

Implementasi Algoritma Program Dinamis untuk Memaksimalkan Pendapatan Banana Farm pada Permainan Bloons TD 6 dalam Batasan Waktu Boss Event

Moreno Syawali Ganda Sugita - 13524096

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung, Jalan Ganesha 10 Bandung

E-mail: moreno.syawali2910@gmail.com , 13524096@std.stei.itb.ac.id

Abstract— Makalah ini membahas penerapan Program Dinamis dan algoritma *Beam Search* untuk mengoptimalkan panduan investasi (*build order*) menara *Banana Farm* pada permainan *Bloons TD 6*. Mekanik ekonomi permainan yang melibatkan pendapatan pasif, biaya yang bervariasi antar tingkat kesulitan, serta bunga majemuk dari *Monkey Bank* menciptakan ledakan kombinatorial pada *state space*. Hal ini menyebabkan pencarian rute investasi optimal tidak dapat diselesaikan dengan pendekatan *greedy* konvensional. Sebagai solusi, pemodelan sistem dibangun menggunakan pendekatan waktu diskrit, di mana Program Dinamis mengevaluasi setiap keadaan berdasarkan fungsi heuristik Kekayaan Bersih (*Net Worth*). Untuk menjaga efisiensi memori dan komputasi, metode ini dipadukan dengan pemangkasan *Beam Search* yang membatasi memori hanya pada 2000 keadaan terbaik di setiap rondonya. Pengujian dilakukan menggunakan integrasi data harga faktual secara langsung (*data ingestion*). Hasil simulasi pada tingkat kesulitan *Impoppable* (100 ronde) menunjukkan bahwa algoritma berhasil menangani kompleksitas *nested state* (saldo bank) secara akurat dan menemukan rute paling efisien, menghasilkan perputaran uang kotor sebesar \$351.431 dengan total pengeluaran investasi sebesar \$167.040 dalam waktu komputasi singkat. Dari hasil pengujian, pendekatan ini mampu menangani permasalahan optimasi ekonomi bertahap dengan cukup baik.

Keywords—*Bloons TD 6; Banana Farm; dynamic programming; beam search; combinatorial explosion; state space optimization*

I. LATAR BELAKANG

Perkembangan industri permainan video dalam beberapa tahun terakhir tidak lagi hanya berfungsi sebagai hiburan, tetapi juga dapat digunakan sebagai simulasi pengambilan keputusan. Salah satu genre yang secara intensif menguji ketajaman berpikir pemain adalah *tower defense*, yang dapat diambil kasusnya oleh permainan *Bloons TD 6* (BTD6) [1]. Di dalam permainan ini, kemenangan tidak hanya ditentukan oleh penempatan menara pertahanan yang taktis, melainkan juga oleh kemampuan manajemen ekonomi yang presisi melalui aset penghasil *money* bernama *Banana Farm* [2]. Tantangan ini

mencapai puncaknya pada mode Boss Event, di mana pemain dihadapkan pada musuh berkekuatan masif yang muncul pada milestone ronde tertentu, seperti ronde 40, 60, 80, hingga 100. Batasan waktu ini memaksa pemain untuk mengumpulkan modal dan keuntungan sebesar-besarnya dalam durasi yang sangat terbatas agar mampu mengkonversi aset ekonomi menjadi sistem pertahanan yang memadai.



Gambar 1.1. Jenis Peningkatan Banana Farm

Meskipun aspek ekonomi memegang peranan krusial, sebagian besar pemain masih mengandalkan intuisi atau pendekatan heuristik *greedy* lokal dalam menentukan urutan pembangunan dan *upgrade* dari *Banana Farm*. Pendekatan *greedy* konvensional cenderung memprioritaskan opsi yang memberikan pengembalian modal (*Return on Investment*) tercepat pada satu ronde spesifik tanpa mempertimbangkan akumulasi pertumbuhan kekayaan secara global hingga ronde target [4]. Padahal, mekanik permainan BTD6 melibatkan struktur biaya dan pendapatan yang dinamis, bervariasi secara khusus berdasarkan tingkat kesulitan, serta memiliki fitur kompleks seperti bunga majemuk pada *Monkey Bank* dan skema nilai jual kembali aset sebesar 70%. Karena variabel-variabel saling bergantung, jumlah kemungkinan *state* bertambah sangat cepat dari ronde ke ronde. Akibatnya, pencarian kombinasi aksi investasi yang menghasilkan keuntungan absolut selalu dihadapkan pada ancaman *combinatorial explosion* yang mustahil diselesaikan secara manual maupun melalui metode pencarian buta.

Sebagai solusi atas permasalahan optimasi bertahap (*multi-stage optimization*) tersebut, dapat memanfaatkan metode *Dynamic Programming* [3, 4]. Pendekatan ini cukup efektif karena masalah dapat dipecah menjadi bagian-bagian yang lebih kecil. Untuk mengatasi kendala densitas ruang keadaan akibat variasi sisa *money* dan akumulasi bunga bank, strategi Program Dinamis dalam penelitian ini dikombinasikan dengan metode Beam Search sebagai skema *pruning* berbasis fungsi heuristik kekayaan bersih [5]. Melalui pemodelan matematis berbasis akhir ronde terhadap parameter riil permainan BTDD6, makalah ini ditujukan untuk mengimplementasikan dan menganalisis sebuah program pencari rute optimal yang mampu menyusun panduan investasi Banana Farm paling efisien, dengan harapan dapat memberikan gambaran penerapan strategi algoritma pada optimasi sistem ekonomi permainan.

II. DASAR TEORI DAN PEMODELAN

A. Permainan Bloons TD 6 dan Mekanik Banana Farm

Bloons TD 6 (BTDD6) adalah permainan *tower defense* di mana pemain harus menempatkan menara untuk menghancurkan gelombang musuh (*bloons*) yang datang pada setiap ronde. Sumber daya utama dalam permainan ini adalah *money*, yang digunakan untuk membeli dan *upgrade* menara. Selain pendapatan pasif alami per ronde, pemain dapat membangun Banana Farm, sebuah menara khusus yang tidak memiliki daya serang namun berfokus pada penghasilan *money* [2].

Banana Farm memiliki tiga jalur *upgrade* (Path 1, 2, dan 3) yang masing-masing dapat ditingkatkan hingga *tier* 5. Sesuai aturan *cross-pathing* permainan, jika satu jalur telah mencapai *tier* 3 atau lebih, satu jalur lainnya hanya dibatasi hingga *tier* 2, dan jalur ketiga akan terkunci [2]. Setiap jalur menawarkan karakteristik ekonomi yang berbeda:

1. Path 1: Menghasilkan pendapatan dalam jumlah besar secara langsung setiap akhir ronde.
2. Path 2: Mengubah Farm menjadi Monkey Bank yang tidak langsung memberikan *money*, melainkan menyimpan pendapatan di dalam bank dan memberikan bunga majemuk sebesar 15% setiap akhir ronde hingga mencapai kapasitas maksimal [1, 2].
3. Path 3: Marketplace yang secara konsisten menghasilkan *money* secara otomatis tanpa perlu interaksi manual, dengan nilai jual yang lebih menguntungkan.

Nilai jual (*sell value*) sebuah Farm secara standar adalah 70% dari total *money* yang telah diinvestasikan pada Farm tersebut [1], ditambah dengan sisa saldo utuh jika Farm tersebut adalah sebuah Monkey Bank. Penentuan kapan harus menabung, membeli Farm baru, melakukan *upgrade*, mencairkan bank (*collect*), atau menjual aset (*sell*) merupakan permasalahan optimasi yang sangat kompleks.

B. Dynamic Programming

Program Dinamis digunakan untuk menyelesaikan masalah optimasi dengan cara memecahnya menjadi sub-masalah [4].

Terdapat dua karakteristik utama agar sebuah masalah dapat diselesaikan dengan Program Dinamis:

1. Struktur Optimal: Solusi optimal dari masalah keseluruhan mengandung solusi optimal dari sub-masalahnya.
2. Sub-masalah Tumpang Tindih: Masalah yang sama dievaluasi berulang kali. Untuk menghindari komputasi yang redundan, Program Dinamis menyimpan hasil komputasi sub-masalah di dalam struktur memori (memoisasi pada pendekatan Top-Down atau tabulasi pada Bottom-Up) [3, 4].

Dalam konteks simulasi ekonomi, jika algoritma menemukan dua rute keputusan berbeda yang menghasilkan formasi aset yang sama persis di ronde yang sama, Program Dinamis hanya akan menyimpan rute yang menyisakan *money* paling banyak dan membuang rute lainnya.

C. Combinatorial Explosion dan Beam Search

Implementasi Program Dinamis murni pada BTDD6 dihadapkan pada masalah *combinatorial explosion*. Karena variasi sisa *money* akibat penambahan pendapatan tetap, perhitungan persen diskon, dan bunga Monkey Bank, nilai *money* menjadi sangat bervariasi [4]. Komputer akan menganggap dua keadaan dengan *money* \$1000 dan \$1001 sebagai dua *state* yang berbeda, sehingga memoisasi hampir tidak pernah terpakai secara efektif.

Untuk mengatasi hal ini, penulis mengintegrasikan Program Dinamis dengan algoritma Beam Search. Beam Search adalah varian dari Breadth-First Search (BFS) yang mengeksplorasi graf dengan hanya memperluas simpul (*node*) yang paling optimal [5]. Algoritma ini menggunakan parameter Beam Width untuk membatasi jumlah *state* yang disimpan di memori pada setiap level pencarian. Pemilihan *state* terbaik dievaluasi menggunakan fungsi heuristik. Pendekatan campuran ini memangkas cabang-cabang keputusan yang buruk, memastikan program tidak kehabisan memori meskipun melakukan komputasi hingga ronde 100.

D. Pemodelan Matematis Sistem Ekonomi

Untuk dapat diproses oleh komputer, permainan divisualisasikan ke dalam model waktu per ronde dengan persamaan matematis formal. Keadaan permainan (*State*) pada ronde ke-*i* didefinisikan sebagai *tuple*:

$$State = (i, C, F)$$

Di mana:

- *i* : Ronde permainan saat ini ($1 \leq i \leq TargetRonde$).
- *C* : Sisa *money* yang dimiliki pengguna.
- *F* : Representasi himpunan *Banana Farm* yang dimiliki (maksimal 3 *Farm*). Setiap elemen $F_k \in F$ menyimpan informasi *tier upgrade* dan saldo internal bank.

Pada setiap pergantian ronde, sistem mengevaluasi himpunan aksi yang mungkin dari keputusan yang meliputi: Skip, Buy_Farm, Upgrade_Path, Collect_Bank, dan Sell_Farm.

Fungsi relasi rekurensi transisi untuk Program Dinamis dimodelkan sebagai [3]

$$DP(i, C, F) = \max(DP(i + 1, C_{baru}, F_{baru}))$$

Perubahan *money* (C_{baru}) dipengaruhi oleh biaya dari aksi ($Cost(d)$), pendapatan pasif alami per ronde ($P(i)$), dan total pendapatan aktif dari seluruh Farm ($W(F)$). Sementara itu, heuristik yang digunakan oleh Beam Search untuk memilih B state terbaik adalah fungsi Nilai Kekayaan Bersih (Net Worth), yang dirumuskan sebagai:

$$NetWorth = C_{baru} + \sum_{k=1}^{|F|} SellValue(F_k)$$

Dengan mendefinisikan *base case* pada ronde target munculnya *Boss Event*, algoritma akan merunut balik (*backtracking*) jejak keputusan (*history*) untuk menemukan urutan aksi (*build order*) optimal.

III. IMPLEMENTASI ALGORITMA

Pengembangan *engine* simulasi optimasi Banana Farm ini diimplementasikan menggunakan bahasa pemrograman Python. Python dipilih karena memudahkan pengelolaan struktur data seperti *tuple* dan *dictionary*. Arsitektur program dibagi menjadi beberapa subsistem utama yang bekerja secara sekuensial: ekstraksi data, pemrosesan transisi state tingkat ronde, pemangkasan heuristik, dan kalkulasi evaluasi ekonomi.

A. Integrasi Data Faktual

Agar program mensimulasikan lingkungan ekonomi yang identik dengan permainan asli, sistem tidak menggunakan perhitungan matematis absolut untuk penentuan harga, melainkan melakukan pembacaan data dinamis langsung dari berkas CSV) File CSV berisi metrik Base Cost dan matriks Upgrade Cost untuk seluruh jalur pembaruan (Path 1, Path 2, Path 3) pada empat tingkat kesulitan utama (Easy, Medium, Hard, Impoppable).

Tabel 3.1. Harga Beli Dasar Banana Farm berdasarkan Tingkat Kesulitan

Tingkat Kesulitan	Harga Beli
Easy	\$1,060
Medium	\$1,250
Hard	\$1,350
Impoppable	\$1,500

Tabel 3.2. Matriks Harga Upgrade Banana Farm Berdasarkan Jalur dan Tier

Path	Tier	Easy	Medium	Hard	Impoppable
1	1	425	500	540	600
	2	510	600	650	720
	3	2550	3000	3240	3600
	4	16150	19000	20520	22800
	5	97750	115000	124200	138000
2	1	255	300	325	360

	2	680	800	865	960
	3	3100	3650	3940	4380
	4	6120	7200	7775	8640
	5	85000	100000	108000	120000
3	1	210	250	270	300
	2	340	400	430	480
	3	2295	2700	2915	3240
	4	12750	15000	16200	18000
	5	59500	70000	75600	84000

Modul *ingestion* menggunakan fungsi pembaca yang dilengkapi dengan *error handling* dan *data cleansing* untuk mengonversi string numerik yang mengandung koma atau teks deskriptif (seperti "Tier 1") menjadi tipe data *integer* murni. Data ini kemudian dipetakan ke dalam struktur *dictionary* dengan kunci berupa tuple (Top, Mid, Bot, Mode), yang memungkinkan akses pencarian harga dalam kompleksitas waktu konstan $O(1)$ selama simulasi berlangsung.

B. Representasi State dan Penggabungan Cabang

Pencarian rute optimal diimplementasikan melalui *level-by-level traversal*. Pada setiap ronde, sistem menyimpan kumpulan state ke dalam struktur *array current level*. Setiap state direpresentasikan sebagai *immutable tuple* yang berisi sisa *money*, rekam jejak pengeluaran, riwayat keputusan, serta *array* yang memuat konfigurasi Farm saat ini.

```
def add_future_state(f_cash, f_farms, f_spent, f_acts):  
  
    # Mengurutkan tuple agar formasi Farm menjadi  
    state yang unik dan konsisten  
  
    f_farms = tuple(sorted(f_farms))  
  
    # DP: Jika state belum ada, atau rute baru  
    memiliki sisa cash lebih besar, simpan/ganti state  
  
    if f_farms not in next_level_dict or f_cash >  
    next_level_dict[f_farms][0]:  
  
        next_level_dict[f_farms] = (f_cash,  
        f_spent, f_acts)
```

Untuk mengoptimalkan penggunaan memori, riwayat keputusan tidak lagi disimpan dalam bentuk *array* atau *list* dinamis yang disalin secara utuh pada setiap iterasi transisi ronde ($O(N)$). Sebagai gantinya, riwayat diimplementasikan melalui struktur *Functional Linked List*. Setiap *state* memori hanya menyimpan *pointer* berupa aksi saat ini beserta *pointer* referensi ke *tuple history* pada *state* sebelumnya. Ketika simulasi menyentuh *base case* di ronde terakhir, program cukup melakukan *backtracking* dari *node* pemenang untuk merangkai rute secara efisien dalam ruang $O(B)$.

Untuk memenuhi karakteristik *overlapping subproblems* pada Program Dinamis, susunan *Farm* selalu diurutkan (*sorted*) sebelum dimasukkan ke dalam memori. Jika program menemukan dua atau lebih rute keputusan pada konfigurasi *Farm* yang identik di ronde yang sama, sistem melakukan penggabungan cabang menggunakan *dictionary* *next_level_dict*. Sistem mengevaluasi dan hanya menyimpan

state yang memiliki nilai uang tunai (C) paling optimal, sedangkan cabang dengan nilai yang inferior secara otomatis dibuang.

C. Pemangkasan Beam Search dan Evaluasi Heuristik

Pembangkitan seluruh permutasi tindakan pada setiap ronde meliputi evaluasi untuk mempertahankan (*skip*), membeli *Farm* baru, melakukan *upgrade*, mengumpulkan saldo bank, hingga menjual aset. Kompleksitas ini memicu ledakan ruang keadaan. Oleh karena itu, diterapkan algoritma *Beam Search* yang dikombinasikan dengan injeksi *domain knowledge* berupa Late-Game Action Pruning.

Late-Game Action Pruning berfungsi untuk memblokir aksi investasi (pembelian *Farm* baru atau *upgrade*) secara spesifik pada dua ronde terakhir simulasi. Secara matematis, *Return on Investment* (ROI) dari aset berskala makro tidak akan pernah tercapai dalam sisa durasi yang sangat singkat tersebut. Dengan memotong paksa cabang keputusan investasi di penghujung permainan, *branching factor* dapat direduksi secara signifikan untuk memberikan ruang alokasi memori penuh pada manuver pencairan (*Collect*) dan likuidasi (*Sell*).

Setelah seluruh *state* untuk ronde berikutnya dibangkitkan, sistem menerapkan fungsi heuristik *Net Worth* untuk mengukur nilai objektif dari setiap keadaan, dirumuskan sebagai:

$$NetWorth = C + \sum_{k=1}^n SellValue(F_k)$$

```
# Pemangkasan Beam Search per Tingkat Ronde
scored_states = []

for f_tuple, (c, spent, history) in
next_level_dict.items():

    # Menghitung fungsi heuristik Net Worth (Uang
    tunai + Nilai likuidasi bangunan)

    net_worth = c + sum(get_sell_value(farm,
mode_name) for farm in f_tuple)

    scored_states.append((net_worth, c, f_tuple,
spent, history))

# Sorting secara descending berdasarkan Net Worth
tertinggi

scored_states.sort(reverse=True, key=lambda x:
x[0])

# Memangkas semua state yang berada di luar
batas beam_width

current_level = [(c, f, spent, h) for nw, c, f,
spent, h in scored_states[:beam_width]]
```

Di mana *SellValue* mengevaluasi nilai jual bangunan fisik (*Farm*) dengan rasio dinamis berdasarkan jalurnya, ditambah sisa saldo yang belum dicairkan pada *Monkey Bank*. Seluruh *state* kemudian diurutkan secara *descending* berdasarkan heuristik ini. Sistem memangkas seluruh kandidat kecuali sejumlah B kandidat terbaik, di mana parameter lebar lintasan (*Beam Width*) ditetapkan sebesar $B = 2000$.

D. Formulasi Metrik Laporan Ekonomi

Pada saat pencarian mencapai *base case* atau target ronde maksimal (*TargetRonde*), *state* yang menduduki peringkat heuristik teratas akan diekstraksi sebagai solusi global. Untuk menyajikan analisis yang komprehensif, program melakukan perhitungan akuntansi ekonomi terhadap jejak *state* tersebut.

Metrik yang dihitung mencakup *Total Spent*, *Remaining Cash*, dan *Final Liquidation*. Selanjutnya, program merumuskan Keuntungan Bersih dan Total Pendapatan Kotor dengan kalkulasi matematis:

$$GrossRevenue = NetWorth_{akhir} + TotalSpent - ModalAwal$$

$$NetProfit = NetWorth_{akhir} - ModalAwal$$

Laporan akhir yang dihasilkan merangkum keseluruhan putaran transaksi ekonomi secara utuh, membuktikan validitas rute optimal yang dipilih oleh algoritma pemangkasan.

E. Penanganan Bunga Majemuk pada Monkey Bank

Untuk mengakomodasi kombinasi percabangan *upgrade* yang bersilangan (*cross-pathing*) hingga Tier 5, sistem ekonomi *Farm* tidak dievaluasi menggunakan basis data tabel statis, melainkan melalui fungsi prosedural dinamis *evaluate_farm_economy()*. Pendekatan ini memungkinkan *engine* simulasi untuk mendeteksi berbagai perubahan metrik secara simultan, seperti peningkatan produksi unit pisang pada Path 1, injeksi dana pasif sebesar \$4.000 pada Path 3 tingkat 5 (*Monkey Wall Street*), hingga penyesuaian *sell ratio* dari 70% menjadi 80% apabila struktur memiliki fitur *Salvage* (0-0-2).

Tantangan komputasi tertinggi pada modul ini adalah mekanik spesifik dari Path 2 tingkat 3 ke atas (*Monkey Bank*). Berbeda dengan pendapatan konvensional yang langsung masuk ke dompet pemain (C), *Monkey Bank* menahan uang tersebut sebagai variabel *nested state*. Untuk menanganinya, representasi elemen pada algoritma dikonstruksi menjadi *tuple* 4-dimensi: (Top, Mid, Bot, BankBalance).

Jika program mendeteksi *Farm* dengan nilai *Mid* ≥ 3 , algoritma memproses saldo internal dengan persamaan:

$$Saldo_{baru} = \min(KapasitasMaks, [(Saldo_{lama} + PendapatanDasar) \times SukuBunga])$$

Penanganan *nested state* ini memaksa program untuk mengevaluasi dua opsi cabang independen tambahan pada *Beam Search*, yaitu tindakan Pencairan (*Collect*) utuh ke uang global dan me-reset nilai bank menjadi nol, serta tindakan Likuidasi (*Sell*) yang menjual wujud bangunan sekaligus mengembalikan 100% saldo kas yang mengendap di dalamnya.

```
# Evaluasi aksi penambahan uang atau perhitungan
bunga untuk setiap Farm

for f in farms:

    t, m, b, bank = f

    # Pendapatan non-bank langsung masuk ke
    variabel cash pemain

    new_cash += INCOME_TABLE.get((t, m, b), 0)
```

```
# Logika Nested State: Menghitung bunga
majemuk 15% jika Path 2 >= Tier 3

if m >= 3:
    new_bank = min(BANK_CAPACITY, int((bank +
    BANK_INCOME) * INTEREST_RATE))
else:
    new_bank = bank

new_farms.append((t, m, b, new_bank))
```

Implikasi dari *nested state* ini menuntut program untuk menyediakan dua opsi cabang keputusan tambahan yang dievaluasi secara independen oleh Beam Search:

1. Tindakan Pencairan (*Collect*): Memindahkan secara utuh *Saldo_{baru}* ke dalam *money* global (*C*), kemudian me-reset parameter saldo bank pada *tuple* tersebut menjadi nol.
2. Tindakan Likuidasi (*Sell*): Menjual bangunan yang mengharuskan algoritma mengkalkulasi 70% dari nilai Farm, ditambah pengembalian 100% dari saldo internal yang belum dicairkan.

Penanganan terpisah ini memastikan bahwa algoritma Program Dinamis dapat secara akurat menilai kapan waktu yang paling tepat secara matematis untuk membiarkan bunga majemuk berkembang, dan kapan harus mencairkannya demi modal ekspansi infrastruktur ekonomi baru.

F. Penanganan Bunga Majemuk pada Monkey Bank

Salah satu indikator keberhasilan dari perancangan strategi algoritma adalah efisiensi komputasionalnya, yang dievaluasi melalui kompleksitas waktu asimptotik dan kompleksitas ruang. Pada implementasi murni Breadth-First Search (BFS) di ruang keadaan permainan ini, kompleksitas waktu akan tumbuh secara eksponensial menjadi $O(A^R)$, di mana A adalah rata-rata faktor percabangan (*branching factor*) dari aksi yang legal, dan R adalah jumlah target ronde.

Dengan diterapkannya pemangkasan Beam Search, algoritma mengeksekusi batasan ruang keadaan yang ketat pada setiap transisi tingkat level. Jika didefinisikan B sebagai parameter Beam Width, maka pada setiap iterasi ronde, sistem mengevaluasi maksimal $B * A$ kandidat *state* baru. Proses evaluasi heuristik dan penyortiran deret ini membutuhkan kompleksitas waktu sebesar:

$$O(R \cdot (B \cdot A) \log(B \cdot A))$$

Dalam simulasi ini, nilai B ditetapkan konstan pada angka 2000, A memiliki batas konstan maksimum sekitar 6 tindakan legal per Farm, dan $R = 100$. Karena variabel B dan A diperlakukan sebagai nilai skalar konstan pembatas, maka secara asimptotik kompleksitas waktu program ini tereduksi dan mendekati linier terhadap jumlah target ronde, yaitu $O(R)$.

Sementara itu, penggunaan memori program dieksekusi secara *in-place* saat peralihan ronde. Memori tidak menyimpan seluruh pohon keputusan sejak awal, melainkan hanya menyimpan susunan dictionary level saat ini dan deret antrean

kandidat level berikutnya. Oleh karena itu, kompleksitas ruang murni bergantung pada lebar pemangkasan, yaitu sebesar $O(B)$. Dengan cara ini, penggunaan memori tetap terkontrol sehingga program tidak mengalami kehabisan memori.

IV. HASIL DAN PEMBAHASAN

Pada bab ini, dilakukan pengujian terhadap program simulasi optimasi farming menggunakan data harga riil yang diserap dari berkas konfigurasi. Analisis difokuskan pada pembedahan skenario tingkat kesulitan tertinggi, yaitu mode Impoppable dengan target batas hingga ronde 100, untuk melihat sejauh mana strategi *hybrid* Program Dinamis dan Beam Search dapat mempertahankan efisiensi pencarian dan menghasilkan profitabilitas absolut untuk pemain.

A. Lingkungan Pengujian dan Parameter Skenario

Pengujian dilakukan pada laptop tanpa menggunakan akselerasi GPU. Pengujian murni mengandalkan efisiensi algoritma dalam memproses ruang keadaan di dalam RAM. Parameter dasar yang dimasukkan ke dalam sistem pengujian disesuaikan dengan aturan ketat permainan tanpa intervensi eksternal, dengan rincian sebagai berikut:

- Modal Awal (Initial Cash): \$650 (Ronde 1)
- Batas Maksimal Struktur (Max Farms): 3 Unit
- Lebar Lintasan Pencarian (Beam Width): 2000 State per ronde
- Mode Kesulitan Pengujian: Impoppable (Target akhir: Ronde 100)

Melalui pembacaan data eksternal, harga dasar pendirian Banana Farm pada mode Impoppable terkunci secara akurat pada angka \$1.500, dengan indeks kenaikan harga pembaruan (*upgrade*) sekitar 1,2 kali lipat dari harga dasar mode Medium.

B. Analisis Log Jalur Keputusan Optimal

Berdasarkan log akhir yang dikeluarkan oleh sistem pembaca jalur terbaik, algoritma tidak selalu mengikuti pola greedy dan mampu mengambil keputusan yang lebih menguntungkan dalam jangka panjang. Pada fase awal permainan, algoritma melakukan konvergensi keputusan ke arah pembaruan Jalur Tengah, secara spesifik mengarah pada pembentukan *Monkey Bank* (2-3-0).

Memasuki fase pertengahan (*mid-game*), log program menunjukkan manuver taktis yang sangat krusial. Pada Ronde 60, algoritma melakukan pencairan (*Collect*) saat kapasitas bank mencapai titik jenuh optimal (\$7.000). Lima ronde berselang, di Ronde 65, algoritma mengeksekusi tindakan likuidasi aset (*Sell*) pada bank tersebut untuk meraup suntikan dana segar senilai \$7.705. Dana instan hasil likuidasi ini tidak ditabung, melainkan langsung digunakan sebagai *snowballing* modal untuk melompat ke infrastruktur *Central Market* (0-2-4).

Pada fase akhir permainan (*late-game*), pembukaan akses algoritma hingga ke tingkat 5 terbukti mendominasi performa. Dengan dukungan modifikasi *Late-Game Action Pruning* yang melonggarkan blokir investasi di penghujung permainan, algoritma berhasil mengeksekusi pembelian aset *Tier 5*, yaitu

Monkey Wall Street (0-2-5) tepat pada Ronde 94. Keputusan investasi besar di sisa 6 ronde terakhir ini sangat akurat, mengingat injeksi kas absolut sebesar \$4.000 per putaran dari *Monkey Wall Street* menjamin *Return on Investment* (ROI) yang sangat cepat sebelum simulasi berakhir.

C. Evaluasi Akuntansi Ekonomi dan Validitas Heuristik

Setelah simulasi menyelesaikan perhitungan total hingga akhir Ronde 100, program mengeluarkan laporan ekonomi yang presisi. Hasil numerik final dari skenario pengujian terbaik mutlak dicantumkan pada matriks evaluasi berikut:

- Sisa Uang Tunai Cair di Ronde 100: \$58.225
- Hasil Likuidasi Jual Semua Farm: \$126.816
- Biaya yang Dikeluarkan (Investasi): \$167.040
- Biaya Kotor yang Didapatkan (*Gross Revenue*): \$351.431
- Keuntungan Total Akhir (*Net Profit*): \$184.391

Hasil ini menunjukkan bahwa perluasan ruang keadaan memberikan hasil yang lebih baik. Total biaya yang diinvestasikan sebesar \$167.040 mampu memicu lonjakan perputaran arus kas kotor (*Gross Revenue*) hingga menyentuh \$351.431. Hal ini menaikkan Keuntungan Bersih menjadi \$184.391.

Tingginya Hasil Likuidasi yang mencapai \$126.816 pada akhir putaran menegaskan bahwa heuristik Kekayaan Bersih (*Net Worth*) berhasil menjalankan fungsinya. Algoritma tidak berfokus pada seberapa banyak uang tunai statis di tangan, melainkan mempertahankan bangunan-bangunan bernilai tinggi yang siap dilikuidasi di detik terakhir permainan untuk memaksimalkan akumulasi nilai ekonomi.

D. Efisiensi Komputasi dan Mitigasi Combinatorial Explosion

Pengamatan terhadap performa eksekusi menunjukkan bahwa integrasi memoisasi Program Dinamis tingkat ronde yang dikombinasikan dengan pemangkasan *Beam Search* terbukti sangat efisien. Keberhasilan ini juga ditopang secara signifikan oleh penerapan struktur memori *Tuple Bersarang* (*Functional Linked List*). Penghapusan mekanisme penyalinan riwayat dalam bentuk *array* atau *list* membebaskan algoritma dari beban overhead penyalinan data ($O(N)$), mengubah beban komputasi penelusuran sejarah keputusan menjadi sepenuhnya berbasis *pointer*.

Dengan mengunci parameter *Beam Width* pada nilai 2000 *state*, algoritma dapat menyaring ratusan ribu cabang keputusan yang tidak efisien akibat variasi perhitungan desimal bunga majemuk dan kombinasi *cross-pathing*. Terbukti secara empiris melalui penggunaan instrumen pengukuran waktu bawaan (*built-in timer*), program mampu memproses seluruh kompleksitas *state space* untuk batas 100 ronde (*Impoppable mode*) murni hanya dalam waktu eksekusi aktual 4,492 detik pada perangkat keras standar tanpa bantuan akselerasi GPU.

V. KESIMPULAN DAN SARAN

A. Kesimpulan

Berdasarkan perancangan, implementasi, dan pengujian program yang telah dilakukan, dapat ditarik beberapa kesimpulan berikut:

1. Pemodelan *State-Space* dengan campuran antara Program Dinamis dan pemangkasan *Beam Search* berhasil memitigasi *combinatorial explosion* akibat variasi bunga majemuk. Dengan membatasi 2000 *state* terbaik per ronde, program mampu mensimulasikan hingga 100 ronde dalam 4.5 detik.
2. Analisis build order membuktikan bahwa algoritma ini mampu mengambil keputusan jangka panjang yang melampaui pendekatan heuristik greedy. Algoritma terbukti optimal dalam menahan pengeluaran di ronde awal untuk memaksimalkan efek bunga majemuk Monkey Bank pada fase menengah.
3. Pada pengujian mode *Impoppable*, program berhasil menghasilkan keuntungan kotor (*Gross Revenue*) sebesar \$351.431 hanya dari total investasi \$167.040. Hasil ini menunjukkan bahwa algoritma mampu menangani perhitungan aset dinamis dengan baik.

B. Saran

Terdapat beberapa ruang pengembangan yang dapat dieksplorasi untuk penelitian atau penyempurnaan algoritma selanjutnya:

1. Transisi ke Waktu Kontinu (*Intra-Round Tick*): Akurasi model dapat ditingkatkan dengan memecah durasi ronde menjadi interval waktu detik (*ticks*) untuk mengakomodasi manuver upgrade di pertengahan ronde.
2. Integrasi Variabel Eksternal: Penelitian berikutnya dapat menyertakan variabel Hero Benjamin dan Monkey Village untuk menguji algoritma terhadap perluasan cabang keputusan yang lebih masif.
3. Pengembangan Antarmuka Visual (GUI): Program berbasis teks (CLI) saat ini dapat dikonversi menjadi aplikasi web atau desktop agar lebih interaktif bagi pengguna kasual.

LAMPIRAN

Berikut merupakan tautan terkait github dan juga video demonstrasi dalam makalah ini yang telah dipublish melalui youtube:

Tautan codebase github: <https://github.com/DeSince29/btd6-farm-optimizer/releases/tag/v1.0>

Tautan video demonstrasi: <https://youtu.be/iv4e2M6oapA>

UCAPAN TERIMA KASIH

Pertama-tama penulis ingin mengucapkan terima kasih kepada Tuhan Yang Maha Esa atas anugerah-Nya sehingga penulis dapat menyelesaikan makalah ini sedemikian rupanya.

Penulis juga ingin mengucapkan terima kasih kepada dosen mata kuliah Strategi Algoritma IF2211, yaitu Pak Rinaldi Munir, Pak Rila Mandala, Pak Catur, Bu Fazat, dan Bu Tricya. Tidak lupa penulis mengucapkan terima kasih kepada para asisten mata kuliah Strategi Algoritma IF2211 yang mendukung kelancaran dalam proses pembelajaran mata kuliah ini. Penulis juga ingin mengucapkan terima kasih kepada seluruh pembaca makalah ini, dan semoga makalah ini dapat memberikan manfaat sebagaimana yang penulis harapkan.

REFERENSI

- [1] Ninja Kiwi, *Bloons TD 6*, Version 42.0, Auckland, New Zealand, 2018.
- [2] "Banana Farm (BTD6)," Bloons Wiki Fandom. [Online]. Available: [https://bloons.fandom.com/wiki/Banana_Farm_\(BTD6\)](https://bloons.fandom.com/wiki/Banana_Farm_(BTD6)). [Accessed: Jun. 11, 2026].
- [3] R. Munir, "Strategi Algoritma 2025-2026," STEI ITB. [Online]. Available: <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/stmik.htm>. [Accessed: Jun. 11, 2026].

- [4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 4th ed. Cambridge, MA, USA: MIT Press, 2022.
- [5] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed. Hoboken, NJ, USA: Pearson, 2020.

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, 19 Juni 2025



Moreno Syawali Ganda Sugita
13524096