Implementasi Algoritma Greedy dan Template Matching untuk Otomatisasi Pencapaian Skor Maksimal pada Permainan Candy Crush

Dika Pramudya Nugraha - 13524132
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: pramudyan196@gmail.com , 13524132@std.stei.itb.ac.id

***Abstract*—Permainan *puzzle match 3* seperti *Candy Crush* menuntut pemain untuk memilih langkah penukaran yang mampu menghasilkan kombinasi objek terpanjang guna mencapai skor maksimal. Makalah ini membahas implementasi bot otomatis berbasis bahasa pemrograman Python yang mampu membaca tangkapan layar papan permainan, mengonversinya menjadi representasi matriks karakter, dan menentukan langkah terbaik menggunakan algoritma Greedy. Modul *computer vision* yang dikembangkan memanfaatkan deteksi *grid* otomatis dan klasifikasi warna berdasarkan ruang warna HSV untuk memetakan setiap sel menjadi simbol permen spesifik (seperti B, H, O, U, dan sebagainya) maupun rintangan (X). Selanjutnya, modul logika permainan menyimulasikan proses pencocokan, efek gravitasi, dan reaksi berantai (*cascade*). Hasil eksperimen membuktikan keandalan sistem dengan keberhasilan melewati dua puluh empat skenario pengujian unit, termasuk verifikasi pembacaan matriks berdimensi dinamis pada tangkapan layar uji dengan tingkat akurasi pengenalan sel yang sempurna.**

Keywords—algoritma greedy; template matching; computer vision; match 3; Candy Crush; OpenCV; otomatisasi permainan

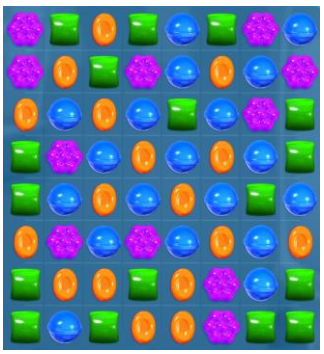
I. PENDAHULUAN

Permainan teka-teki bergenre match 3, seperti *Candy Crush*, merupakan salah satu bentuk hiburan interaktif yang menuntut pengenalan pola visual dan pengambilan keputusan taktis. Mekanik dasar permainan ini mengharuskan pemain untuk menukar dua elemen yang bersebelahan guna membentuk barisan linier yang terdiri dari minimal tiga elemen dengan warna atau tipe yang sama. Setiap langkah yang divalidasi oleh sistem akan memicu serangkaian aksi, seperti elemen yang cocok akan dihancurkan, elemen di atasnya akan jatuh mengisi kekosongan akibat efek gravitasi, dan kombinasi baru dapat terbentuk secara otomatis yang memicu reaksi berantai (*cascade*) [1].

Bagi manusia, memprediksi hasil akhir dari sebuah langkah yang memicu reaksi berantai panjang sering kali di luar batas kemampuan kalkulasi mental, sehingga pemain rentan melewatkan langkah yang paling menguntungkan. Menentukan langkah dengan potensi akumulasi skor tertinggi pada setiap giliran pada dasarnya merupakan masalah optimasi kombinatorial yang sangat kompleks [1]. Menyelesaikan masalah ini secara eksak menggunakan metode pencarian komprehensif (*Brute Force*) pada papan permainan yang besar akan memakan waktu komputasi yang sangat lambat, sehingga tidak relevan untuk diterapkan pada skenario permainan waktu nyata [2].

Sebagai solusi atas permasalahan tersebut, makalah ini mengimplementasikan sebuah agen cerdas atau *decision support system* berbasis algoritma Greedy. Sistem ini dirancang untuk membaca tangkapan layar papan permainan, mendeteksi ukuran dan bentuk *grid* secara dinamis, serta mengklasifikasikan isi dari setiap sel secara otomatis menggunakan teknik *computer vision* [3]. Data visual tersebut kemudian dikonversi menjadi representasi matriks untuk dievaluasi. Algoritma Greedy akan menyimulasikan setiap kemungkinan langkah penukaran yang valid dan memilih satu langkah dengan perolehan skor simulasi tertinggi saat itu juga (*local optimum*) [2]. Pendekatan ini dipilih karena efisien secara komputasional, mudah diimplementasikan, dan terbukti sangat efektif untuk menghasilkan langkah berkualitas tinggi secara instan pada otomatisasi video game.

Secara spesifik, penulisan makalah ini bertujuan untuk merancang arsitektur agen otomatis yang mengintegrasikan ekstraksi data visual menjadi representasi matriks, menyimulasikan logika internal permainan, dan menerapkan algoritma Greedy untuk merekomendasikan langkah paling optimal secara lokal. Keandalan dan akurasi sistem ini kemudian divalidasi lebih lanjut melalui serangkaian pengujian unit terhadap kinerja visi komputer maupun efisiensi pencarian algoritmanya.



Gambar 1. Contoh screenshot papan permainan Candy Crush yang dianalisis oleh sistem

II. LANDASAN TEORI

A. Mekanika Permainan Match 3

Permainan teka-teki match-3 dimainkan di atas sebuah *grid* berdimensi $N \times M$ yang setiap sel koordinatnya direpresentasikan oleh elemen dengan atribut warna atau jenis rintangan spesifik. Aturan dasar permainan ini mengizinkan perpindahan atau penukaran posisi antara dua elemen yang saling bersebelahan secara ortogonal (vertikal maupun horizontal). Sebuah langkah penukaran dinyatakan valid secara logis apabila manuver tersebut memicu pembentukan pola linear yang memuat sekurang-kurangnya tiga elemen beratribut warna identik.

Penyelesaian pola ini memicu sistem evaluasi berbasis bobot, yaitu 3 elemen bernilai 30 poin, 4 elemen bernilai 60 poin, dan 5 elemen bernilai 100 poin, dengan penambahan konstan 20 poin untuk setiap entitas tambahan yang terhubung. Proses penghancuran elemen ini pada praktiknya akan memicu jatuhnya entitas acak baru dari sumbu tertinggi akibat efek gravitasi. Hal ini membuka peluang terjadinya reaksi berantai yang tidak dapat diprediksi seutuhnya, menjadikan permainan ini sebuah permasalahan optimasi kombinatorial yang memiliki ruang pencarian sangat kompleks [1].

B. Paradigma Algoritma Greedy

Algoritma Greedy merupakan paradigma pemecahan masalah yang berfokus pada pengambilan keputusan heuristik yang paling menguntungkan pada saat itu juga (local optimum) [2]. Algoritma ini beroperasi tanpa mengevaluasi atau memperhitungkan konsekuensi dan struktur percabangan di masa depan.

$$m_{opt} = \arg\max_{m \in C} \text{Score}(m, B) \tag{1}$$

Dalam persamaan tersebut, variabel m_{opt} merepresentasikan langkah penukaran paling optimal secara lokal yang akan direkomendasikan oleh sistem. Sementara itu, fungsi Score(*m*, *B*) bertindak sebagai fungsi heuristik yang mengkalkulasi total perolehan poin apabila langkah *m* disimulasikan pada matriks papan *B*, di mana kalkulasi ini sudah mencakup akumulasi poin dari efek gravitasi maupun reaksi berantai (*cascade*). Lebih lanjut, pembentukan himpunan kandidat langkah *C* dibatasi oleh *feasibility constraints*. Sebuah langkah *m* yang melibatkan penukaran koordinat (r1, c1) dan (r2, c2) dinyatakan sah dan valid (m ∈ C) hanya jika memenuhi tiga batasan matematis secara simultan. Pertama, jarak *Manhattan* antara kedua sel harus sama dengan satu (|r1 - r2| + |c1 - c2| = 1), yang memastikan kedua sel saling bersebelahan secara ortogonal. Kedua, kedua koordinat yang ditukar tidak boleh berupa sel rintangan statis, area kosong, maupun entitas tak diketahui ($B(r_1, c_1) \notin \{X, ?\} \wedge B(r_2, c_2) \notin \{X, ?\}$). Ketiga, simulasi penukaran wajib menghasilkan minimal satu kombinasi kecocokan atau bernilai mutlak lebih dari nol Score(*m*, *B*) > 0).

Algoritma 1 Pencarian Langkah Optimal (*Greedy Match 3*)

Membutuhkan: *Board* (Matriks berukuran N x M)
Pastikan: *best_move* (Koordinat langkah penukaran dengan skor tertinggi)

```
1: max_score ← 0
2: best_move ← NULL
3: for r ← 0 to N − 1 do
4:   for c ← 0 to M − 1 do
5:     if Board[r][c] ∉ Rintangan then
6:       ▷ Evaluasi penukaran dengan sel Kanan
7:       if (c + 1 < M) and (Board[r][c + 1] ∉ Rintangan) then
8:         score ← SimulateSwap(Board, r, c, r, c + 1)
9:         if score > max_score then
10:          max_score ← score
11:          best_move ← (r, c, r, c + 1)
12:        end if
13:      end if
14:
15:      ▷ Evaluasi penukaran dengan sel Bawah
16:      if (r + 1 < N) and (Board[r + 1][c] ∉ Rintangan) then
17:        score ← SimulateSwap(Board, r, c, r + 1, c)
18:        if score > max_score then
19:          max_score ← score
20:          best_move ← (r, c, r + 1, c)
21:        end if
22:      end if
23:    end if
24:  end for
25: end for
26: return best_move
```

Dalam konteks otomatisasi permainan ini, algoritma bekerja dengan cara menyimulasikan seluruh permutasi penukaran antar sel yang bersebelahan secara komprehensif pada keadaan matriks saat ini. Penukaran yang menghasilkan kalkulasi akumulasi poin tertinggi termasuk estimasi poin dari reaksi berantai akan diekstraksi sebagai langkah rekomendasi mutlak. Pendekatan ini dieksekusi dengan kompleksitas asimtotik waktu $O(N \times M)$ pada setiap gilirannya. Tingkat kompleksitas linier ini menjadikan Greedy sangat efisien dan jauh lebih ringan secara beban memori komputasi apabila dikomparasikan dengan metode pencarian graf ruang keadaan yang mendalam [2].

C. Computer Vision dan Klasifikasi Ruang Warna

Template matching di dalam ranah computer vision adalah sebuah teknik analitik yang bertujuan untuk mengidentifikasi bagian sub citra yang memiliki tingkat korelasi atau kemiripan matematis tertinggi dengan citra referensi asal [3]. Untuk mengatasi kelemahan akurasi akibat variasi intensitas pencahayaan dan pantulan animasi pada video game, implementasi sistem ini mengadaptasi pendekatan pencocokan berbasis warna (color-based template matching) menggunakan model ruang warna HSV (Hue, Saturation, Value).

Representasi HSV dipilih karena keunggulannya dalam memisahkan informasi warna murni atau pigmen (Hue) dari intensitas pencahayaan (Value) [3]. Program dirancang untuk mengekstraksi piksel dengan tingkat saturasi tinggi pada area pusat koordinat sel, lalu memetakan spektrum Hue dominan tersebut ke dalam klasifikasi warna permen target. Sementara itu, untuk sel yang berada di luar batas operasional permainan

(seperti area kosong berwarna toska) dan rintangan statis, deteksi dilakukan melalui pembatasan parameter kombinasi HSV.

D. Pemodelan Representasi Matriks

Struktur grafis papan permainan dikonversi ke dalam bentuk representasi matriks dua dimensi, supaya data visual diskrit dapat diproses oleh operasi matematis logis. Setiap elemen leksikal dalam matriks ini menyimpan karakter tunggal (tipe data *string* atau *char*) yang mendeskripsikan wujud fisik objek pada koordinat kisi tersebut. Pemetaan variabel karakter yang ditetapkan meliputi:

- B = Elemen dominan biru
- H = Elemen dominan hijau
- O = Elemen dominan Oranye
- U = Elemen dominan Ungu
- X = Penanda untuk sel di luar zona permainan atau rintangan permanen yang tidak memiliki properti penukaran.
- ? = Penanda ketidakpastian (*wildcard*). Digunakan untuk merepresentasikan permen misterius hasil efek gravitasi yang warnanya belum dapat diklasifikasikan oleh sistem sensor visual sebelum objek tersebut berhenti bergeser.

III. PERANCANGAN SISTEM

A. Arsitektur Perangkat Lunak Modular

Pengembangan agen otomatis untuk permainan *match 3* ini dirancang menggunakan pendekatan arsitektur modular. Pemisahan komponen fungsional ini bertujuan untuk mempermudah proses pemeliharaan kode, *unit testing*, serta melokalisasi *bug* tanpa mengganggu integritas proses algoritma utama [4]. Secara hierarkis, sistem ini didekomposisi menjadi empat modul utama yang saling berinteraksi secara sekuensial, yaitu Modul Antarmuka Grafis (GUI), Modul Visi Komputer, Modul Logika Permainan, dan Modul Algoritma *Greedy*. Penjabaran fungsi spesifik dari masing-masing modul dirangkum pada Tabel I berikut:

Tabel I. Modul utama sistem bot Match 3

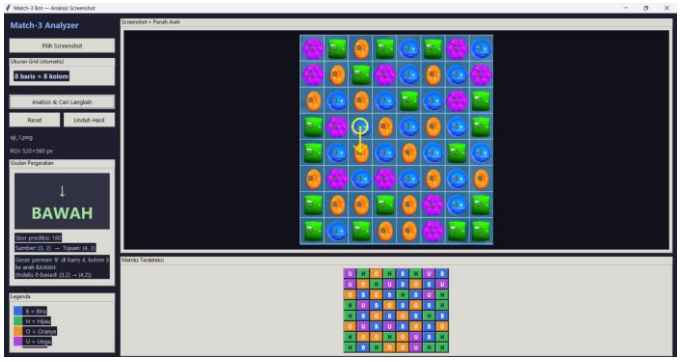
Nama Modul	Tanggung Jawab
gui_app.py	Bertindak sebagai lapisan presentasi (*front-end*). Menyediakan antarmuka bagi pengguna untuk mengunggah tangkapan layar, serta memvisualisasikan *overlay grid* dan arah rekomendasi langkah optimal.
vision.py	Bertanggung jawab atas ekstraksi data visual. Melakukan deteksi batas wilayah *Region of Interest* (ROI), kalkulasi ukuran matriks kisi secara dinamis, dan klasifikasi nilai HSV pada tiap sel untuk menghasilkan matriks representasi karakter.
game_logic.py	Berfungsi sebagai engine fisika internal. Menyediakan fungsi fundamental seperti *find_matches* (mendeteksi pola linear), *apply_gravity* (menyimulasikan elemen jatuh), dan *simulate_swap* (menjalankan penukaran elemen logis).
greedy.py	Merupakan inti pemrosesan pengambilan keputusan. Menggunakan fungsi *analyze_board* dan *find_best_move* untuk mengevaluasi matriks dan mengeksekusi heuristik pencarian langkah dengan potensi kombo skor tertinggi.

B. Alur Kerja Sistem

Eksekusi program diinisiasi ketika pengguna memuat gambar tangkapan layar papan permainan ke dalam sistem melalui antarmuka *gui_app.py*. Setelah citra digital diterima, kontrol program diserahkan kepada modul *vision.py* untuk menjalankan fungsi *detect_board()*. Fungsi ini secara otomatis mengisolasi Region of Interest (ROI) dari elemen antarmuka lain yang tidak relevan, sekaligus mendeteksi resolusi dan dimensi grid permainan secara dinamis.

Selanjutnya, fungsi *image_to_board()* pada modul yang sama memproses piksel pada tiap sel koordinat menggunakan klasifikasi ruang warna HSV, kemudian mentranslasikannya menjadi matriks karakter dua dimensi. Matriks diskrit ini kemudian dioper ke dalam modul *greedy.py*. Dengan memanfaatkan heuristik yang selalu mencari kandidat solusi optimum lokal di setiap tahapan komputasi [5], fungsi *analyze_board()* akan menyimulasikan setiap probabilitas penukaran melalui aturan fisika dari modul *game_logic.py*.

Proses ini bermuara pada penemuan satu titik koordinat penukaran dan arah pergerakan yang menghasilkan akumulasi skor tertinggi pada state saat itu. Sebagai hasil akhir, sistem mengembalikan data tersebut ke *gui_app.py* untuk dirender secara visual kepada pengguna dalam bentuk matriks yang terdeteksi beserta indikator panah arah langkah yang direkomendasikan.



Gambar 2. Antarmuka grafis bot menampilkan screenshot dengan *overlay grid*, panah arah *swap*, dan matriks hasil deteksi.

IV. Implementasi

A. Modul Game Logic (game_logic.py)

Modul ini menjadi mesin simulasi inti. Fungsi *find_matches()* memindai baris dan kolom untuk menemukan run permen sama (panjang ≥ 3), mengabaikan sel 'X' dan '?'. Fungsi *apply_gravity()* menghapus sel yang match, menjatuhkan permen ke bawah, dan mengisi bagian atas dengan '?'. Sel 'X' bertindak sebagai penghalang gravitasi di kolomnya. Fungsi *simulate_swap()* menukar dua sel bersebelahan lalu mengulang match-gravitasi hingga tidak ada *cascade* lagi.

```
# CUPLIKAN KODE 1: Logika Pencarian dan Simulasi Penukaran
# Sumber: game_logic.py, baris 36-43 dan 152-190


def find_matches(board: Board) -> Tuple[List[Coord], int]:
    """
    Cari semua match horizontal/vertikal (panjang
    >= 3). Mengabaikan sel 'X' dan '?'.
    """
    ...


def simulate_swap(board: Board, r1: int, c1: int, r2: int, c2: int) -> int:
    sim_board = copy.deepcopy(board)
    sim_board[r1][c1], sim_board[r2][c2] = sim_board[r2][c2], sim_board[r1][c1]
    total_score = 0
    while True:
        matches, score = find_matches(sim_board)
        if not matches:
            break
        total_score += score
        sim_board = apply_gravity(sim_board, matches)
    return total_score
```

Penerapan *copy.deepcopy(board)* pada fungsi *simulate_swap()* sangat krusial dalam paradigma algoritma Greedy. Manipulasi data ini memastikan bahwa simulasi dilakukan pada memori tiruan, sehingga status matriks papan asli tidak terkorupsi selama proses komputasi skor berantai berlangsung.

```
# CUPLIKAN KODE 2: Kalkulasi Skor Berdasarkan Panjang Kombinasi
# Sumber: game_logic.py, baris 15-16 dan 22-28


    MATCH_SCORES = {3: 30, 4: 60, 5: 100}


    def _score_for_run(length: int) -> int:
        if length < 3:
            return 0
        if length in MATCH_SCORES:
            return MATCH_SCORES[length]
        return MATCH_SCORES[5] + (length - 5) * 20
```

B. Modul Algoritma Greedy (greedy.py)

Algoritma Greedy beroperasi dengan melakukan iterasi secara sistematis terhadap seluruh koordinat sel di dalam papan permainan. Untuk setiap sel yang tervalidasi bukan sebagai rintangan, program akan menyimulasikan penukaran posisi hanya ke arah kanan dan bawah. Secara matematis, pengecekan dua arah ini sudah cukup untuk mengevaluasi seluruh permutasi pasangan unik yang bersebelahan tanpa redundansi komputasi. Skor potensial dari setiap pasangan dihitung menggunakan simulate_swap(), dan pasangan dengan akumulasi skor tertinggi akan diekstraksi sebagai langkah absolut.

```
# CUPLIKAN KODE 3: Inti Pencarian Langkah Algoritma Greedy
# Sumber: greedy.py, baris 75-114


def find_best_move(board: Board) -> Optional[Move]:
    best_score = 0
    best_move = None
    directions = ((0, 1), (1, 0))


    for r in range(rows):
        for c in range(cols):
            if board[r][c] in BLOCKED_CHARS:
                continue
            for dr, dc in directions:
                r2, c2 = r + dr, c + dc
                score = simulate_swap(board, r, c, r2, c2)
                if score > best_score:
```

```
                    best_score = score
                    best_move = (r, c, r2, c2)
        return best_move
```

Hasil komputasi koordinat ini kemudian dibungkus oleh fungsi *analyze_board()* ke dalam struktur data kamus, yang menyertakan informasi mengenai titik asal, titik tujuan, estimasi perolehan skor, serta representasi tekstual dari arah pergerakan.

C. *Modul Computer Vision (vision.py)*

Modul ini merepresentasikan saluran utama (*pipeline*) pemrosesan citra digital yang mengonversi tangkapan layar mentah menjadi matriks diskrit. Eksekusi program terbagi menjadi lima tahapan utama:

1. *_content_bounds()*: Mengisolasi *Region of Interest* (ROI) papan permainan berdasarkan kerapatan piksel bersaturasi tinggi.
2. *detect_grid_size()*: Menganalisis dimensi *grid* secara dinamis dengan menguji permutasi ukuran (misalnya 5 hingga 10) dan memilih probabilitas dengan tingkat konsistensi warna antarsel tertinggi.
3. *_align_roi_to_grid()*: Melakukan kalibrasi agar koordinat ROI habis dibagi dengan dimensi *grid* yang ditemukan.
4. *_get_cell_patch()*: Melakukan pemotongan (*cropping*) sebesar 85% pada area tengah sel untuk menghindari distorsi pembacaan akibat garis tepi.
5. *_classify_cell_raw()*: Mengekstraksi karakteristik warna berdasarkan ruang warna HSV.

Kalibrasi spektrum *Hue* yang digunakan untuk pengenalan objek dirangkum pada Tabel II.

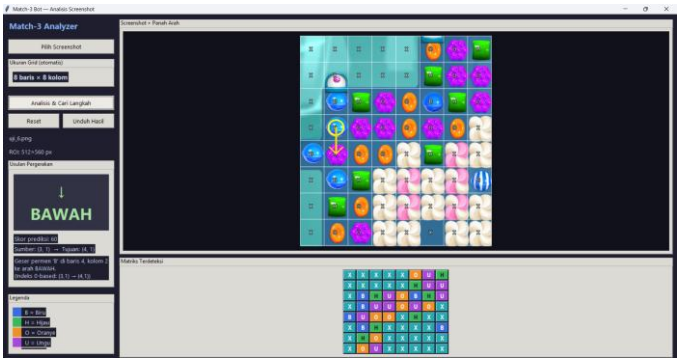
Tabel II. Pemetaan Spektrum *Hue* HSV terhadap Simbol Karakter Representasi

Simbol	Rentang *Hue*	Deskripsi Objek Visual
O	48-67	Objek permen berwarna oranye
H	67-86	Objek permen berwarna hijau
X	86-101	Elemen latar belakang hampa
B	95-118	Objek permen berwarna biru
U	120-148	Objek permen berwarna ungu
X	Khusus	Rintangan statis bersaturasi rendah

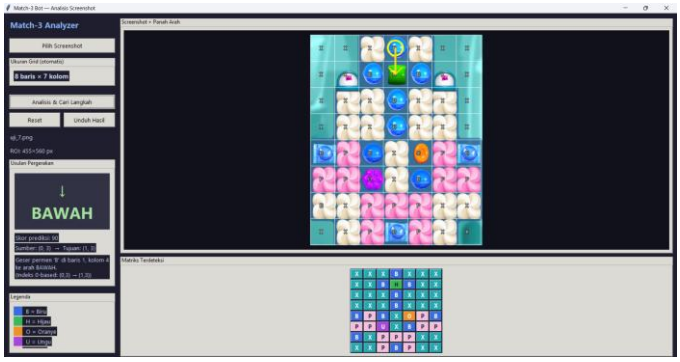
```
# CUPLIKAN KODE 4: Deklarasi Rentang Spektrum Warna
# Sumber: vision.py, baris 32-38 dan 261-273


HUE_ORANGE = (48, 67)
HUE_GREEN = (67, 86)
HUE_WATER = (86, 101)
HUE_BLUE = (95, 118)
HUE_PURPLE = (120, 148)
```

```
def _hue_to_candy(mean_hue: float, mean_sat: float)
-> Optional[str]:
    if HUE_ORANGE[0] <= mean_hue < HUE_ORANGE[1]:
        return "O"
    if HUE_GREEN[0] <= mean_hue < HUE_GREEN[1]:
        return "H"
    if HUE_BLUE[0] <= mean_hue <= HUE_BLUE[1]:
        return "B"
    if HUE_PURPLE[0] <= mean_hue <= HUE_PURPLE[1]:
        return "U"
    return None
```

```
# CUPLIKAN KODE 5: Pemrosesan Karakteristik Piksel
Tiap Sel
# Sumber: vision.py, baris 276-300


def _classify_cell_raw(patch: np.ndarray) -> str:
    metrics = _compute_cell_metrics(patch)
    mean_hue, mean_sat, low_frac, mean_sat_all,
edge = _dominant_hue_stats(patch)


    if _is_swirl_blocker(mean_hue, mean_sat,
low_frac, mean_sat_all, edge):
        return "X"
    if _is_water_background(metrics, mean_hue,
edge):
        return "X"


    candy = _hue_to_candy(mean_hue, mean_sat)
    if candy:
        return candy
    return "X"
```



Gambar 3. Hasil ekstraksi visi komputer yang mengonversi informasi piksel tangkapan layar menjadi representasi matriks logis 8 × 8. Karakter 'X' menandakan identifikasi absolut terhadap area hampa atau rintangan.



Gambar 4. Demonstrasi keandalan fitur deteksi *grid* otomatis pada papan yang tidak merepresentasikan struktur persegi sempurna.

D. Modul Antarmuka Pengguna (gui_app.py)

Antarmuka Grafis Pengguna (GUI) dibangun memanfaatkan pustaka standar Tkinter yang diintegrasikan dengan Python Imaging Library (PIL). Modul ini menjembatani interaksi manusia dengan komputer. Pengguna dapat memuat instrumen uji berupa tangkapan layar secara langsung. Sistem secara *real time* akan memproyeksikan *overlay* pembatas *grid*, melabeli karakter hasil pemindaian pada tiap sel, menampilkan matriks akhir pada panel terpisah, serta menggambar indikator vektor berupa panah kuning untuk mengarahkan pengguna pada langkah penukaran yang paling optimal.

```
# CUPLIKAN KODE 6: Orkestrasi Analisis Integrasi Modul pada GUI
# Sumber: gui_app.py, baris 20-28 dan sekitar 260-280


    from greedy import analyze_board
    from vision import detect_board, image_to_board, draw_grid_overlay, draw_move_arrow


    self.vision_config = detect_board(self.image_bgr)
    self.board = image_to_board(self.image_bgr, self.vision_config)
    self.analysis = analyze_board(self.board)
```

Gambar 5. Visualisasi antarmuka utama *Match-3 Analyzer* dalam mode *standby*

Gambar 6. Keadaan antarmuka pasca-analisis. Vektor panah kuning merepresentasikan rekomendasi heuristik dari algoritma *Greedy*, menghubungkan titik koordinat asal penukaran menuju titik koordinat tujuan yang divalidasi.

V. PENGUJIAN DAN HASIL

Fase verifikasi dan validasi fungsionalitas sistem dieksekusi menggunakan kerangka kerja pengujian bawaan Python, yakni unittest. Metodologi pengujian perangkat lunak ini diaplikasikan untuk memastikan bahwa setiap komponen terisolasi beroperasi sesuai dengan spesifikasi rancangan [4]. Rangkaian pengujian ini mencakup dua puluh empat kasus uji (*test cases*) yang berfokus pada evaluasi modul logika permainan, akurasi algoritma *Greedy*, dan ketepatan pemrosesan modul visi komputer.

A. Pengujian Modul Logika Permainan (test_logic.py)

Evaluasi pada modul logika ditujukan untuk memvalidasi kepatuhan simulasi program terhadap aturan internal permainan. Pengujian ini merangkum empat parameter fungsional utama:

1. Validasi *find_matches*

   Menguji keberhasilan sistem dalam mendeteksi pola kombinasi berukuran tiga atau empat elemen secara horizontal dan vertikal, mendeteksi pola persilangan kompleks, serta memastikan bahwa sel rintangan (X) diabaikan secara mutlak dari kalkulasi.

2. Validasi *apply_gravity*

   Memverifikasi penurunan elemen secara presisi setelah proses penghancuran, menguji perilaku rintangan statis sebagai pemblokir laju gravitasi di suatu kolom, serta memastikan pengisian elemen ruang kosong di baris teratas dengan karakter *wildcard* (?).

3. Validasi *simulate_swap*

   Memastikan bahwa fungsi sanggup menangani simulasi reaksi berantai (*cascade*) yang bersifat multi-langkah. Pengujian ini juga memvalidasi bahwa penukaran yang tidak sah akan mengembalikan nilai

skor nol, dan proses simulasi terbukti tidak memutasi keadaan asli dari matriks papan.

4. Validasi Heuristik *Greedy*

   Menguji kapabilitas algoritma dalam mengekstraksi titik koordinat dengan kalkulasi skor tertinggi, serta kemampuan penanganan eksepsi (*exception handling*) apabila papan berada dalam kondisi ketiadaan langkah yang valid (*deadlock*).

B. Pengujian Modul Computer Vision (test_vision.py)

Efektivitas pemrosesan citra digital diukur melalui seberapa presisi modul mampu merekonstruksi ukuran *grid* dan memetakan warna piksel menjadi karakter matriks. Tabel 3 memaparkan ringkasan hasil pengujian pada berbagai instrumen tangkapan layar.

Tabel III. Ringkasan Hasil Pengujian Deteksi Grid dan Klasifikasi Sel

Instrumen Uji	Resolusi Grid	Parameter Validasi
uji_1.png	8 x 8	Terverifikasi tidak mendeteksi sel hampa (matriks penuh dengan elemen aktif).
uji_4.png	6 x 7	Area komputasi tengah dan rintangan (*swirl*) terdeteksi akurat sebagai 'X'. Elemen sisanya terklasifikasi sebagai karakter permen.
uji_5.png	8 x 8	Deteksi absolut pada genangan air di kuadran kiri-atas sebagai 'X'. Sel pada koordinat (4,0) teridentifikasi sebagai 'B'.
uji_6.png	8 x 8	Matriks terisi penuh dan terverifikasi sempurna (64/64 sel cocok secara absolut dengan *ground truth*).

Untuk memvalidasi keakuratan instrumen uji_6.png, hasil luaran program dikomparasikan secara langsung dengan matriks *ground truth* berikut:

```
X X X X X O U H
X X X X X H U U
X B H U O B H U
X B U U O U O X
B U O O X H X X
X B H X X X X B
X H O X X X X X
X O U X X X X X
```

Instruksi terminal yang dieksekusi untuk menjalankan otomatisasi rangkaian pengujian ini sebagai berikut:

```
python -m unittest discover tests -v.
```

Gambar 7. Tangkapan layar antarmuka terminal pasca-eksekusi *unit test*, menampilkan status keberhasilan seratus persen pada dua puluh empat kasus uji (*Ran 24 tests OK*).

Gambar 8. Komparasi visual antara data tangkapan layar asli uji_6.png dengan hasil pemetaan matriks karakter yang diproyeksikan oleh antarmuka grafis

C. Analisis Kinerja Heuristik Greedy

Berdasarkan instrumen uji uji_6.png, fungsi analisis algoritma *Greedy* mengevaluasi seluruh permutasi penukaran pada *state space* dan merekomendasikan langkah penukaran elemen. Program mengarahkan penukaran antara permen berkarakter 'B' (Biru) pada koordinat asal (4,0) menuju permen berkarakter 'U' (Ungu) pada koordinat tujuan (4,1) dengan arah pergerakan vektor ke kanan. Rekomendasi koordinat ini diekstraksi secara independen oleh sistem karena menyajikan nilai simulasi tertinggi di antara seluruh kandidat penukaran.

Sesuai dengan landasan teoretis fundamental mengenai strategi algoritma heuristik [2], [5], pendekatan Greedy pada dasarnya tidak memberikan jaminan matematis untuk selalu menghasilkan perolehan skor maksimal secara global (*global optimum*) pada akhir permainan. Meskipun rentan terjebak dalam jebakan *local optimum*, arsitektur algoritma ini menyajikan performa komputasi instan yang sangat efisien secara kompleksitas waktu, menjadikannya sebuah model penyelesaian yang sangat ideal dan layak pakai untuk diimplementasikan sebagai asisten otomatis dalam lingkungan video game berkinerja waktu nyata.

VI. KESIMPULAN

Makalah ini telah berhasil memaparkan implementasi agen cerdas (bot) permainan match-3 berskala modular yang mengintegrasikan teknik visi komputer berbasis model ruang warna *HSV*, simulasi logika internal permainan dengan efek reaksi berantai (*cascade*), serta penerapan algoritma *Greedy* untuk merekomendasikan langkah penukaran elemen. Sistem yang dibangun terbukti tangguh dalam mendeteksi dimensi *grid* secara dinamis (seperti 8 × 8 maupun 6 × 7), mengekstraksi objek permen dan rintangan secara akurat dari instrumen tangkapan layar, serta memproyeksikan luaran analisis tersebut melalui antarmuka grafis pengguna yang interaktif.

Walaupun sistem mampu beroperasi dengan efisiensi tinggi, terdapat beberapa batasan struktural pada implementasi saat ini. Pertama, akurasi klasifikasi ruang warna masih memiliki tingkat sensitivitas terhadap variasi intensitas pencahayaan atau perubahan tema visual di dalam permainan. Kedua, pendekatan heuristik *Greedy* secara teoretis dikonstruksi untuk mengejar nilai optimal lokal, sehingga tidak menjamin tercapainya perolehan skor kumulatif yang paling maksimal secara global pada akhir permainan [2]. Ketiga, arsitektur saat ini beroperasi murni sebagai sistem pendukung keputusan (*decision support*), sehingga belum dilengkapi dengan modul pengontrol otomatis yang mampu mengeksekusi instruksi pergerakan kursor secara langsung ke dalam jendela permainan.

Guna mengoptimalkan kinerja sistem pada penelitian mendatang, beberapa pengembangan lebih lanjut sangat disarankan. Pengembangan tersebut mencakup integrasi pustaka eksekusi otomatis (seperti PyAutoGUI) agar sistem dapat sepenuhnya mengambil alih kendali permainan secara waktu nyat. Selain itu, peningkatan modul pengenalan objek disarankan untuk beralih pada teknik *template matching* berbasis kecocokan fitur citra seutuhnya dan bukan sekadar rentang spektrum warna murni program mampu mengidentifikasi rupa permen spesial (seperti striped atau *wrapped candy*). Terakhir, komparasi algoritma heuristik dengan algoritma pencarian yang mengevaluasi struktur pohon keadaan lebih dalam (*lookahead search*), seperti *Minimax* atau *Beam Search*, patut dieksplorasi guna memecahkan masalah pencapaian optimal global.

VIDEO LINK AT YOUTUBE

https://youtu.be/DhWuKm-OVjk?si=Bxn6VtE1j6fHRgQU

REPOSITORY LINK AT GITHUB

https://github.com/Dikanugraha-hub/Makalah-Stima-Match3-Candy-Crush

UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih kepada Dosen Pengampu Mata Kuliah Strategi Algoritma serta pihak yang telah memberikan masukan dalam penyusunan laporan dan pengembangan sistem bot Match 3 ini.

REFERENCES

[1] T. Walsh, "Candy Crush is NP-hard," CoRR, vol. abs/1403.1911, 2014. [Online]. Tersedia: https://arxiv.org/abs/1403.1911

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, dan C. Stein, Introduction to Algorithms, edisi ke-3. Cambridge, MA, USA: MIT Press, 2009.

[3] G. Bradski, "The OpenCV Library," Dr. Dobb's Journal of Software Tools, vol. 25, no. 11, hal. 120-123, 2000.

[4] I. Sommerville, *Software Engineering*, edisi ke-10. Boston, MA, USA: Pearson, 2015.

[5] R. Munir, "Diktat Kuliah IF2211 Strategi Algoritma: Algoritma Greedy," Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung, 2025. [Online]. Tersedia: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/stima25-26.htm

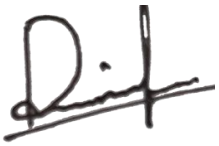
[6] R. C. Gonzalez dan R. E. Woods, *Digital Image Processing*, edisi ke-4. New York, NY, USA: Pearson, 2018.

[7] S. Russell dan P. Norvig, *Artificial Intelligence: A Modern Approach*, edisi ke-4. 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 2026



Dika Pramudya Nugraha
13524132