggunakan GEQ setelah responden menyelesaikan sesi permainan. Responden berjumlah 9 orang berusia 16-19 tahun, seluruhnya pernah memainkan tower defense, dengan playtime minimum 10 menit. Skala jawaban menggunakan nilai 1 sampai 5, dari Sangat Tidak Setuju sampai Sangat Setuju.

3. HASIL DAN PEMBAHASAN

3.1 Implementasi Pembentukan Wave UKP-DP

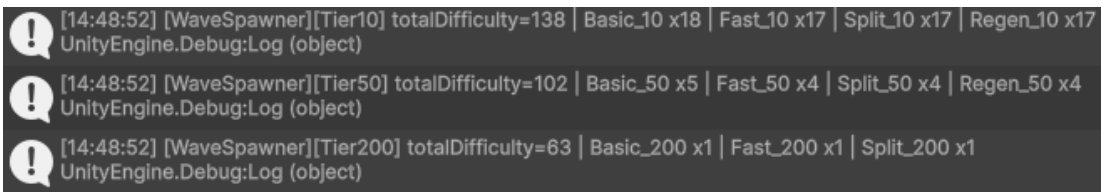
Wave spawner menjadi inti implementasi PCG pada Castle Vanguard. Ketika sebuah wave akan dibentuk, sistem menghitung hpCapacity, mengubahnya menjadi capacityUnits, mengambil kandidat yang telah terbuka pada wave tersebut, kemudian menjalankan penyelesaian UKP. Apabila useTierAwareGeneration aktif, sistem membagi budget ke tier sebelum melakukan penyelesaian UKP pada tiap kelompok.

Hasil pada wave 60 memperlihatkan capacityHP sebesar 2.140 dan usedHP sebesar 2.140. Total difficulty yang terbentuk adalah 303 dengan targetDifficulty 300 dan difficulty band [240,360]. Console menampilkan ExactFill=True dan InBand=True. Pembagian budget terdiri atas 690 HP untuk tier 10, 850 HP untuk tier 50, dan 600 HP untuk tier 200. Jumlah ketiga budget tersebut tepat 2.140 HP.



Gambar 2 Console pembentukan wave 60

Detail komposisi wave 60 menunjukkan bahwa tier 10 menghasilkan Basic_10 sebanyak 18, Fast_10 sebanyak 17, Split_10 sebanyak 17, dan Regen_10 sebanyak 17 dengan totalDifficulty 138. Tier 50 menghasilkan Basic_50 sebanyak 5 serta Fast_50, Split_50, dan Regen_50 masing-masing sebanyak 4 dengan totalDifficulty 102. Tier 200 menghasilkan Basic_200, Fast_200, dan Split_200 masing-masing sebanyak 1 dengan totalDifficulty 63. Dengan demikian, wave 60 memiliki 11 unique variants dan empat enemy types.



Gambar 3 Komposisi musuh hasil pembentukan wave 60

3.2 Pengujian Pemenuhan Constraint Total HP

Tabel 3 Hasil pengujian pemenuhan constraint total HP

Wave	Capacity HP	Used HP	Selisih HP	Exact Fill
17	280	280	0	True
60	2.140	2.140	0	True
100	5.540	5.540	0	True
123	8.230	8.230	0	True

Tabel 3 menunjukkan seluruh sampel wave memiliki usedHP yang sama dengan capacityHP dan selisih 0. Kondisi tersebut membuktikan bahwa kombinasi musuh yang dikembalikan sistem menggunakan seluruh budget HP pada wave yang diuji. Dengan kata lain, prinsip exact HP fill berhasil dipertahankan pada wave awal, menengah, lanjutan, dan wave berskala besar yang digunakan sebagai sampel.

Dari sisi perancangan, penggunaan equality constraint membuat kapasitas yang telah dihitung tidak berhenti sebagai batas maksimum saja, tetapi benar-benar menjadi jumlah HP normal yang diwujudkan menjadi unit musuh. Hal tersebut sesuai dengan tujuan penelitian untuk menjaga pembentukan wave tetap terukur berdasarkan constraint total HP.

3.3 Pengujian Batas Dominasi Tipe Musuh

Tabel 4 Hasil pengujian batas maksimal dominasi tipe musuh per tier

Wave	Tier	Total Musuh	Batas Maks/Tipe	Tipe Terbesar	Status
17	10	28	8	7	Sesuai
60	10	69	20	18	Sesuai
60	50	17	6	5	Sesuai
60	200	3	1	1	Sesuai
100	10	169	50	50	Sesuai
100	50	45	18	12	Sesuai
100	200	8	2	2	Sesuai
123	10	248	74	74	Sesuai
123	50	67	26	17	Sesuai
123	200	12	3	3	Sesuai

Seluruh tier pada sampel pengujian memenuhi batas maksimal dominasi. Pada wave 17, tier 10 memiliki 28 musuh dengan batas maksimal 8 musuh untuk satu tipe dan tipe terbesar berjumlah 7. Pada wave 60, jumlah tipe terbesar pada tier 10, 50, dan 200 masing-masing adalah 18, 5, dan 1, seluruhnya berada di bawah atau sama dengan batas yang ditentukan.

Pada wave 100 dan 123, jumlah musuh bertambah secara signifikan. Meskipun demikian, nilai tipe terbesar tetap tidak melampaui batas maksimal. Hasil tersebut menunjukkan bahwa within-tier monopoly guard tetap bekerja ketika jumlah unit meningkat dan membantu mencegah satu tipe musuh mendominasi komposisi pada tier yang sama.

3.4 Pengujian Variasi Wave

Tabel 5 Hasil pengujian variasi wave prosedural

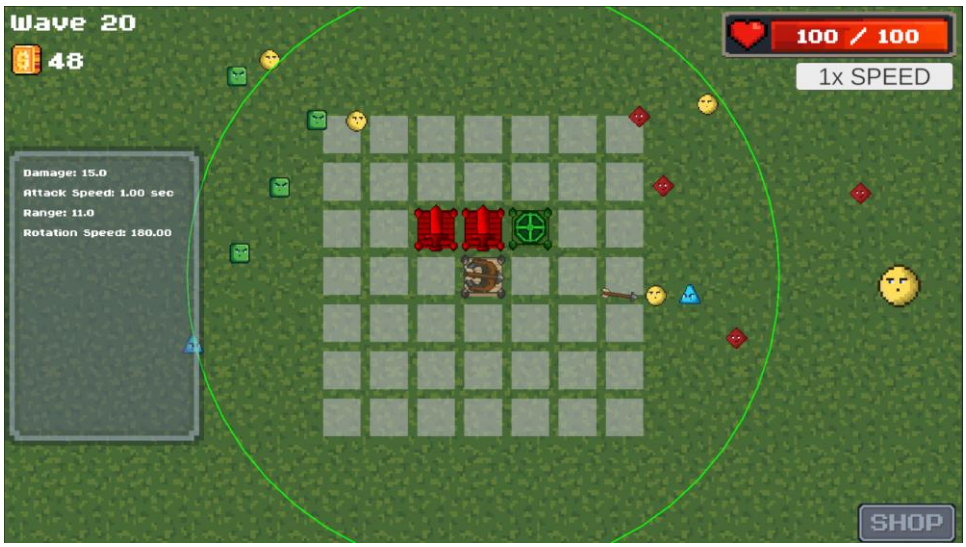
Wave	Unique Variants	Unique Enemy Types
17	4	4
60	11	4
100	12	4
123	12	4

Wave 17 memiliki empat unique variants karena hanya tier 10 yang digunakan. Pada wave 60, unique variants meningkat menjadi 11 ketika tier 10, 50, dan 200 mulai digunakan secara bersama. Wave 100 dan 123 menghasilkan 12 unique variants, yang berarti seluruh kombinasi tipe dan tier normal yang tersedia pada konfigurasi tersebut muncul dalam komposisi wave.

Jumlah unique enemy types tetap empat pada seluruh sampel, yaitu Basic, Fast, Split, dan Regen. Peningkatan variasi pada wave yang lebih tinggi berasal dari kombinasi tipe dengan tier HP. Hasil ini mendukung fungsi tier-aware generation dan monopoly guard sebagai lapisan variasi di atas optimasi UKP.

3.5 Implementasi Boss Wave

BossSpawner memeriksa apakah nomor wave merupakan kelipatan bossEveryNWaves. Dengan nilai interval 10, boss muncul pada wave 10, 20, 30, dan seterusnya. Boss dipilih dari katalog boss yang memenuhi minWave, kemudian HP-nya dihitung dari waveBudgetHp dan persentase boss. Boss wave digabungkan dengan normal wave sehingga boss mulai muncul setelah jadwal spawn normal selesai.



Gambar 4 Tampilan boss wave pada wave 20

Pemisahan normal wave dan boss wave menjaga proses UKP tetap berfokus pada constraint HP musuh normal. Boss menjadi tambahan tantangan yang mengikuti perkembangan kapasitas wave, bukan komponen yang mengurangi budget normal wave.

3.6 Hasil Game Experience Questionnaire

Formulir GEQ diisi oleh 9 responden yang telah mencoba game Castle Vanguard. Setiap pertanyaan menggunakan skala 1 sampai 5 dengan ketentuan sebagai berikut:

1. menunjukkan Sangat Tidak Setuju
2. menunjukkan Tidak Setuju
3. menunjukkan Netral
4. menunjukkan Setuju
5. menunjukkan Sangat Setuju

6. Tabel 6 Rekap hasil GEQ

No	Pertanyaan	1	2	3	4	5	Keterangan dominan
1	Gameplay Menarik	0	0	1	6	2	8 setuju/sangat setuju
2	Mudah Dimainkan	0	0	0	4	5	9 setuju/sangat setuju
3	Merasa Bosan	4	2	3	0	0	6 tidak/sangat tidak setuju
4	Lupa Waktu & Sekitar	6	3	0	0	0	9 tidak/sangat tidak setuju
5	Merasa Senang	0	0	2	6	1	7 setuju/sangat setuju
6	Merasa Berkesan	0	0	1	5	3	8 setuju/sangat setuju
7	Merasa Tertantang	0	0	1	6	2	8 setuju/sangat setuju
8	Merasa Frustrasi	7	2	0	0	0	9 tidak/sangat tidak setuju
9	Merasa Tersinggung	9	0	0	0	0	9 sangat tidak setuju
10	Menyukai Game	0	0	0	8	1	9 setuju/sangat setuju

Hasil GEQ menunjukkan 88,9% responden menilai gameplay menarik, 100% menyatakan game mudah dimainkan, 77,8% merasa senang, 88,9% merasa gameplay berkesan, 88,9% merasa tertantang, dan 100% menyukai game. Pada aspek negatif, 66,7% responden tidak merasa bosan, seluruh responden tidak merasa frustrasi, dan seluruh responden sangat tidak setuju bahwa game membuat mereka merasa tersinggung.

Aspek yang memperoleh respons paling rendah adalah pernyataan lupa waktu dan sekitar. Seluruh responden memberikan jawaban tidak setuju atau sangat tidak setuju. Berdasarkan pembahasan pada skripsi, hasil tersebut menunjukkan bahwa tingkat imersi masih perlu ditingkatkan melalui pengembangan variasi tower, sistem upgrade, feedback visual dan audio, serta progression.

Secara keseluruhan, pengujian teknis dan GEQ menunjukkan dua sisi hasil penelitian. Dari sisi sistem, UKP-DP mampu mempertahankan exact fill dan pembatasan komposisi pada sampel wave yang diuji. Dari sisi player, game memperoleh respons positif pada ketertarikan, kemudahan bermain, tantangan, dan tingkat kesukaan. Pengujian GEQ tidak digunakan sebagai pembuktian matematis UKP, tetapi sebagai evaluasi penerimaan pengalaman bermain setelah mekanisme wave prosedural diterapkan.

3.7 Pembahasan Metode UKP-DP

Pemodelan wave sebagai UKP membuat batas desain utama dapat dinyatakan secara eksplisit melalui total HP. Berbeda dengan wave manual, jumlah dan tipe musuh tidak disimpan sebagai daftar statis untuk setiap gelombang. Sistem menghitung kapasitas, memilih kandidat, mencari kombinasi melalui DP, dan merekonstruksi komposisi yang memenuhi kapasitas tersebut.

Hasil exact fill pada empat sampel menunjukkan bahwa kapasitas 280, 2.140, 5.540, dan 8.230 dapat dikonversi seluruhnya menjadi komposisi musuh. Namun, exact fill saja tidak menjamin variasi. Karena itu, tier-aware generation dan monopoly guard berperan penting untuk membatasi hasil optimasi agar tidak terlalu homogen. Data variasi dan dominasi menunjukkan

bahwa seluruh empat enemy types tetap muncul dan batas dominasi dapat dipenuhi pada tier yang diuji.

Pendekatan ini memiliki karakter berbeda dengan penelitian berbasis DDA [7]. Sistem tidak mengubah kapasitas berdasarkan performa player secara real-time, melainkan mengikuti kurva constraint yang telah ditentukan. Dibandingkan pendekatan GA pada penelitian terkait [5], penelitian ini menggunakan DP untuk menyelesaikan state kapasitas-difficulty secara deterministik pada konfigurasi kandidat yang diberikan. Penelitian ini tidak melakukan eksperimen komparatif langsung terhadap GA atau DDA, sehingga perbandingan tersebut dibatasi pada karakter pendekatan yang dibahas dalam landasan teori skripsi.

Keterbatasan implementasi yang teridentifikasi pada skripsi antara lain jumlah tower ofensif yang masih terbatas dan musuh Regen yang memiliki effective HP tinggi. Selain itu, pengujian teknis difokuskan pada empat wave sampel dan GEQ melibatkan sembilan responden. Keterbatasan tersebut menjadi dasar pengembangan sistem dan pengujian pada penelitian berikutnya.

4. KESIMPULAN DAN SARAN

Penerapan Procedural Content Generation dengan optimasi UKP yang diselesaikan menggunakan DP berhasil digunakan pada wave spawner game tower defense Castle Vanguard. Sistem memodelkan musuh sebagai item, HP sebagai weight, difficulty value sebagai value, dan total HP wave sebagai constraint. Pembentukan wave menggunakan prinsip exact HP fill sehingga total HP komposisi musuh harus sama dengan kapasitas HP yang telah ditentukan.

Pengujian pada wave 17, 60, 100, dan 123 menghasilkan usedHP yang sama dengan capacityHP dan selisih 0 pada seluruh sampel. Tier-aware generation dan within-tier monopoly guard juga mempertahankan variasi komposisi tanpa melanggar batas dominasi tipe musuh pada tier yang diuji. Unique variants meningkat dari 4 pada wave 17 menjadi 11 pada wave 60 dan 12 pada wave 100 serta 123. Boss wave berhasil diterapkan pada kelipatan 10 dengan HP yang mengikuti kapasitas wave.

Hasil GEQ terhadap 9 responden menunjukkan respons positif pada ketertarikan gameplay, kemudahan bermain, kesenangan, tantangan, dan tingkat kesukaan terhadap game. Dengan hasil tersebut, PCG berbasis UKP-DP dapat digunakan untuk membentuk wave musuh secara otomatis, terukur, bervariasi, dan mudah diskalakan pada prototype yang dikembangkan.

Pengembangan selanjutnya dapat menambahkan variasi tower ofensif, fitur move dan remove/sell tower pada build phase, sistem upgrade tower, serta fungsi menu Options. Dari sisi penelitian algoritma, pengujian dapat diperluas pada lebih banyak wave dan dibandingkan secara langsung dengan metode pembentukan wave lain agar karakteristik performa dan kualitas komposisi dapat dianalisis lebih lanjut.

DAFTAR PUSTAKA

[1] P. Avery, J. Togelius, E. Alistar, and R. P. van Leeuwen, "Computational intelligence and tower defence games," in Proceedings of the 2011 IEEE Congress on Evolutionary Computation (CEC), 2011, pp. 1084–1091.

[2] M. Hendrikx, S. Meijer, J. Van Der Velden, and A. Iosup, "Procedural content generation for games: A survey," ACM Transactions on Multimedia Computing, Communications, and Applications, vol. 9, no. 1, Article 1, 2013.

[3] R. Lara-Cabrera, M. Nogueira-Collazo, C. Cotta, and A. J. Fernández-Leiva, "Procedural content generation for real-time strategy games," International Journal of Interactive Multimedia and Artificial Intelligence, vol. 3, no. 2, pp. 40–48, 2015.

[4] S. Liu, C. Li, Y. Li, H. Ma, X. Hou, Y. Shen, et al., "Automatic generation of tower defense levels using PCG," in Proceedings of the 14th International Conference on the Foundations of Digital Games (FDG '19), 2019, pp. 1–9.

[5] V. Kraner, I. Fister Jr., and L. Brezočnik, "Procedural content generation of custom Tower defense game using genetic algorithms," in New Technologies, Development and Application IV, vol. 233, 2021, pp. 493–503.

[6] D. M. P. L. Dias, J. P. Sousa, B. Barroso, and I. M. B. Magalhães, "A novel procedural content generation algorithm for tower defense games," in Proceedings of the 6th International Conference on Algorithms, Computing and Systems (ICACS 2022), 2022, Article 9, pp. 1–7.

[7] D. Hind and C. Harvey, "Using a NEAT approach with curriculums for dynamic content generation in video games," Personal and Ubiquitous Computing, vol. 28, no. 4, pp. 629–641, 2024.

[8] U. Pferschy, G. Nicosia, A. Pacifici, and J. Schauer, "On the Stackelberg knapsack game," European Journal of Operational Research, vol. 291, no. 1, pp. 18–31, 2021.

[9] H. Kellerer, U. Pferschy, and D. Pisinger, Knapsack Problems. Berlin: Springer, 2004.

[10] R. Bellman, Dynamic Programming. Princeton, NJ: Princeton University Press, 1957.

[11] S. P. Bradley, A. C. Hax, and T. L. Magnanti, Applied Mathematical Programming. Reading, MA: Addison-Wesley, 1977.

[12] H. Becker and L. S. Buriol, "An empirical analysis of exact algorithms for the unbounded knapsack problem," European Journal of Operational Research, vol. 277, no. 1, pp. 84–99, 2019.