Bermain Card Thief dengan Algoritma *Greedy*

Anindya Prameswari Ekaputri / 135 18 034
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: pamsrewari@gmail.com

***Abstract*—Card Thief bertujuan membuat strategi keluar dari kastil tanpa ketahuan oleh penjaga dan mengumpulkan *treasure* sebanyak-banyaknya. Melalui makalah ini, algoritma *greedy* digunakan sebagai perkiraan untuk menentukan algoritma yang lebih baik dalam memainkan Card Thief. Hasilnya, algoritma *greedy* memang kurang tepat karena tidak bisa menggambarkan perubahan *state* di setiap pergerakan. Disarankan algoritma program dinamis, namun karena *state* dan *stage* permainan begitu beragam, penyelesaian dengan program dinamis pun berharga cukup mahal.**

Keywords—Card Thief; greedy; ordering

I. PENDAHULUAN

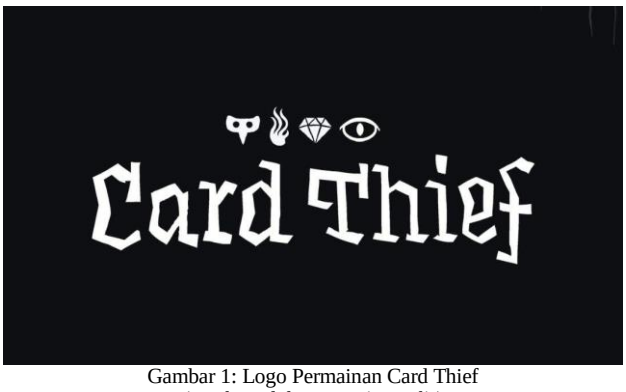
Algoritma adalah sesuatu yang dikaitkan erat dengan informatika dan pemrograman. Namun, menurut Kamus Besar Bahasa Indonesia, algoritma adalah "prosedur sistematis untuk memecahkan masalah matematis dalam langkah-langkah terbatas". Algoritma adalah langkah-langkah untuk menyelesaikan suatu masalah yang matematis.

Merujuk pada definisi tersebut, algoritma seharusnya tidak hanya dapat dimanfaatkan dalam ilmu informatika, tapi juga menyelesaikan masalah matematis yang lain yang umum ditemukan dalam kehidupan sehari-hari, termasuk di antaranya adalah menyelesaikan permainan.

Ada bermacam-macam permainan yang beredar di pasaran saat ini yang memiliki keunikan dan cara bermainnya masing-masing. Salah satu permainan yang menarik bernama Card Thief. Permainan ini mengambil ide dari *video game* klasik, Thief, tapi menghilangkan unsur grafis dan hanya menyisakan *apple core*-nya saja yang disajikan dalam bentuk permainan kartu. Menurut kreatornya, mereka sengaja membuat *game* seperti ini untuk memfokuskan pemain menyusun strategi.

Strategi. Strategi adalah sesuatu yang dapat dihitung, dikalkulasi, diperkirakan hasilnya. Dapat dibuat suatu langkah-langkah formal untuk memastikan setiap permainan akan berakhir dengan *ending* terbaik. Kira-kira, strategi apa yang paling cocok untuk menyelesaikan permainan ini? Algoritma apa yang paling sesuai untuk memainkan Card Thief?

Algoritma begitu beragam. Masing-masing memiliki tujuan dan peruntukkannya sendiri. Karena masih belum diketahui algoritma mana yang kiranya paling cocok untuk Card Thief, maka untuk memulai akan digunakan algoritma dasar yang sederhana, yaitu *greedy* untuk mengira-ngira hasil permainan.



Gambar 1: Logo Permainan Card Thief
(*sumber: dokumentasi penulis*)

II. DASAR TEORI

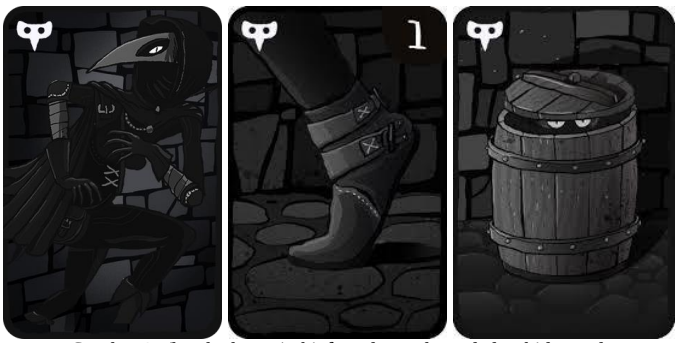
A. *Card Thief*

Card Thief adalah sebuah permainan ponsel (*mobile game*) yang dikembangkan oleh Arnold Rauers, Max Fiedler, dan Oliver Salkic. Permainan ini dapat diunduh di App Store dengan harga 1,99 dolar dan diunduh gratis di Google Play dengan video iklan. Card Thief dibuat sebagai penghormatan atas permainan klasik, Thief, dari perusahaan Looking Glass. Cara bermain keduanya memiliki kemiripan, hanya saja jika Thief merupakan permainan video (*video game*) 3D, Card Thief merupakan versi permainan dengan gaya *solitaire.*

Permainan ini dimainkan di papan berukuran 3 x 3 kartu. Kotak-kotak pada papan diisi dengan kartu dari *deck* yang memiliki berbagai jenis kartu, misalnya kartu musuh, kartu hambatan, kartu harta karun, kartu efek, dan seterusnya. Masing-masing kartu memiliki pengaruh yang berbeda. Pemain diminta membuat sebuah *path* untuk melewati kartu-kartu tersebut, mengumpulkan uang sebanyak-banyaknya, dan kabur dari para musuh tanpa terlihat.

B. *Aturan Permainan*

Pemain memiliki dua atribut penting: *stealth point* dan *treasure*. *Stealth point* menunjukkan kelihaian pemain saat ini, digunakan untuk mengendap-endap. Jika *stealth point* pemain kurang dari satu (berarti nol atau negatif), pemain bisa terlihat oleh penjaga. *Stealth point* dapat bertambah jika pemain menyertakan *hide card* atau *sneak card* dalam *path* yang dibuatnya atau menukarkan *treasure* dengan *stealth point* melalui kartu *traitor*.



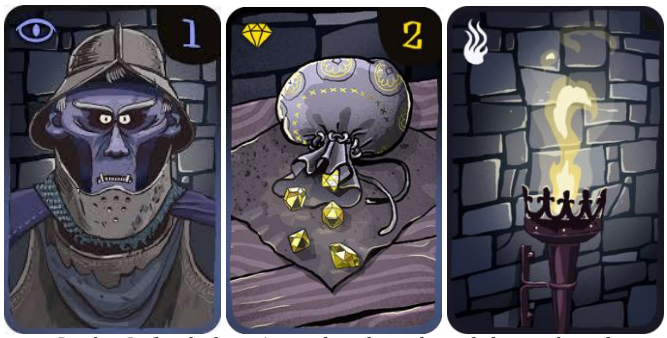
Gambar 2: (kiri ke kanan) *thief card*, *sneak card*, dan *hide card.*
Angka atas menyatakan *stealth point*. Ketiga kartu dalam keadaan gelap.
(*sumber: card-thief.fandom.com dan gamasutra.com*)

Kartu spesial lain selain *hide* dan *sneak card* adalah *torch card. Torch* akan menerangi kartu-kartu di sekitarnya. Jika kartu penjaga ada di sebelah *torch*, penjaga tersebut akan menjadi lebih kuat satu poin. Jika *hide* dan *sneak card* berada di sebelah *torch*, kedua kartu tersebut menjadi tidak berguna (tidak ada efeknya). *Torch* dapat dipadamkan dengan menyertakannya ke dalam *path*, namun memadamkan *torch* membutuhkan *stealth point*.

Kemudian, kartu penjaga memiliki dua atribut: arah jaga dan tingkat kewaspadaan *(alertness)*. Jika pemain mengusik kartu yang ada di depan seorang penjaga, tingkat kewaspadaan penjaga akan bertambah satu secara permanen. Jika pemain mengusik kartu yang ada di kiri atau kanan penjaga, arah jaganya akan berubah ke kartu yang diusik pemain. Jika pemain berada di belakang penjaga, tingkat kewaspadaan penjaga tersebut akan berkurang satu. Tingkat kewaspadaan merupakan angka yang dibandingkan dengan *stealth point* pemain untuk menentukan keadaan permainan.

Ada atribut pemain yang belum dibahas, yaitu *treasure*. *Treasure* digunakan sebagai skor permainan. Atribut ini dapat bertambah dengan menyertakan kartu *goods* ke dalam *path*. Mencuri *goods* tidak mengurangi *stealth point*, tapi dapat menyebabkan kecurigaan penjaga. Selain mencuri *goods*, *treasure* juga dapat bertambah dengan mencopet penjaga. Pencopetan hanya bisa dilakukan dalam keadaan gelap.

Perhatikan tabel 1 untuk semua kemungkinan kasus yang terjadi ketika melewati penjaga dengan mengombinasikan *stealth point* pemain, tingkat kewaspadaan penjaga, dan keadaan kartu (terang atau gelap).



Gambar 3: (kiri ke kanan) *guard card*, *goods card*, dan *torch card.*
Angka di atas menyatakan nilai kartu. Ketiga kartu dalam keadaan terang.
(*sumber: card-thief.fandom.com, gamasutra.com, dan mexer.pigsell.com*)

Tabel 1: Berbagai Keadaan dalam Permainan

Stealth point	**Datang dari**	**Dalam keadaan terang**	**Dalam keadaan gelap**
lebih besar dari tingkat kewaspadaan penjaga (*guard alertness*)	kanan atau kiri penjaga	*Stealth point* berkurang sesuai dengan *alertness* penjaga, pencopetan gagal.	*Stealth point* berkurang sesuai dengan *alertness* penjaga, pencopetan berhasil.
	belakang penjaga	*Stealth point* berkurang sesuai dengan *alertness* penjaga, pencopetan berhasil.	*Stealth point* tidak berkurang, nilai *alertness* menjadi tambahan *treasure*.
	depan penjaga	*Stealth point* berkurang sesuai dengan *alertness* penjaga, *alert* semua penjaga lain bertambah 1.	*Stealth point* berkurang sesuai dengan *alertness* penjaga, pencopetan berhasil.
lebih kecil dari tingkat kewaspadaan, tidak nol atau negatif	arah mana pun	Pemain kalah, tertangkap, permainan selesai.	
nol atau negatif	arah mana pun	Ketika kartu pemain dalam keadaan terang, pemain langsung tertangkap dan kalah. Permainan selesai.	*Stealth point* tidak berkurang, pemain dan penjaga bertukar posisi.

Permainan tidak langsung berakhir jika *stealth point* pemain menjadi nol atau negatif, permainan baru berakhir ketika pemain tertangkap oleh penjaga. Ketika *stealth point* sudah nol atau negatif, pemain memang tidak bisa konfrontasi langsung dengan penjaga (hanya bisa bertukar posisi dalam gelap), namun masih ada kartu-kartu lain seperti *torch* dan *goods* yang dapat disertakan ke dalam *path.*

Tujuan dari permainan ini adalah kabur dari kastil tanpa ketahuan oleh para penjaga. Ada dua tipe kartu yang belum disebutkan sebelumnya, yaitu kartu pintu dan kartu *exit*. Dalam keadaan terang, kedua kartu tersebut hanya dapat dilewati dari arah kuncinya dan membutuhkan *stealth point*. Dalam keadaan gelap, kartu tersebut dapat dilewati dari mana saja tanpa mengurangi *stealth point.* Jika pemain telah mencapai kartu *exit*, permainan selesai dan pemain dinyatakan menang.

Selain yang telah dijelaskan di atas, masih banyak tipe-tipe kartu yang ada, seperti *treasure chest* dan *equipment*, namun kartu-kartu tersebut tidak dibahas karena tidak digunakan dalam eksperimen ini untuk menyederhanakan permainan.

C. *Algoritma Greedy*

Algoritma *greedy* adalah salah stau algoritma yang umum digunakan untuk menyelesaikan persoalan optimasi yang menuntut pencarian solusi optimum, artinya solusi yang bernilai paling minimum atau maksimum dari semua alternatif solusi yang mungkin. Ada dua elemen persoalan optimasi, yaitu kendala (*constraint*) dan fungsi objektif.

Algoritma *greedy* bekerja dengan membentuk solusi langkah per langkah dengan mengambil pilihan yang paling menguntungkan saat ini, dengan harapan selalu mengambil pilihan optimum lokal akan mengarah ke hasil optimum global. Hal ini tergambar dalam prinsip dari algoritma *greedy*, yaitu *take what you can get now.*

Terdapat lima buah elemen dalam algoritma *greedy*:

1. Himpunan kandidat, yaitu himpunan yang memuat bagian-bagian persoalan akan dipilih menjadi solusi.
2. Himpunan solusi merupakan himpunan yang berisi kandidat-kandidat yang telah dipilih.
3. Fungsi seleksi adalah fungsi yang memilih solusi dengan nilai paling tinggi dari himpunan kandidat.
4. Fungsi layak bertugas memeriksa apakah solusi yang dipilih tidak melanggar batasan atau *constraint.*
5. Fungsi objektif adalah fungsi yang menyatakan tujuan dari persoalan yang ingin diselesaikan.

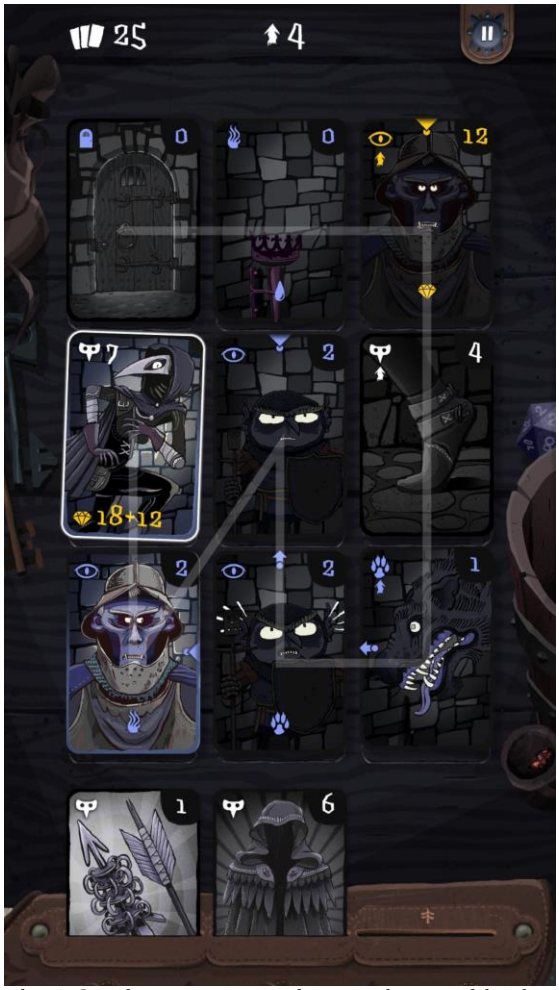
Algoritma *greedy* juga dapat diaplikasikan menggunakan strategi yang berbeda. Contohnya, untuk menyelesaikan persoalan *integer knapsack*, terdapat tiga strategi *greedy* untuk memilih objek yang akan dimasukkan:

1. *greedy by profit*, berarti selalu memilih barang yang memberikan keuntungan paling besar,
2. *greedy by weight*, berarti selalu memilih barang yang memiliki berat paling kecil, atau
3. *greedy by density*, berarti selalu memilih barang dengan densitas paling besar, di mana densitas barang didapatkan dengan menghitung *profit per weight.*

Selain itu, algoritma *greedy* juga memiliki dua tipe paradigma:

1. *Subset paradigm*, di mana algoritma *greedy* digunakan untuk memilih kandidat-kandidat yang membentuk *subset* paling optimal. Algoritma *greedy* dengan *subset paradigm* digunakan untuk menyelesaikan masalah *integer knapsack, job sequencing with deadlines*, dan *minimum cost spanning tree.*
2. *Ordering paradigm*, yaitu algoritma *greedy* digunakan untuk membuat urutan memilih kandidat yang akan menghasilkan solusi optimum. Algoritma *greedy* dengan *ordering paradigm* digunakan untuk menyelesaikan masalah *optimal storage on tapes, optimal merge patterns*, dan *single-soure shortest path.*

Penyelesaian masalah dengan algoritma *greedy* tidak selalu menghasilkan solusi optimal, namun algoritma *greedy* memiliki kompleksitas waktu cukup rendah, O(*n*) jika waktu untuk mengurutkan kandidat solusi tidak dihitung, dan relatif lebih mudah untuk diimplementasikan. Algoritma ini biasanya dimanfaatkan untuk mencari perkiraan.



Gambar 4: Contoh papan permainan dengan *path* yang sudah terbentuk (*sumber: toucharcade.com*)

III. PERSIAPAN

Pada bagian ini, akan ditentukan hal-hal apa saja yang menjadi pertimbangan dalam memilih langkah, cara menghitung *cost* tiap simpul, dan bagaimana menerapkannya ke dalam algoritma *greedy* dan *branch and bound.*

A. *Pertimbangan dan Pembatasan*

Berdasarkan studi literatur yang telah dilakukan sebelumnya, diketahui beberapa perubahan yang dapat terjadi setelah membuat suatu langkah adalah sebagai berikut:

1. *stealth point* berkurang atau bertambah
2. *treasure* berkurang atau bertambah
3. *guard alertness* berkurang atau bertambah
4. arah jaga *guard* berubah
5. pintu menjadi terbuka atau terkunci

Poin (4) dan (5), perubahan arah jaga dan kondisi kunci pintu, merupakan aspek kualitatif yang menentukan nilai langkah dalam putaran selanjutnya. Menentukan nilai untuk kedua poin tersebut relatif lebih sulit karena membutuhkan data heuristik, sehingga tidak akan dipertimbangkan dalam eksperimen ini. Maka, aspek yang akan dipertimbangkan adalah poin (1), (2), dan (3).

Tujuan akhir dari permainan adalah keluar dari kastil tanpa tertangkap penjaga, yang berarti mencapai kartu *exit.* Kartu *exit* akan muncul setelah *deck* habis, dengan catatan satu *deck* berisi kurang lebih 40 kartu. Karena proses akan sangat panjang jika eksperimen dilakukan hingga mencapai *exit*, dalam laporan ini, tujuan permainan diubah: mendapatkan *treasure* sebanyak-banyaknya dalam tiga putaran.

Kemudian, dalam satu putaran, pemain dapat membuat *path* dengan tiga sampai sembilan kartu (*variable size*). Karena ukuran *path* yang beragam membuat kemungkinan langkah menjadi sangat banyak, dalam eksperimen ini ukuran *path* dibatasi. Dalam satu kali putaran, akan ditentukan empat kartu untuk dijadikan *path*, jika tidak memungkinkan maka akan dipilih tiga, dan seterusnya.

B. *Penghitungan Cost*

Sebelumnya, telah diputuskan bahwa hal-hal yang akan dijadikan pertimbangan adalah perubahan nilai *stealth point, treasure,* dan *guard alertness.* Algoritma untuk menghitung *cost* simpul berdasarkan ketika hal tersebut adalah sebagai berikut:

$$(ag - sp) \div t \qquad (1)$$

Dengan *ag* menyatakan jumlah penjaga yang tingkat kewaspadaannya bertambah, *sp* menyatakan perubahan nilai *stealth point,* dan *t* adalah *treasure* yang didapatkan. Nilai *sp* akan negatif jika *stealth point* mengalami pengurangan dan sebaliknya.

C. *Strategi Bermain dengan Algoritma Greedy*

Bermain dengan algoritma *greedy* berarti selalu memilih kartu yang memiliki *cost* paling sedikit. Maka, untuk setiap langkah dalam permainan, akan dihitung terlebih dulu *cost* untuk masing-masing kartu, kemudian akan dipilih kartu yang memiliki *cost* minimum.

Berikut ini adalah elemen-elemen algoritma *greedy* dalam permainan:

1. Himpunan kandidat: semua kartu yang mungkin dipilih oleh pemain
2. Himpunan solusi: kartu-kartu yang sudah terpilih untuk membentuk sebuah *path*
3. Fungsi seleksi: fungsi yang memilih *cost* kartu paling minimum, *cost* kartu dihitung dengan rumus (1).
4. Fungsi layak: fungsi yang memeriksa bahwa *stealth point* pemain tidak lebih sedikit dari tingkat kewaspadaan penjaga
5. Fungsi objektif: mendapatkan *treasure* sebanyak-banyaknya

Strategi *greedy* yang dipilih adalah *greedy by density* dengan paradigma *ordering paradigm.* Sementara itu, kartu yang menyebabkan pemain tertangkap akan terkena *constraint.*

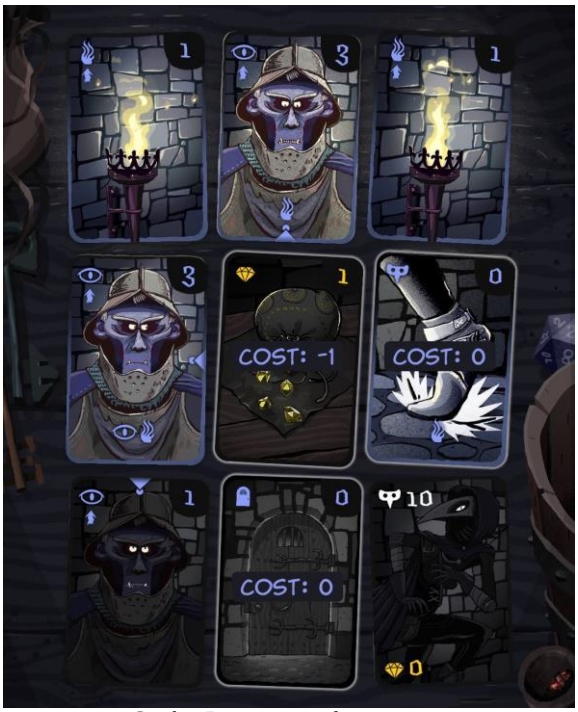
IV. Permainan dengan Algoritma Greedy

Dalam bagian ini, akan ditunjukkan langkah per langkah dari permainan menggunakan algoritma *greedy.* Seperti yang telah disebutkan sebelumnya, pada tiap putaran akan ditentukan tiga buah kartu yang akan membentuk *path.* Dalam tiap tangkapan layar, kartu-kartu yang mungkin disertakan ke dalam *path* ditandai dengan dituliskannya *cost* pada kartu. Kartu yang terpilih akan ditandai dengan adanya garis yang menghubungkan kartu tersebut dengan kartu sebelumnya.

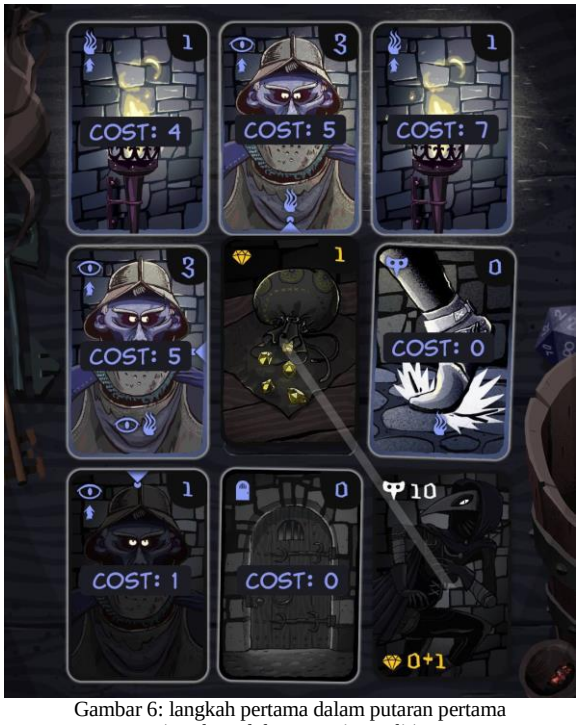
A. *Putaran Satu*

Gambar 5 menunjukkan susunan kartu untuk putaran satu. Kartu pemain berada di paling kanan bawah. Dari posisi tersebut, pemain dapat bergerak ke atas, ke kiri, atau ke kiri atas. *Cost* dari kartu yang dapat disertakan ke dalam *path* sudah dituliskan di tengah kartu.

Cost kartu di atas kartu pemain bernilai nol karena memainkan kartu tersebut tidak memberikan pengaruh apa-apa. Kartu tersebut adalah kartu *sneak* yang dapat menambahkan *stealth point*, namun tidak memberikan manfaat karena sedang dalam keadaan terang. Untuk dapat mengaktifkan kartu tersebut, *torch* di atas kartu harus dipadamkan, namun pemain tidak dapat menjangkau kartu tersebut.



Gambar 5: susunan awal putaran satu
(*sumber: dokumentasi penulis*)

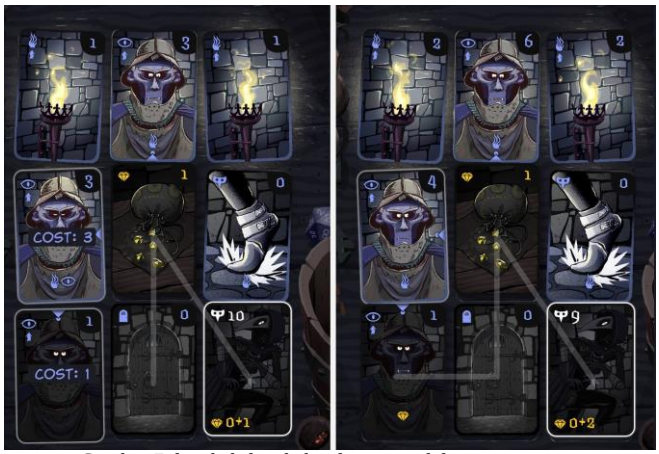


Gambar 6: langkah pertama dalam putaran pertama
(*sumber: dokumentasi penulis*)

Kartu di sebelah kiri pemain yang merupakan *door card* dapat dilewati tanpa mengurangi *stealth point* karena sedang dalam keadaan gelap. Namun, ada kartu dengan *cost* yang lebih murah, yaitu kartu *goods* bagian tengah papan karena menambahkan satu *treasure.* Maka, kartu yang dipilih adalah kartu tengah.

Dari kartu tengah, pemain bisa menuju kartu mana saja. *Cost* dari masing-masing kartu yang dihitung dengan rumus (1) juga sudah dituliskan di tengah setiap kartu pada gambar 6. Dapat dilihat bahwa kartu dengan *cost* terendah adalah kartu di kanan tengah dan bawah tengah dengan nilai nol. Dengan demikian, *greedy* dapat memilih antara dua kartu tersebut.

Gambar 7 mengilustrasikan langkah-langkah selanjutnya dalam putaran dua.



Gambar 7: langkah-langkah selanjutnya dalam putaran satu
(*sumber: dokumentasi penulis*)

Sama seperti langkah-langkah sebelumnya, *cost* kartu akan dihitung dengan rumus (1). Kemudian, dengan algoritma *greedy*, dipilih kartu dengan *cost* paling sedikit. Setelah pemain membuat *path* sampai ke kartu kiri bawah, pemain menyatakan putaran satu selesai.

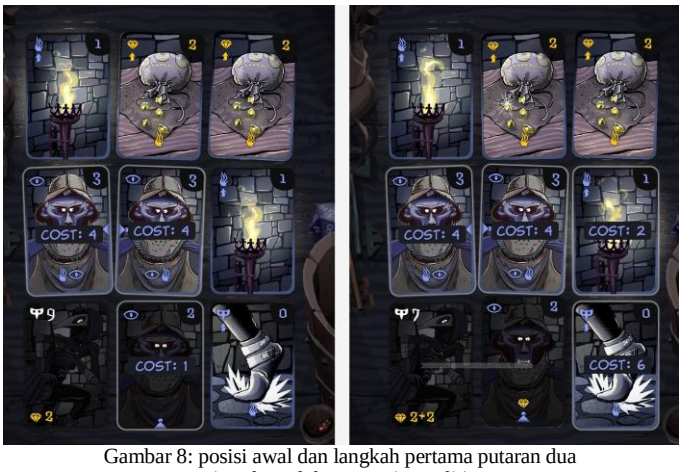
B. Putaran Dua

Setelah menyatakan satu putaran selesai, kartu-kartu dalam *path* akan hilang dan kartu pemain akan berpindah posisi ke posisi akhir *path.* Posisi kosong dari kartu-kartu yang hilang akan diganti dengan kartu-kartu dari *deck.* Susunan kartu menjadi seperti gambar 8. Dari posisi di kiri bawah, pemain dapat menuju ke kartu tengah, kiri tengah, atau bawah tengah.

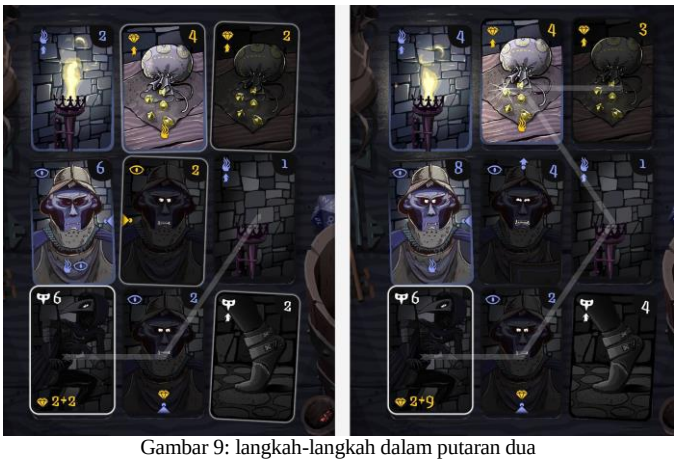
Algoritma *greedy* akan memilih kartu bawah tengah yang memiliki *cost* paling minimum. Setelah itu, akan dipilih kartu kanan tengah yang juga merupakan kartu yang dapat dituju dengan *cost* paling minimum.

Langkah-langkah berikutnya dalam putaran dua dapat dilihat pada gambar 9.

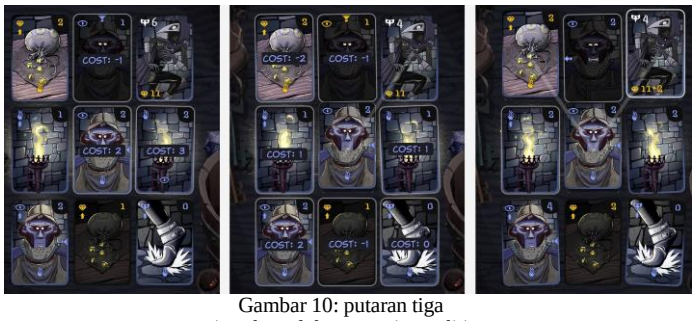
Pada langkah kedua, semua kartu yang mungkin dituju oleh pemain mendatangkan keuntungan: kartu kanan atas dan kartu tengah memberikan dua *treasure*, kartu atas tengah memberikan empat *treasure*, dan kartu kanan bawah memberi-



Gambar 8: posisi awal dan langkah pertama putaran dua
(*sumber: dokumentasi penulis*)



Gambar 9: langkah-langkah dalam putaran dua
(*sumber: dokumentasi penulis*)



Gambar 10: putaran tiga
(*sumber: dokumentasi penulis*)

kan dua *stealth point*. Dengan algoritma *greedy*, kartu yang dipilih adalah kartu kanan tengah yang memberi empat *treasure*, kemudian kartu kanan atas yang nilainya bertambah menjadi tiga *treasure* sehingga total keuntungan pemain untuk putaran ini bernilai sembilan *treasure.*

C. Putaran Tiga

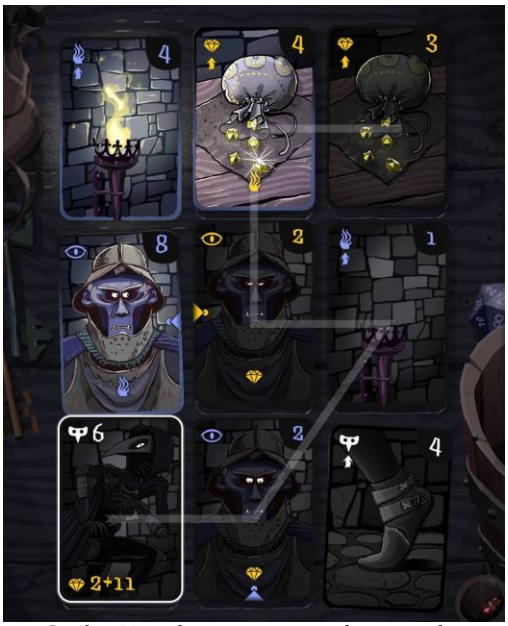
Cara untuk menentukan langkah pada putaran tiga sama dengan cara-cara sebelumnya. Susunan awal dan langkah-langkah di putaran tiga dapat dilihat pada gambar 10.

V. ANALISIS

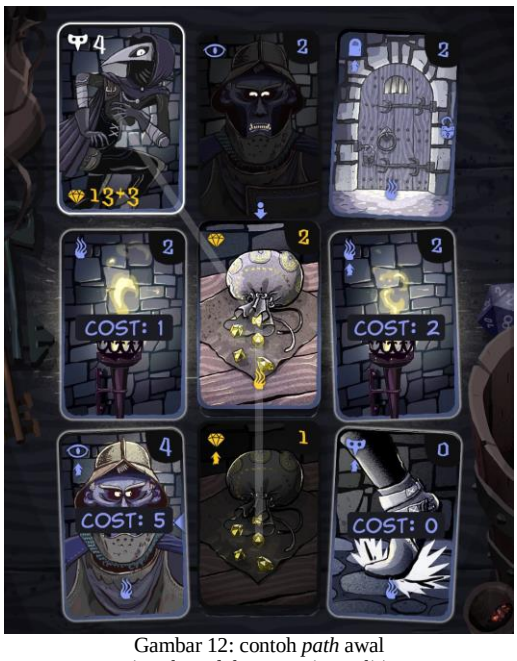
Disebutkan dalam dasar teori bahwa algoritma *greedy* tidak selalu memberikan hasil yang optimum. Saat eksperimen, ditemukan beberapa kasus yang membuktikan pernyataan itu.

Pada putaran dua, *path* yang dihasilkan bukan merupakan *path* yang optimum karena sebenarnya pemain bisa mendapatkan 11 *treasure.* Setelah langkah kedua, pemain dapat mencopet terlebih dahulu dari penjaga di tengah, baru mengambil dua kartu *treasure* di atas. Ilustrasi untuk *path* optimum dapat dilihat pada gambar 11.

Kemudian, selain tidak selalu memberikan *path* yang optimum, kekurangan lain dari algoritma *greedy* adalah tidak memperhi-



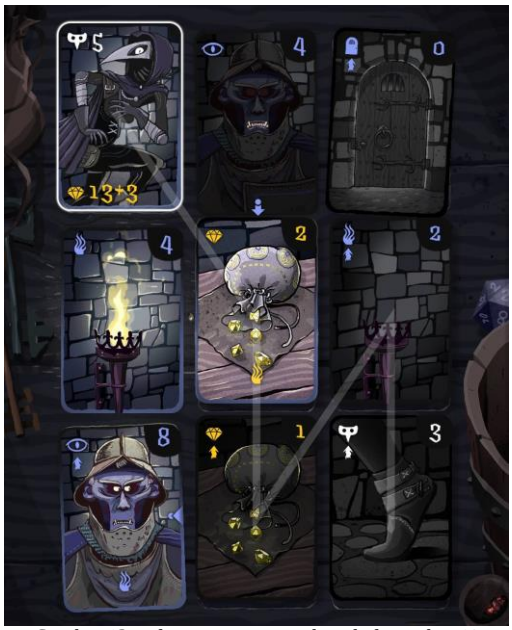
Gambar 11: *path* yang optimum pada putaran dua
(*sumber: dokumentasi penulis*)



Gambar 12: contoh *path* awal
(*sumber: dokumentasi penulis*)

tungkan pilihan-pilihan di masa depan. Dalam permainan Card Thief, pemilihan kartu saat ini sangat berpengaruh terhadap kartu-kartu yang akan dipilih setelahnya. Sebagai contoh, perhatikan *path* pada gambar 12.:

Berdasarkan algoritma *greedy*, kartu yang akan terpilih adalah kartu kanan bawah yang memiliki *cost* nol. Padahal, kartu kanan bawah tersebut adalah kartu *sneak* yang tidak berfungsi karena berada dalam keadaan gelap. Untuk dapat mengaktifkan kartu tersebut, *torch* di bagian atas kartu dapat dimatikan terlebih dahulu. Ilustrasi dari kasus ini dapat dilihat pada gambar 13.



Gambar 13: solusi optimum untuk path di gambar 12
(*sumber: dokumentasi penulis*)

VI. Simpulan dan Saran

Berdasarkan pembahasan, eksperimen, dan analisis yang telah dilakukan sebelumnya, dapat ditarik kesimpulan bahwa algoritma *greedy* memang relatif cepat dan praktis dibanding algoritma lain, tapi kurang cocok untuk diaplikasikan pada permainan Card Thief.

Selain hasil yang diberikan tidak selalu optimal, ada beberapa aspek dari permainan yang tidak bisa digambarkan oleh *greedy*. Salah satunya adalah *greedy* tidak bisa mewakili perubahan kondisi kartu yang disebabkan oleh pergerakan pemain. Padahal, perubahan kondisi kartu ini akan berpengaruh terhadap langkah selanjutnya. Algoritma *greedy* tidak bisa mempertimbangkan bagaimana pilihan kartu saat ini bisa mempengaruhi pilihan kartu selanjutnya.

Untuk menangani kondisi permainan yang selalu berubah dalam setiap langkah, akan lebih cocok menggunakan algoritma program dinamis untuk mencari *path* yang menghasilkan solusi optimum. Namun, karena dalam permainan ini terdapat begitu banyak *state* dan *stage*, membuat program dinamis untuk menyelesaikan permainan ini pun merupakan solusi yang kurang realistis.

Sebagai penutup, berikut kesimpulan akhir dari makalah ini: mengaplikasikan algoritma untuk mencari solusi terbaik dari Card Thief adalah hal yang sangat sulit dan berharga mahal. Oleh karena itu, Card Thief paling baik dimainkan tanpa memperhitungkan algoritma apa pun pada saat senggang sebagai *brain teaser* yang menyenangkan.

Tautan Video

Untuk memberi gambaran yang lebih jelas tentang penelitian ini, telah dibuat sebuah video penjelasan yang dapat diakses melalui tautan: youtu.be/I80pTKFaHcI

Sambutan

Saya mengucapkan terima kasih kepada Tuhan atas berkat dan karunia-Nya, saya dimampukan untuk menyelesaikan makalah ini. Saya juga berterima kasih kepada Dr. Masayu Leylia Khodra, ST., MT., dosen mata kuliah Strategi Algoritma saya yang selalu memastikan mahasiswanya mengerti sebelum melanjutkan materi. Terima kasih untuk kesabaran Ibu selama satu semester ini.

Daftar Pustaka

[1] E. Horowitz, S. Sahmi, S. Rajasekaran, Computer Algorithms. Houndmills: Computer Science Press, 1997, pp.197-198,379-381.

[2] R. Munir, Diktat Strategi Algoritmik.. Bandung: Departemen Teknik Informatika Institut Teknologi Bandung, 2004.

[3] Situs resmi Card Thief, card-thief.com, diakses 1 Mei 2020.

[4] Situs penggemar Card Thief, card-thief.fandom.com, diakses 1 Mei 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, 3 Mei 2020

Anindya Prameswari Ekaputri
135 18 034