TUGAS AKHIR – IF184802

PENERAPAN ALGORITMA *GREEDY* PADA PENYELESAIAN PERMASALAHAN TIMUS ONLINE JUDGE 2082 – POKER

JEREMIA RONALDO MANURUNG
05111640000102

Dosen Pembimbing I:
Rully Soelaiman, S.Kom., M.Kom.

Dosen Pembimbing II:
Yudhi Purwananto, S.Kom., M.Kom.

DEPARTEMEN TEKNIK INFORMATIKA
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya
2020

TUGAS AKHIR – IF184802

PENERAPAN ALGORITMA *GREEDY* PADA PENYELESAIAN PERMASALAHAN TIMUS ONLINE JUDGE 2082 - POKER

JEREMIA RONALDO MANURUNG
05111640000102

Dosen Pembimbing I:
Rully Soelaiman, S.Kom., M.Kom.

Dosen Pembimbing II:
Yudhi Purwananto, S.Kom., M.Kom.

DEPARTEMEN TEKNIK INFORMATIKA
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya
2020

[*Halaman ini sengaja dikosongkan*]

UNDERGRADUATE THESIS – IF184802

IMPLEMENTATION OF GREEDY ALGORITHM IN SOLVING TIMUS ONLINE JUDGE PROBLEM 2082 – POKER

JEREMIA RONALDO MANURUNG
05111640000102

Supervisor I
Rully Soelaiman, S.Kom., M.Kom.

Supervisor II
Yudhi Purwananto, S.Kom., M.Kom.

DEPARTMENT OF INFORMATICS
Faculty of Information and Communication Technology
Institut Teknologi Sepuluh Nopember
Surabaya
2020

[*Halaman ini sengaja dikosongkan*]

LEMBAR PENGESAHAN

PENERAPAN ALGORITMA GREEDY PADA PENYELESAIAN PERMASALAHAN TIMUS ONLINE JUDGE 2082 – POKER

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Algoritma dan Pemrograman
Program Studi S-1
Departemen Teknik Informatika
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
2020

Oleh:
Jeremia Ronaldo Manurung
NRP: 051116 40000 102

Disetujui oleh Dosen Pembimbing Tugas Akhir:

Rully Soelaiman, S.Kom., M.Kom.
NIP. 197002131994021001 ……
(Pembimbing 1)

Yudhi Purwananto, S.Kom., M.Kom.
NIP. 197007141997031002 ……
(Pembimbing 2)

SURABAYA
2020

[*Halaman ini sengaja dikosongkan*]

PENERAPAN ALGORITMA *GREEDY* PADA PENYELESAIAN PERMASALAHAN TIMUS ONLINE JUDGE 2082 – POKER

Nama Mahasiswa	:	Jeremia Ronaldo Manurung
NRP	:	05111640000102
Departemen	:	Teknik Informatika, Fakultas Teknologi Elektro dan Informatika Cerdas, ITS
Dosen Pembimbing 1	:	Rully Soelaiman, S.Kom., M.Kom.
Dosen Pembimbing 2	:	Yudhi Purwananto, S.Kom., M.Kom.

ABSTRAK

Dengan berkembangnya teknologi, game juga membutuhkan perkembangan dalam bidang teknologi. Salah satunya adalah game Poker. Poker adalah sebuah permainan kartu remi yang terdiri dari 52 jenis kartu. Setiap individu bertaruh satu sama lain dengan kepingan plastik. Pemenang dari permainan akan mendapatkan semua kepingan plastik yang dipertaruhkan oleh setiap pemain. Permasalahan 2082 – Poker menjelaskan implementasi checker bot pada salah satu jenis game Poker, yaitu Texas Holdem Poker. Bot ini berfungsi untuk mendata jumlah keping setiap pemain serta kejadian-kejadian yang terjadi selama permainan. Pendekatan penulis untuk menyelesaikan permasalahan tersebut adalah dengan menggunakan algoritma greedy untuk menentukan jumlah keping yang dipakai saat melakukan aksi. Solusi yang dibuat cukup efisien dengan rata-rata waktu penyelesaian 0,0166 detik dengan penggunaan memori 470,8 KB.

Kata kunci: Greedy, Poker, Texas Holdem Poker.

[*Halaman ini sengaja dikosongkan*]

IMPLEMENTATION OF GREEDY ALGORITHM IN SOLVING TIMUS ONLINE JUDGE PROBLEM 2082 – Poker

Name	: Jeremia Ronaldo Manurung
NRP	: 05111640000102
Department	: Informatics, Faculty of Intelligent Electrical and Informatics Technology, ITS
Supervisor 1	: Rully Soelaiman, S.Kom., M.Kom.
Supervisor 2	: Yudhi Purwananto, S.Kom., M.Kom.

ABSTRACT

With the development of technology, games also require developments in the field of technology. One of them is Poker. Poker is a playing card game that consists of 52 types of cards. Each individual bet with each other with plastic chips. The winner of the game will get all the plastic chips at stake by each player. Problem 2082 – Poker explains the implementation of a checker bot in one type of Poker game, namely Texas Holdem Poker. This bot serves to record the number of pieces of each player and the events that occur during the game. The author's approach to solve this problem is to use the greedy algorithm to determine the amount of pieces used when performing actions. The solution made is quite efficient with an average completion time of 0.0166 seconds with 470.8 KB of memory usage.

Keywords: Greedy, Poker, Texas Holdem Poker.

[*Halaman ini sengaja dikosongkan*]

KATA PENGANTAR

Puji syukur penulis ucapkan kepada Tuhan Yang Maha Esa atas pimpinan, penyertaan, dan karunia-Nya sehingga penulis dapat menyelesaikan tugas akhir yang berjudul:

PENERAPAN ALGORITMA *GREEDY* PADA PENYELESAIAN PERMASALAHAN TIMUS ONLINE JUDGE 2082 – Poker

Pengerjaan tugas akhir ini dilakukan untuk memenuhi salah satu syarat meraih gelar Sarjana di Departemen Teknik Informatika, Fakultas Teknologi Elektro dan Informatika Cerdas Institut Teknologi Sepuluh Nopember.

Dengan selesainya tugas akhir ini diharapkan apa yang telah dikerjakan penulis dapat memberikan manfaat bagi perkembangan ilmu pengetahuan terutama di bidang teknologi informasi serta bagi diri penulis sendiri selaku peneliti.

Penulis mengucapkan terima kasih kepada semua pihak yang telah memberikan dukungan baik secara langsung maupun tidak langsung selama penulis mengerjakan tugas akhir maupun selama menempuh masa studi antara lain:

1. Terima kasih kepada Tuhan Yang Maha Esa, di mana penulis diberi kesempatan, kesehatan, dan umur untuk menempuh kuliah di sini.
2. Bapak Rully Soelaiman, S.Kom., M.Kom. selaku Dosen Pembimbing yang telah membimbing saya selama masa kuliah maupun selama penyelesaian tugas akhir ini, Dosen yang paling perhatian kepada saya dalam memberi ilmu, nasihat, dan motivasi selama berada menempuh kuliah di Departemen Teknik Informatika ITS.
3. Bapak Yudhi Purwananto, S.Kom., M.Kom. selaku dosen pembimbing yang telah memberikan ilmu dan masukan kepada penulis.
4. Bapak, Ibu, dan keluarga penulis yang selalu memberikan dukungan, perhatian, dan kasih sayang

bagi penulis yang menjadi semangat selama perkuliahan maupun pengerjaan tugas akhir.

5. Denny Rengganis, Rimba Azhara, dan Farras Rahmatullah yang sangat membantu penulis selama perkuliahan di Teknik Informatika
6. Yoshima Syach Putri yang membantu penulis dalam pembuatan buku tugas akhir.
7. Teman-teman angkatan 2016 Departemen Teknik Informatika ITS yang telah menemani penulis selama 4 tahun masa perkuliahan.
8. Serta pihak-pihak lain yang tidak dapat disebutkan di sini yang telah banyak membantu penulis dalam penyusunan tugas akhir ini.

Penulis mohon maaf apabila masih ada kekurangan pada tugas akhir ini. Penulis juga mengharapkan kritik dan saran yang membangun untuk pembelajaran dan perbaikan di kemudian hari. Semoga melalui tugas akhir ini penulis dapat memberikan kontribusi dan manfaat yang sebaik-baiknya.

Surabaya, Mei 2020

Jeremia Ronaldo Manurung

DAFTAR ISI

[*Halaman ini sengaja dikosongkan*]

DAFTAR GAMBAR

DAFTAR TABEL

[*Halaman ini sengaja dikosongkan*]

DAFTAR *PSEUDOCODE*

[*Halaman ini sengaja dikosongkan*]

DAFTAR KODE SUMBER

[*Halaman ini sengaja dikosongkan*]

BAB I
PENDAHULUAN

Pada bab ini akan dipaparkan mengenai garis besar tugas akhir yang meliputi latar belakang, tujuan, rumusan masalah, batasan permasalahan, metodologi pembuatan tugas akhir, dan sistematika penulisan buku tugas akhir ini.

1.1 Latar Belakang

Game merupakan jenis hiburan yang sangat disukai oleh berbagai kalangan. Pada beberapa jenis *game*, terdapat beberapa kejadian yang terjadi selama *game* berlangsung yang akan menentukan hasil akhir dari *game*. Dulu, kejadian-kejadian ini dilacak secara manual. Dengan berkembangnya teknologi, sistem *tracking* pada *game* dapat dilakukan dengan bantuan *bot*. Salah satu jenis *game* yang dapat menggunakan *bot* untuk *tracking* kejadian yang terjadi adalah *Texas Holdem Poker* [1], salah satu jenis dari *Poker*.

Poker adalah sebuah permainan kartu remi yang terdiri dari 52 jenis kartu. Dalam permainan ini, setiap individu bermain melawan satu sama lain di mana setiap pemain akan bertaruh dengan kepingan plastik. Pemenang dari permainan ini akan mendapatkan semua kepingan plastik yang dipertaruhkan oleh setiap pemain. Terdapat banyak jenis dari *Poker*, salah satunya adalah *Texas Holdem Poker*.

Terdapat dua sampai sembilan pemain dalam *Texas Holdem Poker*. Pemain duduk pada sebuah meja melingkar dan setiap pemain memiliki sejumlah kepingan plastik. Salah satu dari pemain akan menjadi *dealer*, yaitu pemain yang membagikan kartu. *Dealer* akan membagikan dua buah kartu secara tertutup ke setiap pemain di mana kartu tersebut hanya dapat dilihat oleh pemain yang memilikinya. Kemudian, setiap pemain diberi kesempatan untuk bertaruh atau menyerah. Setelah itu, *dealer* akan meletakkan tiga kartu pertama secara terbuka di meja. Setiap pemain akan diberi kesempatan untuk bertaruh atau

menyerah lagi. Lalu, *dealer* akan meletakkan satu kartu lagi secara terbuka di meja dan setiap pemain diberi kesempatan lagi untuk bertaruh atau menyerah. Setelah pemain bertaruh atau menyerah, *dealer* akan meletakkan satu kartu lagi dan setiap pemain diberi kesempatan lagi untuk bertaruh atau menyerah. Pada akhir permainan, setiap pemain yang belum menyerah akan menunjukkan kartu mereka dan pemain yang memiliki nilai terbesar dari kombinasi kartu yang dimilikinya dan lima kartu di meja akan menjadi pemenang dari permainan.

Topik tugas akhir ini berdasarkan pada permasalahan 2082 – Poker [2] pada Timus Online Judge. Permainan *Texas Holdem Poker* pada topik tugas akhir ini berbeda dengan *Texas Holdem Poker* pada umumnya. Pada *Texas Holdem Poker* dalam tugas akhir ini, nilai dari kombinasi kartu dan siapa yang menjadi pemenang tidak dipermasalahkan dalam permainan. Hal ini disebabkan karena tugas akhir ini lebih berfokus pada alur kejadian pada permainan. Oleh karena itu, buku ini tidak menjelaskan mengenai nilai dari kombinasi kartu dan tahap pembagian kartu pada permainan.

Pada permasalahan 2082 – Poker, terdapat sebuah *bot* pada *Texas Holdem Poker*. *Bot* ini memiliki peran untuk mendaftarkan kejadian-kejadian yang telah terjadi dalam permainan sehingga *dealer* dapat melakukan pengecekan atas tindakan-tindakan yang telah terjadi dalam permainan untuk menghindari kecurangan dan kelalaian. Daftar tindakan-tindakan ini juga dapat digunakan oleh pemain-pemain untuk menilai apakah tindakan yang telah dilakukannya sudah tepat. Dalam permainan ini, *bot* juga berperan sebagai pemain. Berikut adalah sistem *tracking* yang dilakukan oleh *bot*. Awalnya, *bot* mengetahui informasi awal dari permainan dan informasi mengenai jumlah keping awal yang dimiliki setiap pemain sebelum permainan dimulai, nama setiap pemain, serta tindakan apa saja yang sudah dilakukan oleh pemain lainnya. *Bot* melakukan *tracking* ketika *bot* tersebut akan melakukan gilirannya. Saat melakukan *tracking*, *bot* mendapatkan informasi mengenai jumlah keping saat ini yang dimiliki setiap pemain sesaat sebelum giliran *bot*. *Bot* akan

menelusuri apa saja yang sudah terjadi sehingga diketahui semua aksi yang terjadi dari kondisi awal sampai kondisi saat ini.

Topik tugas akhir ini mengacu pada algoritma dengan metode *greedy* [3] yang digunakan *bot* dalam mengetahui kejadian-kejadian yang terjadi untuk mencapai kondisi pada saat *bot* melakukan *tracking*.

1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam tugas akhir ini adalah sebagai berikut:

1. Bagaimana cara *bot* mengetahui kejadian-kejadian apa yang sudah terjadi untuk membantu permainan *Texas Holdem Poker*?
2. Bagaimana cara *bot* mengetahui bahwa seorang pemain akan melakukan aksi secara *greedy* atau tidak pada saat gilirannya?

1.3 Batasan Masalah

Permasalahan yang dibahas pada tugas akhir ini memiliki beberapa batasan, yaitu sebagai berikut:

1. Pengaplikasian *greedy* pada *problem* 2082 – Poker.
2. Pembuktian kebenaran didasarkan pada hasil *submission* di sistem penilaian daring Timus Online Judge.
3. Implementasi menggunakan bahasa pemrograman C++.

Batasan pada Timus Online Judge:

1. *sb* sebagai *small blinds* dengan rentang antara 10 sampai dengan 500.
2. *bb* sebagai *big blinds* dengan rentang antara 20 sampai dengan 1000.
3. *a* sebagai *ante* dengan batas maksimum 100.
4. *sb* lebih kecil daripada *bb*.
5. *n* sebagai jumlah pemain dengan rentang antara 1 sampai dengan 9.

6. *k* sebagai jumlah tindakan yang diketahui *bot* dengan batas maksimum 1000.
7. Jumlah keping setiap pemain dengan rentang antara 1 sampai dengan 20000.
8. Batas maksimum panjang nama pemain adalah 20 dan bukan *dealing*.
9. Batas maksimum waktu perangkat lunak berjalan adalah dua detik.
10. Batas memori perangkat lunak adalah 256 MB.

1.4 Tujuan

Tujuan dari tugas akhir ini antara lain:

1. Mengevaluasi algoritma yang dirancang dengan metode *greedy* untuk permasalahan Timus Online Judge 2082 – Poker.
2. Hasil evaluasi di atas diimplementasikan untuk menyelesaikan permasalahan Timus Online Judge 2082 – Poker.

1.5 Manfaat

Manfaat dari pembuatan tugas akhir ini adalah sebagai berikut:

1. Mampu memberikan pemahaman dan penjelasan mencari desain algoritma yang efisien untuk penyelesaian kasus Timus Online Judge 2082 – Poker.
2. Mampu memberikan pemahaman dan penjelasan analisis algoritma *greedy* pada permasalahan pada kasus Timus Online Judge 2082 – Poker.
3. Memberikan hasil uji coba dari implementasi algoritma *greedy* yang akan dilakukan.

1.6 Metodologi

Langkah-langkah yang ditempuh dalam pengerjaan tugas akhir ini yaitu:

1.6.1 Penyusunan Proposal Tugas Akhir

Pada tahap ini dilakukan penyusunan proposal tugas akhir yang berisi permasalahan pada permasalahan Timus Online Judge 2082 – Poker serta gagasan solusi yang akan dibahas pada tugas akhir ini.

1.6.2 Studi Literatur

Pada tahap ini dilakukan pencarian informasi dan studi literatur yang relevan untuk dijadikan referensi dalam melakukan pengerjaan tugas akhir. Informasi didapatkan dari materi-materi yang berhubungan dengan algoritma *greedy*. Informasi tersebut didapatkan dari buku, internet, dan materi kuliah yang berhubungan dengan metode yang akan digunakan. Studi literatur yang digunakan adalah studi algoritma *greedy*.

1.6.3 Desain dan Analisis

Pada tahap ini dilakukan desain rancangan algoritma yang digunakan dalam solusi untuk pemecahan masalah klasik Timus Online Judge 2082 – Poker.

1.6.4 Implementasi Algoritma

Pada tahap ini dilakukan implementasi dari rancangan desain ke dalam bentuk program dalam bahasa pemrograman C++.

1.6.5 Uji Coba Kebenaran

Pada tahap ini dilakukan uji coba kebenaran implementasi yang dilakukan pada sistem penilaian daring Timus Online Judge.

1.6.6 Penyusunan Buku Tugas Akhir

Pada tahap ini dilakukan penyusunan laporan yang menjelaskan dasar teori dan metode yang digunakan dalam tugas akhir ini serta hasil dari implementasi aplikasi algoritma yang telah dibuat.

1.7 Sistematika Penulisan

Buku tugas akhir ini merupakan laporan secara lengkap mengenai tugas akhir yang telah dikerjakan baik dari sisi teori, rancangan, maupun implementasi sehingga memudahkan bagi pembaca dan juga pihak yang ingin mengembangkan lebih lanjut. Sistematika penulisan buku tugas akhir secara garis besar dapat dilihat seperti dibawah ini.

Bab I Pendahuluan

Bab ini berisi penjelasan latar belakang, rumusan masalah, batasan masalah, dan tujuan pembuatan tugas akhir. Selain itu, metodologi pengerjaan dan sistematika penulisan laporan tugas akhir juga dijelaskan di dalamnya.

Bab II Dasar Teori

Bab ini berisi penjelasan secara detail mengenai dasar-dasar penunjang dan teori-teori yang digunakan untuk mendukung pembuatan tugas akhir ini.

Bab III Desain dan Analisis

Bab ini berisi desain algoritma dan struktur data yang digunakan dalam penyelesaian permasalahan.

Bab IV Implementasi

Bab ini berisi penjelasan implementasi dalam bahasa pemrograman C++ berdasarkan desain algoritma yang telah dilakukan pada tahap desain.

Bab V Uji Coba dan Evaluasi

Bab ini berisi uji coba kebenaran implementasi yang dilakukan pada sistem penilaian daring Timus Online Judge.

Bab VI Kesimpulan dan Saran

Bab ini merupakan bab terakhir yang menjelaskan kesimpulan dari hasil uji coba yang dilakukan dan saran untuk pengembangan perangkat lunak ke depannya.

[*Halaman ini sengaja dikosongkan*]

BAB II
DASAR TEORI

Bab ini membahas mengenai teori-teori dasar yang digunakan dalam tugas akhir. Pada bagian awal, bab ini menjelaskan mengenai permainan *Texas Holdem Poker*, kemudian deskripsi permasalahan 2082 – Poker pada Timus Online Judge, dilanjutkan dengan menjelaskan deskripsi umum teori yang akan digunakan pada tugas akhir ini.

2.1 *Texas Holdem Poker*

Poker adalah sebuah permainan kartu remi yang terdiri dari 52 jenis kartu. Setiap individu bertaruh satu sama lain dengan kepingan plastik. Pemenang dari permainan akan mendapatkan semua kepingan plastik yang dipertaruhkan oleh setiap pemain. Terdapat banyak jenis dari *Poker*, dan untuk permasalahan ini yang dipakai adalah *Texas Holdem Poker*.

Terdapat dua sampai sembilan pemain dalam *Texas Holdem Poker*. Pemain duduk pada sebuah meja melingkar dan setiap pemain memiliki sejumlah kepingan plastik. Salah satu dari pemain akan menjadi *dealer*, yaitu pemain yang membagikan kartu. *Dealer* akan membagikan dua buah kartu secara tertutup ke setiap pemain di mana kartu tersebut hanya dapat dilihat oleh pemain yang memilikinya. Kemudian, setiap pemain diberi kesempatan untuk bertaruh atau menyerah. Setelah itu, *dealer* akan meletakkan tiga kartu pertama secara terbuka di meja. Setiap pemain akan diberi kesempatan untuk bertaruh atau menyerah lagi. Lalu, *dealer* akan meletakkan satu kartu lagi secara terbuka di meja dan setiap pemain diberi kesempatan lagi untuk bertaruh atau menyerah. Setelah pemain bertaruh atau menyerah, *dealer* akan meletakkan satu kartu lagi dan setiap pemain diberi kesempatan lagi untuk bertaruh atau menyerah. Pada akhir permainan, setiap pemain yang belum menyerah akan menunjukkan kartu mereka dan pemain yang memiliki nilai

terbesar dari kombinasi kartu yang dimilikinya dan lima kartu di meja akan menjadi pemenang dari permainan.

2.1.1 Perbedaan *Texas Holdem Poker* di Tugas Akhir

Berikut adalah perbedaan – perbedaan antara *Texas Holdem Poker* pada umumnya dengan *Texas Holdem Poker* di tugas akhir:

1. Nilai dari kombinasi kartu dan siapa yang menjadi pemenang tidak dipermasalahkan dalam permainan. Hal ini disebabkan karena tugas akhir lebih berfokus pada alur kejadian pada permainan. Oleh karena itu, buku ini tidak menjelaskan mengenai nilai dari kombinasi kartu dan tahap pembagian kartu pada permainan.

2.1.2 Istilah pada *Texas Holdem Poker*

Dalam *Texas Holdem Poker*, terdapat beberapa istilah yang dipakai. Berikut adalah penjelasan dari setiap istilah tersebut:

- *Dealer*: pemain yang membagikan kartu.
- *Ante*: nilai keping yang harus dipertaruhkan pemain untuk mengikuti permainan.
- *Blinds*: nilai keping yang harus dipertaruhkan pemain tertentu. Terdapat *big blinds* dan *small blinds.* Besar *big blinds* adalah dua kali lipat dari *small blinds*.
- *Street*: tahap pada ronde permainan. Terdapat empat *street* secara urut yaitu *preflop*, *flop*, *turn*, dan *river*.
- *Current bet*: taruhan maksimum pada *street* saat ini.
- *All-in*: bertaruh semua keping yang dimiliki pemain itu.
- *Fold*: menyerah.
- *Call*: bertaruh sehingga taruhan pemain itu setara dengan *current bet*. Hal ini dapat dilakukan pemain jika *current bet* lebih besar dari nol dan pemain telah bertaruh lebih kecil dari *current bet*. Pemain tetap dapat melakukan *call*

walaupun jumlah keping yang dimiliki lebih kecil dari *current bet*. Namun, pemain dalam kondisi ini hanya dapat melakukan *call* secara *all-in*.

- *Bet*: bertaruh. Hal ini dapat dilakukan pemain jika *current bet* senilai nol sehingga *current bet* sebesar jumlah keping yang dipertaruhkan.
- *Raise*: menambahkan keping sehingga taruhan pemain itu lebih besar dari *current bet*. Hal ini dapat dilakukan pemain jika current bet lebih besar dari nol. Nilai *current bet* akan menjadi sebesar taruhan pemain itu. Pemain tidak dapat melakukan *raise* jika jumlah keping yang dimiliki ditambah taruhan yang telah dibayarkan lebih kecil dari *current bet*.
- *Check*: melewatkan taruhan. Hal ini dapat dilakukan pemain jika *current bet* sebesar nol maupun taruhan pemain itu sebesar *current bet*. Pemain akan memilih untuk melakukan *check* daripada *fold*.
- *Dealing*: pergantian *street*. Urutannya adalah *preflop* menjadi *flop*, *flop* menjadi *turn*, dan *turn* menjadi *river*.

2.1.3 Alur dan Aturan Permainan *Texas Holdem Poker*

Berikut adalah alur dan aturan permainan *Texas Holdem Poker* pada tugas akhir ini:

1. Setiap pemain memiliki sejumlah keping terlebih dahulu dan harus bertaruh sebesar *ante*.
2. Dua pemain di sebelah kiri *dealer* harus bertaruh *blinds*. Pemain di sebelah kiri dealer bertaruh *small blinds* dan pemain di sebelah kiri pemain yang bertaruh *small blinds* harus bertaruh *big blinds*. Jika hanya terdapat dua pemain, maka pemain yang bertaruh sebesar *small blinds* adalah *dealer*. Jika pemain yang harus bertaruh *blinds*

ternyata memiliki jumlah keping lebih kecil daripada *blinds* yang ditentukan, maka pemain itu harus bertaruh sejumlah semua keping yang dimilikinya.

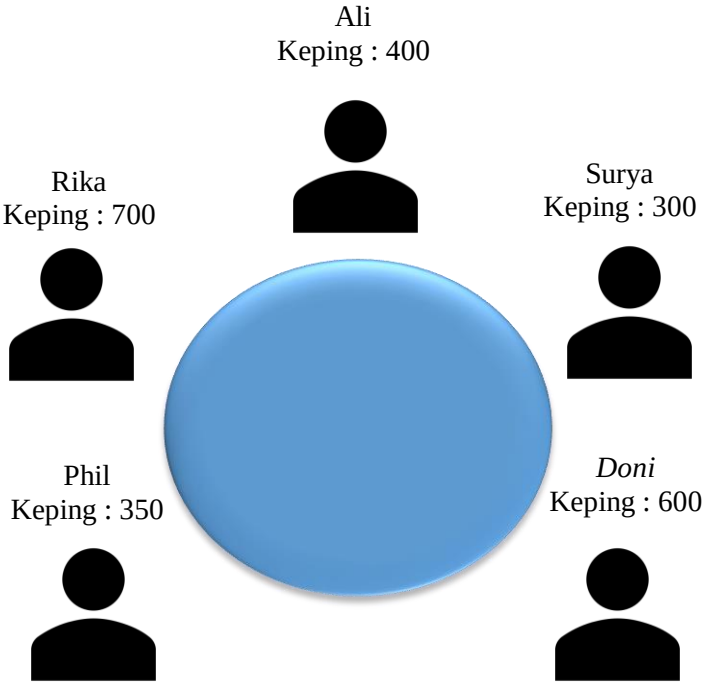
3. Permainan dimulai dengan *street* awal adalah *preflop*. Pada *preflop*, *current bet* sebesar *big blinds* dan taruhan pemain yang bertaruh pada poin ke-2 sebesar jumlah keping yang sudah dipertaruhkan. Pada *street* lain, *current bet* dan taruhan setiap pemain sebesar nol.
4. Pada *preflop*, giliran pertama dimulai pada pemain di sebelah kiri pemain yang bertaruh sebesar *big blinds*. Pada *street* lain, giliran pertama dimulai pada pemain di sebelah kiri *dealer*.
5. Pada gilirannya, pemain dapat melakukan *fold*, *call*, *bet*, *raise*, dan *check*. Pemain hanya dapat melakukan *check* jika pemain telah bertaruh sebesar *current bet*. Pemain hanya dapat melakukan *bet* jika *current bet* sebesar 0. Pemain hanya dapat melakukan *call* dan *raise* jika *current bet* lebih besar dari 0.
6. Jika pemain tidak memiliki jumlah kepingan yang cukup untuk *call* sehingga taruhan pemain setara *current bet*, maka pemain dapat melakukan *all-in*. Pemain juga dapat melakukan *all-in* saat melakukan *call*, *bet*, dan *raise*. Pemain yang telah melakukan *all-in* tidak dapat melakukan aksi lagi selama permainan berlangsung.
7. Giliran selanjutnya diberikan ke pemain selanjutnya searah jarum jam. Jika pemain telah *fold* maupun *all-in*, maka pemain tidak melakukan apa-apa dan giliran diberikan ke pemain selanjutnya.
8. Jika semua pemain yang tidak *fold* maupun *all-in* telah bertaruh sejumlah keping yang sama dan telah melakukan aksi, maka *dealing* terjadi.

9. Jika pada suatu *street* hanya tersisa satu pemain yang tidak *fold*, maka pemain itu menjadi pemenang dari permainan ini. Jika semua pemain yang tidak *fold* telah melakukan *all-in* atau hanya terdapat satu pemain yang tidak *fold* yang tidak melakukan *all-in*, maka permainan akan langsung diteruskan sampai akhir permainan.
10. Jika setelah *river* terdapat lebih dari satu pemain yang tidak *fold*, maka setiap pemain menunjukkan kartu yang dimilikinya dan pemain yang memiliki nilai terbesar dari kombinasi kartu yang dimilikinya dan lima kartu di meja akan menjadi pemenang dari permainan. Namun, permasalahan ini tidak mempedulikan kartu sama sekali maupun siapa yang menang.

2.1.4 Simulasi Permainan *Texas Holdem Poker*

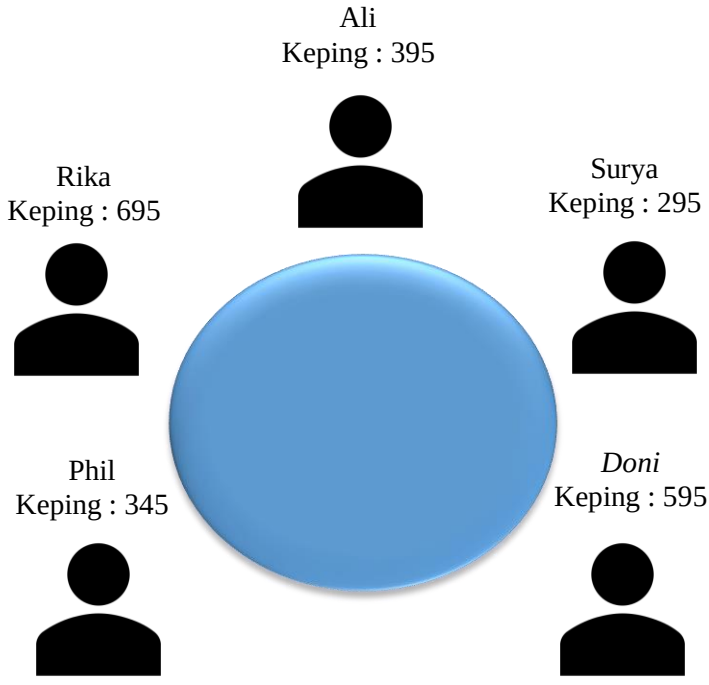
Berikut adalah salah satu contoh simulasi pada *Texas Holdem Poker*:

1. Terdapat 5 orang pemain yang akan bermain *Texas Holdem Poker*. Pemain ini bernama Ali, Surya, Doni, Phil, dan Rika. Doni akan menjadi *dealer* pada permainan ini. Sebelum permainan dimulai, masing-masing pemain telah memiliki kepingan keping. Ali, Surya, Doin, Phil, dan Rika masing-masing memiliki kepingan plastik sebanyak 400 buah, 300 buah, 600 buah, 350 buah, dan 700 buah. Mereka akan memulai permainan pada sebuah meja melingkar. Gambar 2.1 akan menunjukkan ilustrasi dari poin ini. Nama yang ditulis miring menunjukkan bahwa pemain itu adalah *dealer*.



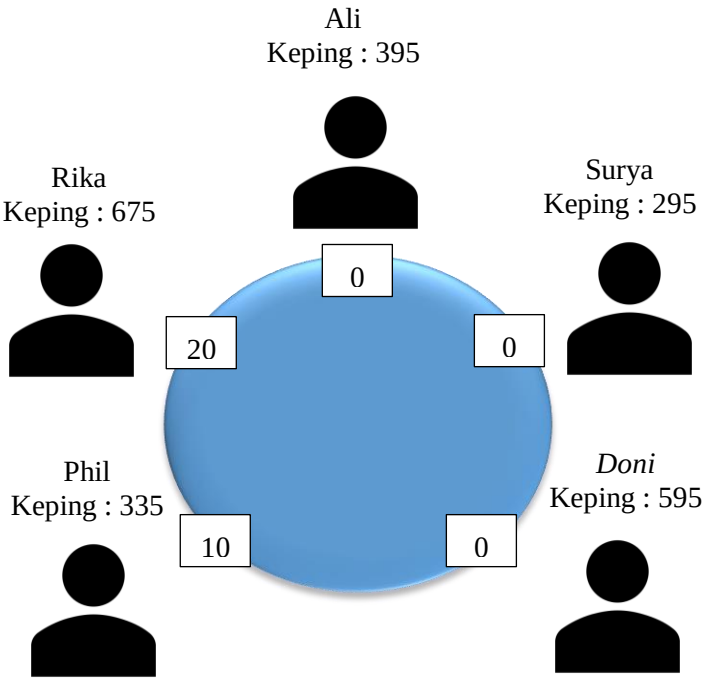
Gambar 2.1 Ilustrasi dari Poin 1 Simulasi *Texas Holdem Poker*

2. Sebelum permainan dimulai, ditentukan *ante*, *small blinds*, dan *big blinds*. *Ante* yang harus dibayarkan sebesar 5 keping, *small blinds* sebesar 10 keping, dan *big blinds* sebesar 20 keping. Permainan dimulai dengan setiap pemain membayarkan *ante*. Gambar 2.2 akan menunjukkan ilustrasi dari poin ini.



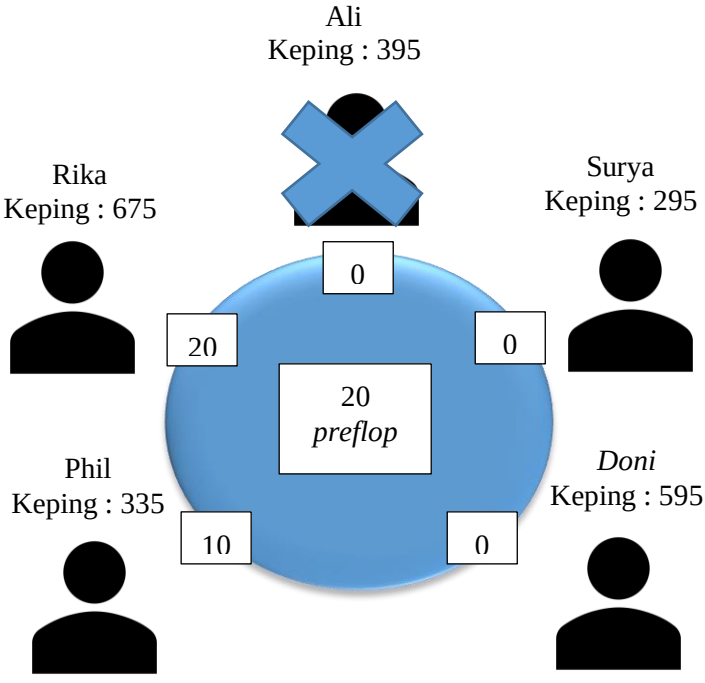
Gambar 2.2 Ilustrasi dari Poin 2 Simulasi *Texas Holdem Poker*

3. Phil akan bertaruh sebesar *small blinds*, yaitu 10 keping, dan Rika akan bertaruh sebesar *big blinds*, yaitu 20 keping. Gambar 2.3 akan menunjukkan ilustrasi dari poin ini. Angka pada sisi lingkaran menunjukkan jumlah taruhan dari pemain yang berada di dekat angka tersebut.



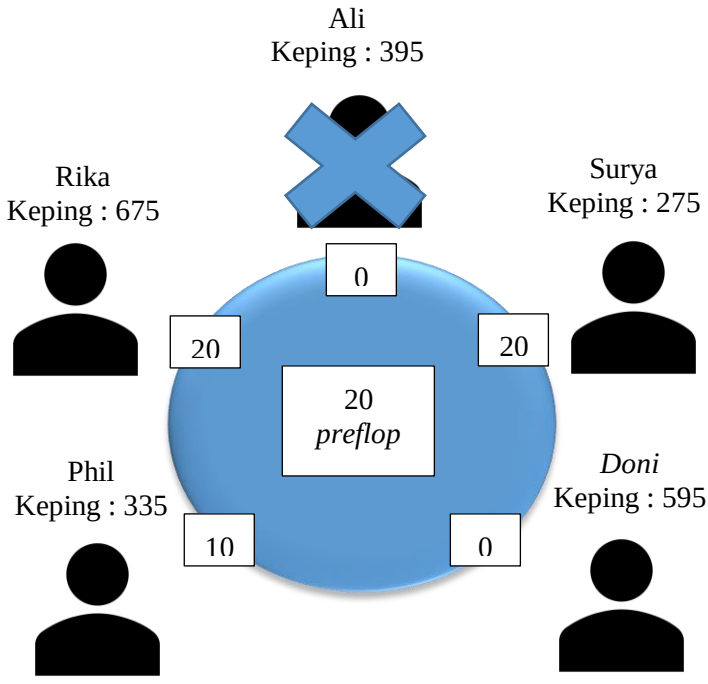
Gambar 2.3 Ilustrasi dari Poin 3 Simulasi *Texas Holdem Poker*

4. Giliran permainan dimulai dari pemain sebelah Rika, yaitu Ali. *Street* pada saat ini adalah *preflop* dan *current bet* sebesar 20. Ali dapat melakukan *fold* , *call*, dan *raise*. Pada simulasi ini, Ali akan melakukan *fold* sehingga Ali menyerah dari permainan ini. Gambar 2.4 menunjukkan ilustrasi dari poin ini. Ikon Ali akan dikasih tanda silang yang menunjukkan bahwa pemain menyerah dari permainan ini. Angka dan tulisan di tengah lingkaran menunjukkan *current bet* dan *street* saat ini.



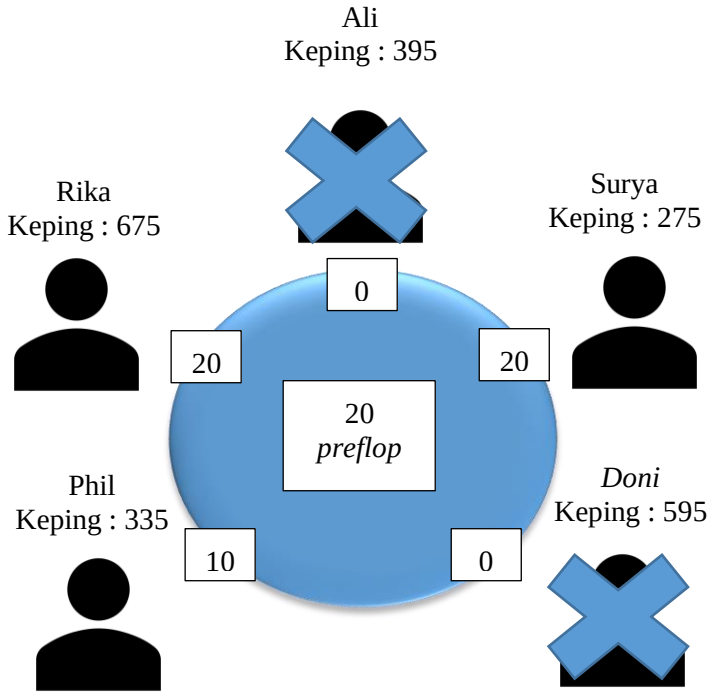
Gambar 2.4 Ilustrasi dari Poin 4 Simulasi *Texas Holdem Poker*

5. Giliran selanjutnya diberikan ke Surya. Surya dapat melakukan *fold*, *call*, dan *raise*. Pada simulasi ini, Surya akan melakukan *call* sehingga jumlah taruhannya sebesar *current bet*, yaitu 20. Gambar 2.5 menunjukkan ilustrasi dari poin ini.



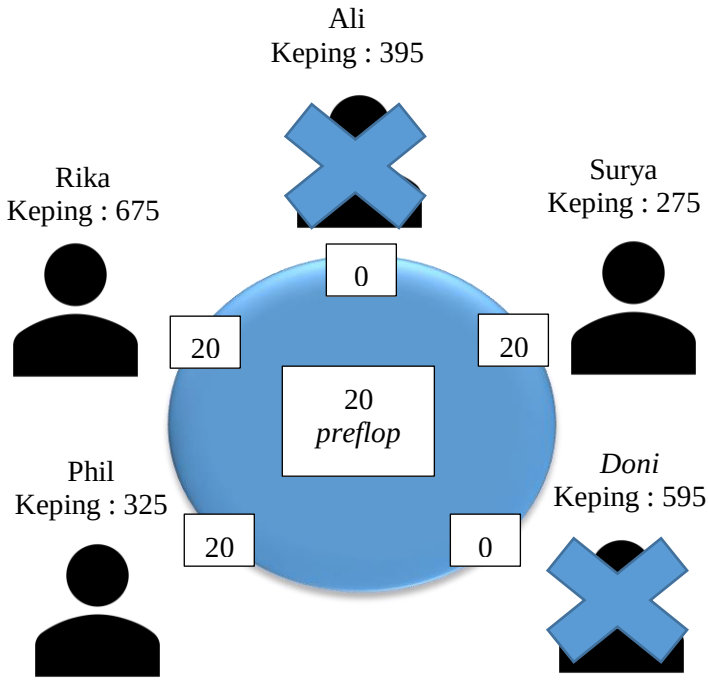
Gambar 2.5 Ilustrasi dari Poin 5 Simulasi *Texas Holdem Poker*

6. Giliran selanjutnya diberikan ke Doni. Doni dapat melakukan *fold*, *call*, dan *raise*. Pada simulasi ini, Doni akan melakukan *fold* sehingga Doni menyerah dari permainan. Gambar 2.6 menunjukkan ilustrasi dari poin ini.



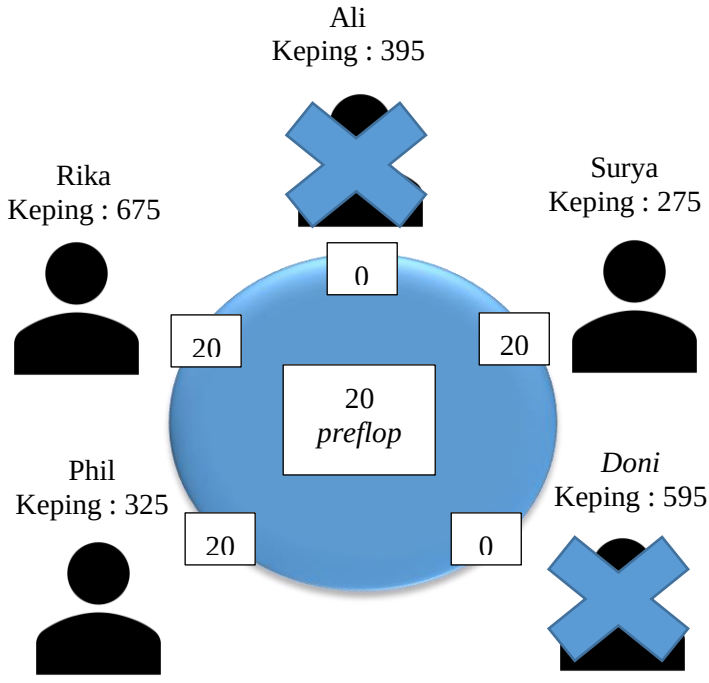
Gambar 2.6 Ilustrasi dari Poin 6 Simulasi *Texas Holdem Poker*

7. Giliran selanjutnya diberikan ke Phil. Phil dapat melakukan *fold*, *call*, dan *raise*. Pada simulasi ini, Phil akan melakukan *call* sehingga jumlah taruhannya sebesar 20. Gambar 2.7 menunjukkan ilustrasi dari poin ini.



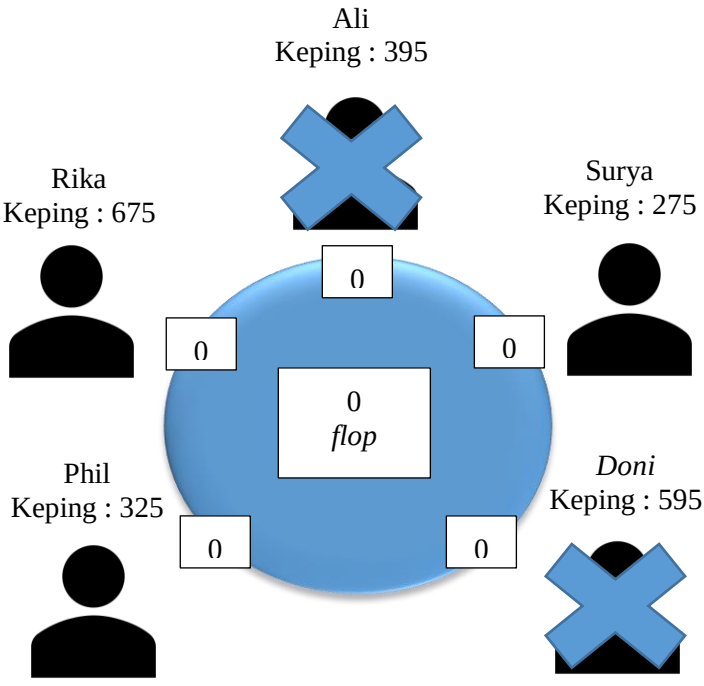
Gambar 2.7 Ilustrasi dari Poin 7 Simulasi *Texas Holdem Poker*

8. Giliran selanjutnya diberikan ke Rika. Rika dapat melakukan *check*, *fold*, dan *raise*. Pada simulasi ini, Rika akan melakukan *check* sehingga Rika melewatkan gilirannya. Gambar 2.8 menunjukkan ilustrasi dari poin ini.



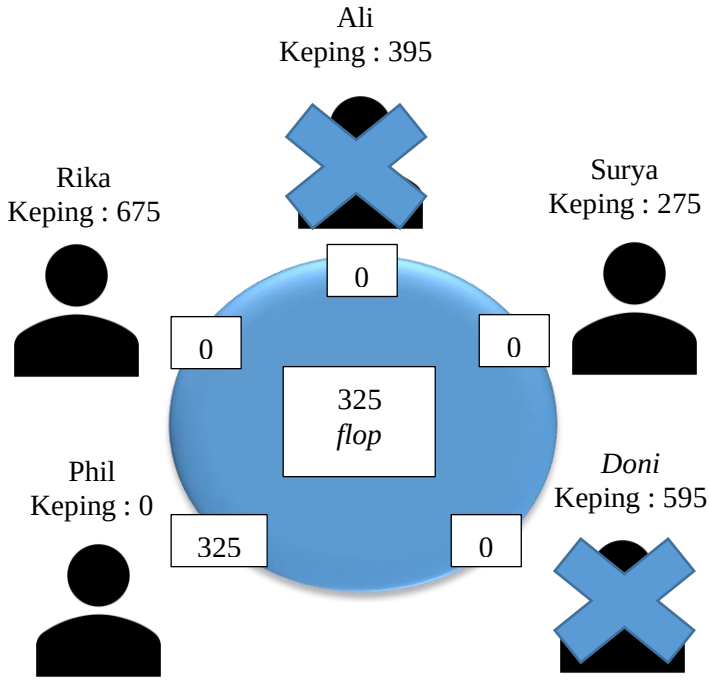
Gambar 2.8 Ilustrasi dari Poin 8 Simulasi *Texas Holdem Poker*

9. Karena setiap pemain yang tidak *fold* maupun *all-in* telah melakukan aksi dan bertaruh sebesar *current bet*, proses *dealing*, yaitu pergantian *street* akan dilakukan. *Street* sekarang menjadi *flop* dan *current bet* sebesar nol. Gambar 2.9 menunjukkan ilustrasi dari poin ini.



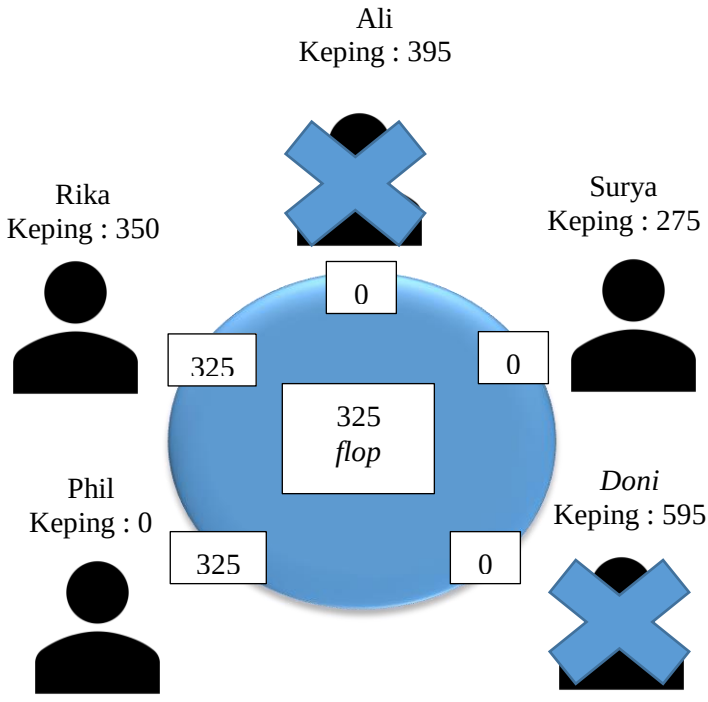
Gambar 2.9 Ilustrasi dari Poin 9 Simulasi *Texas Holdem Poker*

10. Giliran pertama pada *flop* diberikan ke pemain di sebelah kiri Doni, yaitu Phil. Phil dapat melakukan *check*, *fold*, dan *bet*. Pada simulasi ini, Phil akan melakukan *bet* secara *all-in*, yaitu sebesar 325 keping, sehingga taruhan Phil sebesar 325 dan *current bet* sebesar 325. Gambar 2.10 menunjukkan ilustrasi dari poin ini.



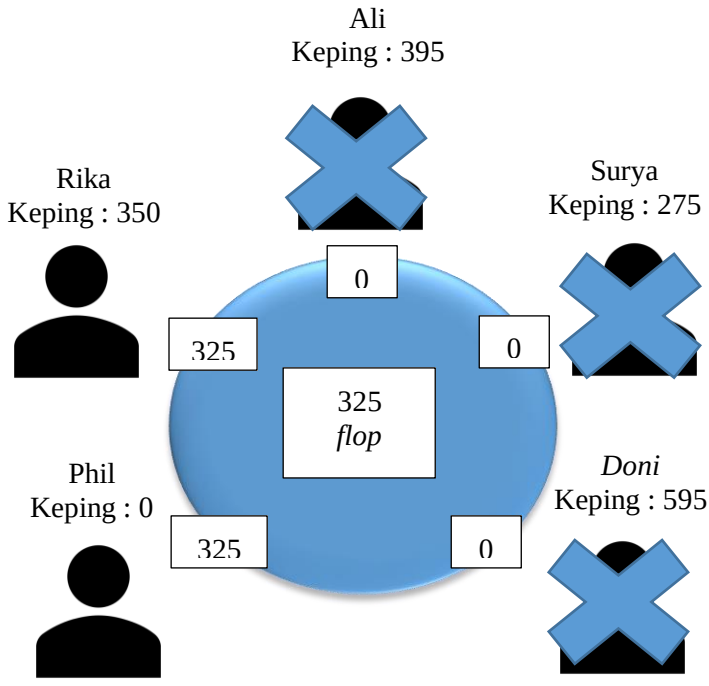
Gambar 2.10 Ilustrasi dari Poin 10 Simulasi *Texas Holdem Poker*

11. Giliran selanjutnya diberikan ke Rika. Rika dapat melakukan *fold*, *call*, dan *raise*. Pada simulasi ini, Rika akan melakukan *call* , yaitu sebesar 325 keping, sehingga taruhan Rika sebesar 325. Gambar 2.11 menunjukkan ilustrasi dari poin ini.



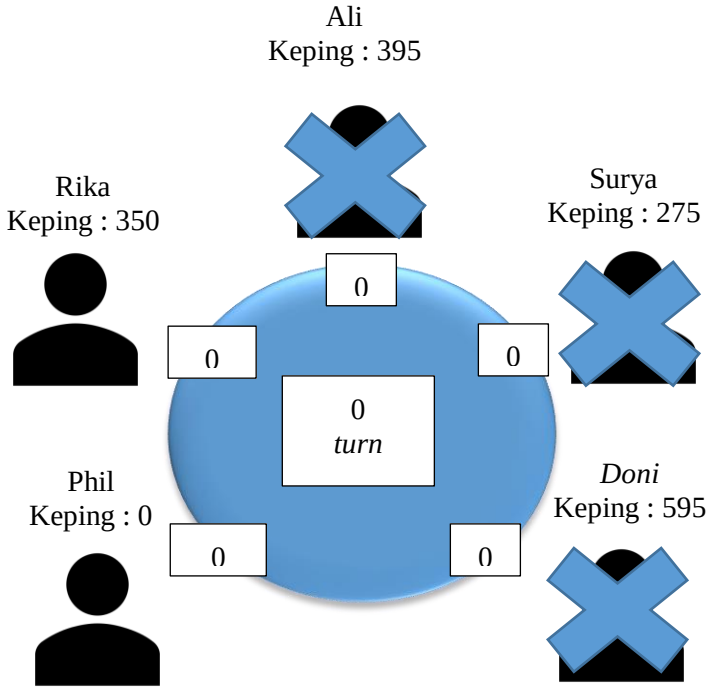
Gambar 2.11 Ilustrasi dari Poin 11 Simulasi *Texas Holdem Poker*

12. Giliran selanjutnya diberikan ke Surya. Surya dapat melakukan *fold* dan *call* secara *all-in*. Pada simulasi ini, Surya akan melakukan *fold* sehingga Surya menyerah dari permainan. Gambar 2.12 menunjukkan ilustrasi dari poin ini.



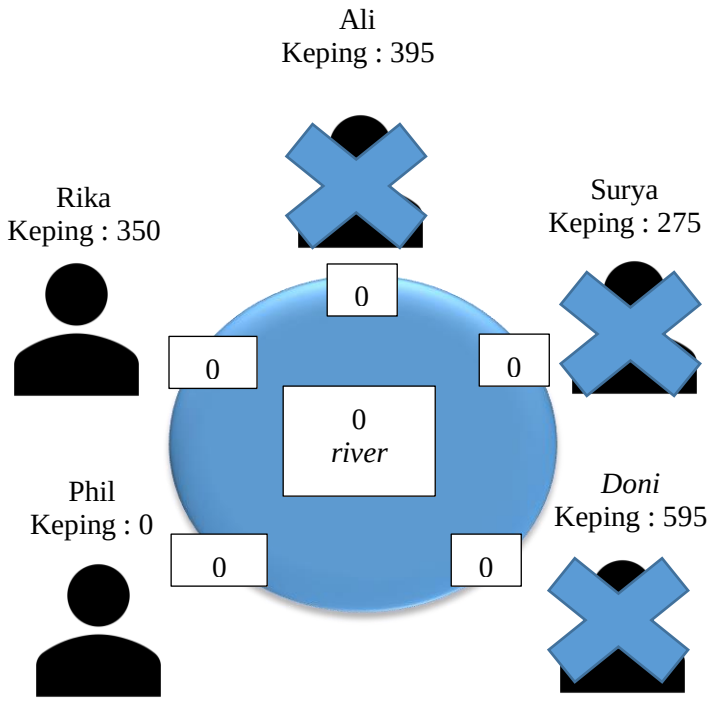
Gambar 2.12 Ilustrasi dari Poin 12 Simulasi *Texas Holdem Poker*

13. Karena setiap pemain yang tidak *fold* maupun *all-in* telah melakukan aksi dan bertaruh sebesar *current bet*, proses *dealing*, yaitu pergantian *street* akan dilakukan. *Street* sekarang menjadi *turn* dan *current bet* sebesar nol. Gambar 2.13 menunjukkan ilustrasi dari poin ini



Gambar 2.13 Ilustrasi dari Poin 13 Simulasi *Texas Holdem Poker*

14. Giliran pertama pada *turn* diberikan ke pemain di sebelah kiri Doni, yaitu Phil. Namun, karena Phil telah *all-in*, giliran diberikan ke pemain selanjutnya, yaitu Rika. Karena hanya terdapat satu pemain yang belum *fold* yang tidak *all-in*, permainan akan diteruskan langsung sampai akhir permainan. Pemenang dari permainan ini tidak dipermasalahkan. Gambar 2.14 menunjukkan kondisi akhir dari permainan ini.



Gambar 2.14 Kondisi akhir dari simulasi *Texas Holdem Poker*

2.2 Deskripsi Permasalahan

Permasalahan yang diberikan berupa *bot* dalam permainan *Texas Holdem Poker* yang memiliki modul untuk melakukan *tracking* kejadian-kejadian yang terjadi dalam permainan. Modul dipanggil ketika *bot* akan melakukan gilirannya pada permainan. Saat modul dijalankan, *bot* memiliki informasi mengenai *ante*, *small blinds*, dan *big blinds*. *Bot* juga mengetahui jumlah pemain, siapa yang menjadi *dealer*, serta nama dan jumlah keping awal setiap pemain. *Bot* juga mengetahui jumlah keping setiap pemain saat ini, namun *bot* tidak memiliki informasi mengenai *current bet* maupun taruhan tiap pemain. Jika modul pertama kali dijalankan, maka *bot* akan *tracking* segala aksi yang terjadi dengan data yang dipunya. Jika tidak, maka *bot* juga memiliki informasi mengenai segala aksi yang terjadi sebelum giliran *bot* sebelumnya dan *bot* perlu *tracking* segala aksi yang terjadi dari giliran *bot* sebelumnya sampai giliran *bot* saat ini.

2.2.1 Parameter Input

Parameter input pada permasalahan Timus Online Judge 2082 – Poker adalah seperti di bawah ini.

1. Baris pertama terdiri dari tiga buah *integer*, yaitu *small blinds sb* ($10 \leq sb \leq 500$), *big blinds bb* ($20 \leq bb \leq 1000$, $sb < bb$), dan *ante a* ($0 \leq a \leq 100$).
2. Baris kedua terdiri dari tiga buah *integer*, yaitu jumlah pemain n ($2 \leq n \leq 9$), posisi *dealer d* ($1 \leq d \leq n$), dan posisi *bot h* ($1 \leq h \leq n$).
3. Baris ketiga terdiri dari sebuah *string*, yaitu *street* saat modul dipanggil.
4. Selanjutnya terdapat baris sejumlah n yang berisi deskripsi setiap pemain secara urut. Setiap baris terdapat nama pemain berupa *string* yang terdiri dari huruf besar dan kecil dan angka bahasa Inggris dengan panjang tidak

lebih dari 20 dan kumpulan keping awal (*integer* dari 1 sampai dengan 20000). Setiap nama pasti berbeda dan tidak ada nama *dealing* pada *input*.

5. Baris selanjutnya terdiri dari sebuah *integer* k ($0 \leq k \leq 1000$), yaitu jumlah aksi yang terjadi.
6. Baris k selanjutnya terdiri dari aksi. Aksi terdiri dari *dealing* maupun aksi pemain. Berikut format dari aksi-aksi tersebut:
    - dealing <street>, pergantian *street*.
    - <player> fold, pemain *fold*
    - <player> checks, pemain *check*
    - <player> calls <amount>, *amount* $>$ 0, pemain *call* sebesar *amount*.
    - <player> bets <amount>, *amount* $>$ 0, pemain *bet* sebesar *amount*.
    - <player> raises <amount> to <total>, *amount* $>$ 0, *total* $>$ *amount*, pemain *raise* sebesar *amount* sehingga *current bet* sebesar *total*

    player adalah nama pemain dan *street* adalah nama *street* yang dituju. Jika pemain *all-in*, maka tindakan ini akan direpresentasikan sebagai *call*, *bet*, dan *raise*. Dipastikan seluruh aksi yang dilakukan benar.
7. Baris terakhir terdiri dari *n integers*, yaitu jumlah keping setiap pemain saat ini secara urut.

Gambar 2.15 adalah contoh standar *input* dari permasalahan ini.

input

```
10 20 2
5 3 4
preflop
PhilIvey 500
TomDwan 500
ViktorBlom 500
Grobot 500
Ziigmund 500
0
498 478 438 488 478
```

Gambar 2.15 Contoh Standar *Input* Program

2.2.2 *Output* Permasalahan

Output program terdiri dari:

1. Baris pertama terdiri dari sebuah integer *m*, yaitu jumlah aksi yang terjadi.
2. Baris *m* selanjutnya adalah aksi-aksi yang terjadi secara urut dengan format yang sama dengan *input*.

Dipastikan terdapat satu solusi dan solusi ini unik. Gambar 2.16 adalah *output* dari gambar 2.15.

output

```
3
PhilIvey folds
TomDwan calls 20
ViktorBlom raises 40 to 60
```

Gambar 2.16 *Output* dari Gambar 2.15

2.3 Deskripsi Umum Teori

Pada subbab ini, akan dijelaskan berbagai landasan teori secara umum yang digunakan untuk melakukan pendekatan terhadap penyelesaian permasalahan.

2.3.1 Greedy

Greedy adalah algoritma yang membentuk solusi dengan mencari nilai maksimum sementara pada setiap langkahnya. Nilai maksimum sementara ini dikenal dengan istilah *local maximum*.

[*Halaman ini sengaja dikosongkan*]

BAB III
DESAIN DAN ANALISIS

Bab ini menjelaskan mengenai desain dan analisis algoritma yang digunakan dalam strategi penyelesaikan permasalahan 2082 – Poker pada Timus Online Judge.

3.1 Analisis Permasalahan

Inti dari permasalahan ini adalah *tracking* segala aksi yang sudah terjadi saat modul dipanggil (saat giliran *bot*). Modul dapat dijalankan meskipun *bot* sudah *folds* atau *all-in*. Jika modul pertama kali dijalankan, maka aksi-aski pada *output* adalah aksi dari awal permainan sampai aksi sebelum giliran *bot*. Jika tidak, maka tindakan yang dikeluarkan pada *output* adalah aksi-aksi dari giliran *bot* sebelumnya sampai giliran sebelum *bot* saat module dipanggil. Oleh karena itu, *output* pasti hanya mengandung aksi oleh *bot* sebanyak maksimal satu. Strategi yang digunakan untuk menyelesaikan permasalahan ini dibagi menjadi dua, yaitu pemrosesan *input* dan pemrosesan *output*.

3.1.1 Pemrosesan *Input*

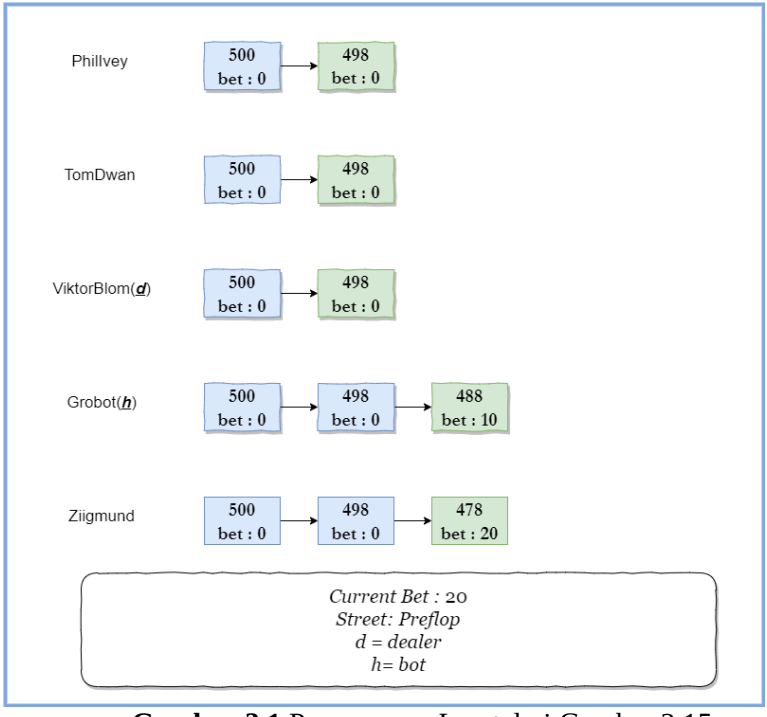
Bagian ini akan menjelaskan mengenai pemrosesan *input* setelah semua *input* dimasukkan sehingga didapatkan kondisi saat *bot* memangil modul. Hal-hal yang diproses adalah pengurangan jumlah keping dan penerapan aksi yang diketahui *bot*. Tahapan pemrosesan *input* adalah sebagai berikut:

1. Jumlah keping setiap pemain dikurangi sejumlah *ante*.
2. Sesuai aturan permainan, jumlah keping pemain yang duduk di sebelah *dealer* dikurangi sebesar *small blind*. Jika jumlah pemain hanya dua, maka jumlah keping yang dikurangi sebesar *small blinds* adalah jumlah keping *dealer*. Jumlah keping pemain yang duduk di sebelah pemain yang mempertaruhkan *small blinds* dikurangi *big blinds*. Jika pemain tidak memiliki jumlah keping yang cukup saat membayar *small blinds* maupun *big blinds*,

maka pemain tersebut membayar semua keping yang dimilikinya.

3. Tindakan yang diketahui *bot* akan diproses sehingga didapat kondisi jumlah keping setiap pemain setelah segala kejadian yang diketahui *bot*.

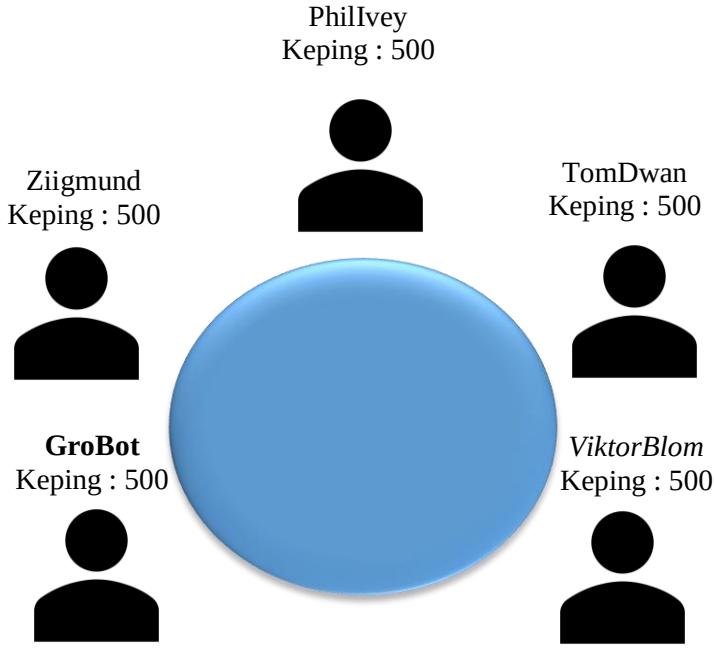
Kondisi yang didapatkan ini akan digunakan selanjutnya untuk pemrosesan *output*. Gambar 3.1 menunjukan pemrosesan umum *input* dari Gambar 2.15.



Gambar 3.1 Pemrosesan *Input* dari Gambar 2.15

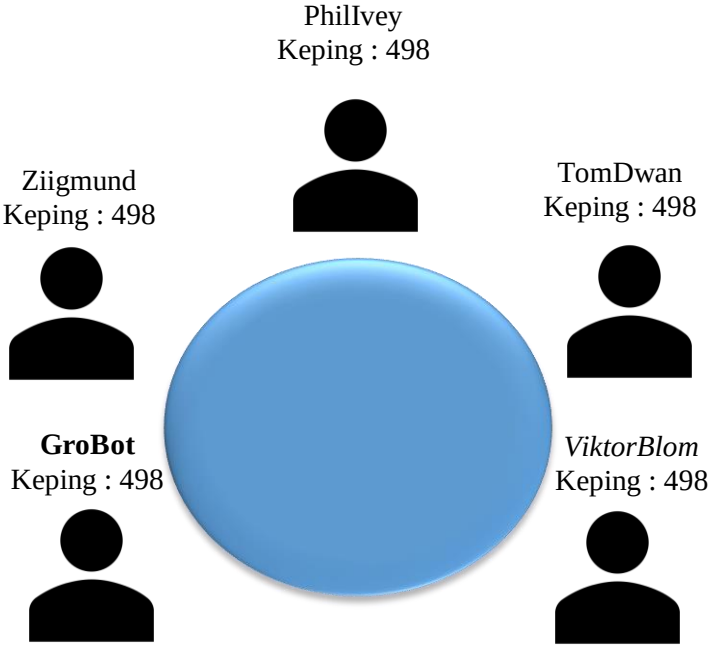
Berikut adalah simulasi dari pemrosesan *Input*:

1. Terdapat 5 orang pemain. Pemain ini bernama PhilIvey, TomDwan, ViktorBlom, GroBot, dan Ziigmund. Viktorblom adalah *dealer* dan Grobot adalah *bot*. ViktorBlom akan menjadi *dealer* pada permainan ini. Masing-masing pemain telah memiliki jumlah keping sebanyak 500. Mereka akan memulai permainan pada sebuah meja melingkar. Gambar 3.2 akan menunjukkan ilustrasi dari poin ini. Nama yang ditulis miring menunjukkan bahwa pemain itu adalah *dealer* dan yang ditulis tebal menunjukkan bahwa pemain itu adalah *bot*.



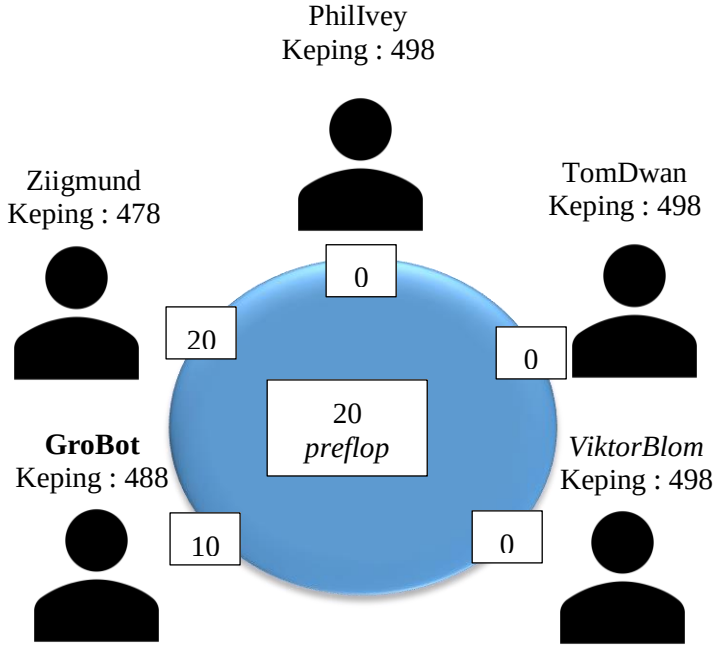
Gambar 3.2 Ilustrasi Pertama Simulasi dari Pemrosesan *Input*

2. Setiap pemain akan bertaruh sebesar *ante*. Gambar 3.3 akan menunjukkan ilustrasi dari poin ini.



Gambar 3.3 Ilustrasi Kedua Simulasi dari Pemrosesan *Input*

3. Grobot akan bertaruh sebesar *small blinds*, yaitu 10, dan Ziigmund akan bertaruh sebesar *big blinds*, yaitu 20. *Street* adalah *preflop* dan *current bet* adalah 20. Gambar 3.4 akan menunjukkan ilustrasi dari poin ini.



Gambar 3.4 Ilustrasi Ketiga Simulasi dari Pemrosesan *Input*

3.1.2 Pemrosesan *Output*

Bagian ini akan menjelaskan mengenai pemrosesan *output*. Pada pemrosesan ini, pengecekan dilakukan pada giliran tiap pemain dan setelah pemain melakukan aksi. Pengecekan dilakukan pada giliran tiap pemain untuk menentukan apakah pemain dapat bertaruh secara *greedy* atau tidak. Terdapat beberapa kondisi yang akan menentukan hal ini. Terdapat juga pengecekan untuk menentukan apakah akan terjadi *dealing*

(pergantian *street*) dan untuk menentukan apakah modul yang dilakukan *bot* telah berakhir atau belum. Berikut kondisi- kondisi yang mungkin terjadi pada saat pengecekan:

1. Jika *street* saat ini berbeda dengan *street* akhir yang diketahui *bot* dan semua pemain yang tidak *folds* maupun tidak *all-in* sudah bertaruh sebesar *current bet* dan sudah melakukan aksi pada *street* ini, maka akan terjadi *dealing*. Setelah *dealing*, giliran pemain selanjutnya adalah pemain di sebelah *dealer*.
2. Jika sekarang giliran *bot*, dan jumlah keping setiap pemain saat ini sama dengan jumlah keping akhir setiap pemain, serta *street* saat ini sama dengan *street* akhir, maka pemrosesan *output* selesai.
3. Jika pemain berada dalam kondisi *folds* maupun telah melakukan *all-in*, maka pemain tidak melakukan aksi sama sekali.
4. Jika *current bet* sebesar nol:
   a) Jika *street* saat ini sama dengan *street* yang diketahui *bot*
      i) Jika jumlah keping pemain saat ini dan akhir sama, pemain akan melakukan *check*.
      ii) Selain itu, pemain akan melakukan *bet* sebesar selisih jumlah keping pemain saat ini dan akhir.
   b) Jika *street* saat ini berbeda dengan *street* yang diketahui *bot*
      i) Jika setelah ganti *street*, pemain melakukan aksi sebelum *bot*, maka pemain akan melakukan *check*.
      ii) Selain itu, hasil dan kondisi sama dengan poin 4.a.
5. Jika *bet* pemain sebesar *current bet* dan *current bet* lebih besar dari nol:
   a) Jika *street* saat ini sama dengan *street* yang diketahui *bot*
      i) Jika jumlah keping pemain saat ini dan akhir sama, pemain akan melakukan *check*.

ii) Selain itu, pemain akan melakukan *raise* sebesar selisih jumlah keping pemain saat ini dan akhir – (*current bet* - *bet* pemain) menjadi selisih jumlah keping pemain saat ini dan akhir + *bet* pemain.

b) Jika *street* saat ini berbeda dengan *street* akhir yang diketahui *bot*

i) Jika setelah ganti *street*, pemain melakukan aksi sebelum *bot*, maka pemain akan melakukan *check*.

ii) Selain itu, hasil dan kondisi sama dengan poin 5.a.

6. Jika *bet* pemain < *current bet*:

a) Jika *street* saat ini sama dengan *street* akhir yang diketahui *bot*

i) Jika jumlah keping pemain saat ini dan akhir sama, pemain akan melakukan *fold*.

ii) Jika selisih jumlah keping pemain saat ini dan akhir > 0 dan ≤ (*current bet* - *bet* pemain), pemain akan melakukan *call* sebesar selisih jumlah keping pemain saat ini dan akhir.

iii) Jika selisih jumlah keping pemain saat ini dan akhir > 0 & selisih jumlah keping pemain saat ini dan akhir > (*current bet* - *bet* pemain), pemain akan melakukan *raise* sebesar selisih jumlah keping pemain saat ini dan akhir – (*current bet* - *bet* pemain) menjadi selisih jumlah keping pemain saat ini dan akhir + *bet* pemain.

b) Jika *street* saat ini berbeda dengan *street* akhir yang diketahui *bot*

i) Jika setelah ganti *street*, giliran pemain sebelum giliran *bot*

(1) Jika jumlah keping pemain saat ini dan akhir sama, pemain akan melakukan *fold*.

(2) Jika selisih jumlah keping pemain saat ini dan akhir > 0 dan ≤ (*current bet* - *bet* pemain), pemain akan melakukan *call* sebesar selisih jumlah keping pemain saat ini dan akhir

(3) Jika selisih jumlah keping pemain saat ini dan akhir > 0 dan > (*current bet* - *bet* pemain), pemain akan melakukan *call* sebesar (*current bet* - *bet* pemain).

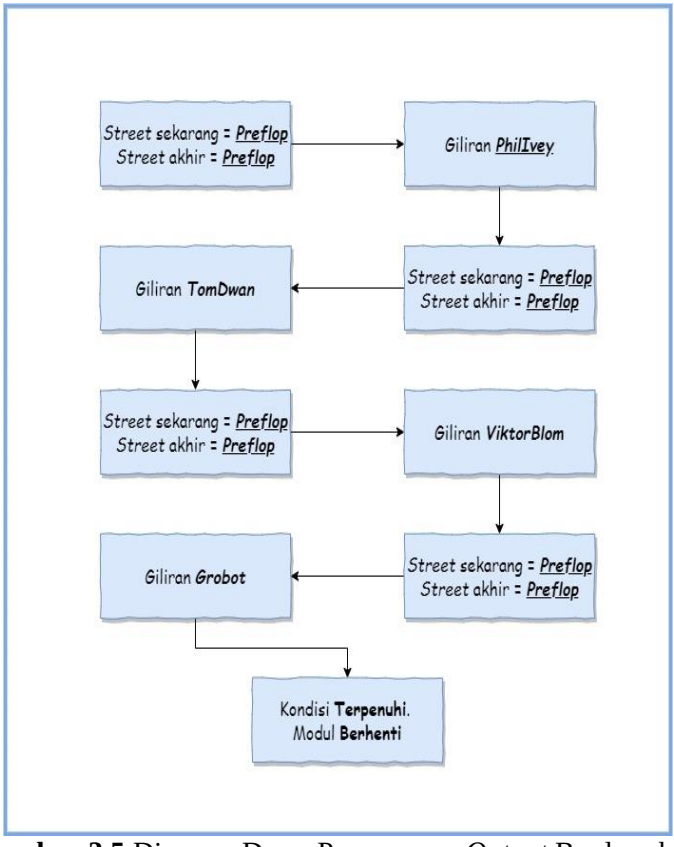
ii) Selain itu,hasil dan kondisi sama dengan poin 6.a

Jika jumlah keping pemain setelah melakukan *bet*, *raise*, atau *call* sebesar nol, maka pemain melakukan *all-in*.

Saat melakukan *call*, *bet*, dan *raise*, pemain melakukannya secara *greedy*. Namun, ada kondisi di mana pemain tidak dapat melakukan aksi secara *greedy* pada gilirannya. Hal ini disebabkan karena di dalam modul ternyata pemain itu dapat melakukan aksi sebanyak dua kali.

Modul akan berhenti jika pada pengecekan pada giliran *bot*, kondisi saat ini sama dengan kondisi akhir yang diketahui *bot*.

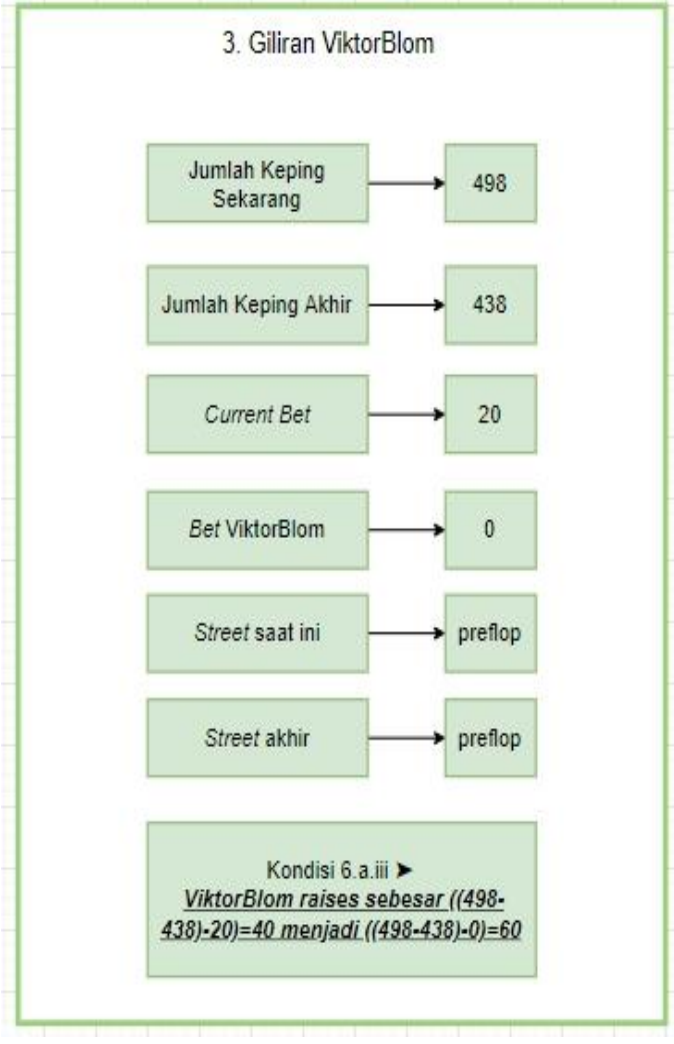
Gambar 3.5, Gambar 3.6, Gambar 3.7, Gambar 3.8, dan Gambar 3.9 merupakan penjelasan umum pemrosesan *output* setelah Gambar 3.1.



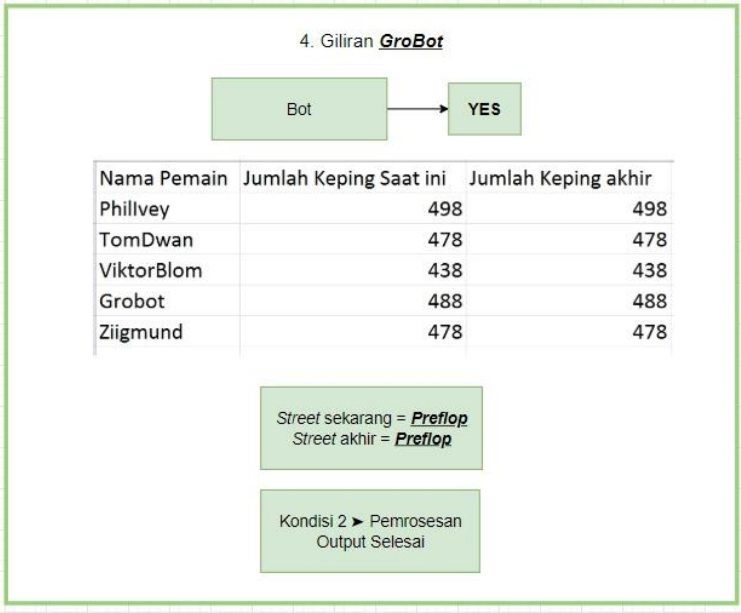
Gambar 3.5 Diagram Dasar Pemrosesan *Output* Berdasarkan Hasil dari Gambar 3.1

Gambar 3.6 Penjelasan Giliran *PhilIvey*

Gambar 3.7 Penjelasan Giliran *TomDwan*



Gambar 3.8 Penjelasan Giliran *ViktorBlom*



Gambar 3.9 Penjelasan Giliran *GroBot*

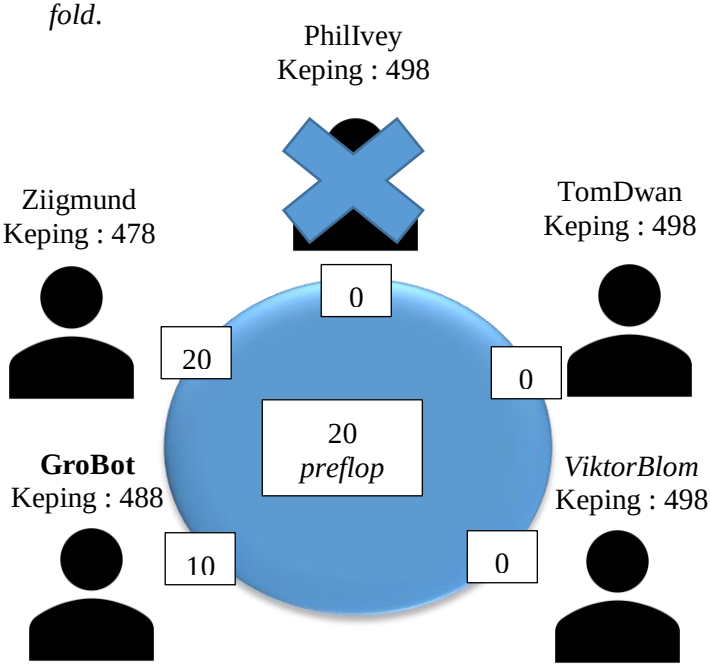
Hasil aksi dari pemrosesan *output* di atas adalah:

- *PhilIvey folds*
- *TomDwan calls* 20
- *ViktorBlom raises* 40 *to* 60

Jumlah aksi pada *output* sebanyak tiga. *Output* ini sama dengan *output* pada Gambar 2.16.

Berikut adalah simulasi untuk pemrosesan *output*:

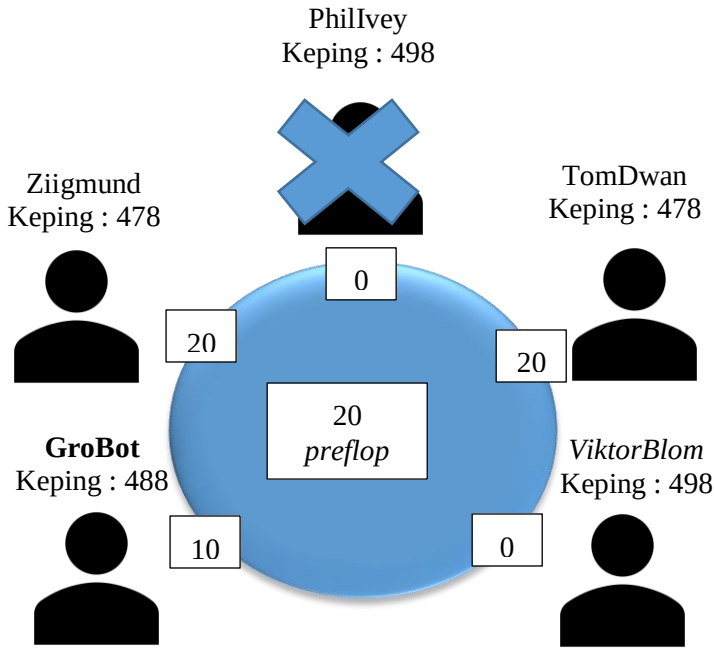
1. Dilakukan pengecekan *street*. Tidak dilakukan pergantian *street* karena *street* saat ini sama dengan *street* akhir.
2. Giliran pertama akan diberikan ke PhilIvey. Jumlah keping PhilIvey saat ini dan akhir sama. *Bet* PhilIvey saat ini sebesar 0 dan *current bet* sebesar 20. Kesimpulannya adalah PhilIvey *fold* sehingga PhilIvey menyerah dari permainan. Gambar 3.10 merupakan ilustrasi dari poin ini. Tanda silang menunjukkan bahwa pemain itu telah *fold*.



Gambar 3.10 Ilustrasi Poin 2 Simulasi dari Pemrosesan *Output*

3. Dilakukan pengecekan *street*. Tidak dilakukan pergantian *street* karena *street* saat ini sama dengan *street* akhir.

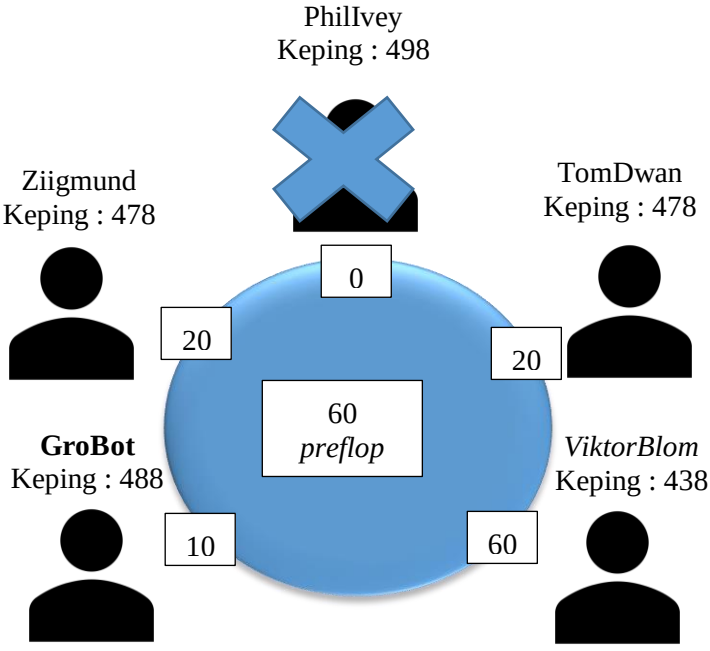
4. Giliran selanjutnya akan diberikan ke TomDwan. Jumlah keping TomDwan saat ini berbeda dengan akhir. *Bet* TomDwan saat ini sebesar 0 dan *current bet* sebesar 20. Selisih jumlah keping saat ini dan akhir sama besarnya dengan selisih *bet* TomDwan dan *current bet*. Kesimpulannya adalah TomDwan *calls* sebesar *current bet*, yaitu 20, sehingga *bet* TomDwan sebesar 20. Gambar 3.11 merupakan ilustrasi dari poin ini.



Gambar 3.11 Ilustrasi Poin 4 Simulasi dari Pemrosesan *Output*

5. Dilakukan pengecekan *street*. Tidak dilakukan pergantian *street* karena *street* saat ini sama dengan *street* akhir.

6. Giliran selanjutnya akan diberikan ke ViktorBlom. Jumlah keping ViktorBlom saat ini berbeda dengan akhir. *Bet* ViktorBlom saat ini sebesar 0 dan *current bet* sebesar 20. Selisih jumlah keping saat ini dan akhir lebih besar dari selisih *bet ViktorBlom* dan *current bet*. Selisih jumlah keping sebesar 60 dan selisih *bet ViktorBlom* dan *current bet* adalah 20. Kesimpulannya adalah ViktorBlom *raises* sebesar 60 – 20 = 40 sehingga *bet* ViktorBlom sebesar 60 dan *current bet* sebesar 20 + 40 = 60. Gambar 3.12 merupakan ilustrasi dari poin ini.



Gambar 3.12 Ilustrasi Poin 6 Simulasi dari Pemrosesan *Output*

7. Dilakukan pengecekan *street*. Tidak dilakukan pergantian *street* karena *street* saat ini sama dengan *street* akhir.
8. Giliran selanjutnya diberikan pada GroBot. Karena Grobot merupakan *bot*, pengecekan dilakukan pada kondisi keping saat ini dan akhir serta *street* saat ini dan akhir. Gambar 3.9 telah menunjukkan bahwa semua kondisi telah sama. Kesimpulannya adalah modul selesai.

3.2 Deskripsi Umum Sistem

Sistem yang dirancang berupa sebuah aplikasi berbasis konsol, menggunakan *input* standar sebagai *input* sistem. *Input* berupa beberapa baris *string* dan *integer* yang telah didefinisikan sebelumnya pada subbab 2.1. *Input* ini nantinya akan diproses sehingga didapatkan kondisi awal dari permasalahan yang akan diselesaikan. Solusi didapatkan dengan memproses kondisi tersebut sehingga didapatkan *output* yang sudah dijelaskan pada subbab 2.3 dengan menggunakan algoritma yang dijelaskan pada subbab 2.4.

3.3 Penjelasan Sistem

Sistem yang dirancang terdiri dari pemrosesan *input* dan pemrosesan *output*. Terdapat beberapa metode pada pemrosesan *input* dan pemrosesan *output* seperti yang sudah dijelaskan pada subbab 2.4.

Batasan penggunaan memori yang sangat terbatas, disertai dengan *constraint* waktu yang sangat singkat membuat pemilihan bahasa pemograman untuk pembuatan sistem ini jatuh pada bahasa pemograman C++.

3.4 Desain Global Variable

Subbab ini menjelaskan *variable-variable* yang digunakan pada sistem. *Variable* bersifat global untuk memudahkan penggunaan *variable* pada beberapa fungsi. Indeks pada *array* dimulai dari 0 sehingga pemain pertama memiliki indeks 0. Tabel 3.1A dan Tabel 3.1B menjelaskan *variable-variable* yang digunakan pada sistem.

Tabel 3.1A Tabel Penjelasan *Global Variable*

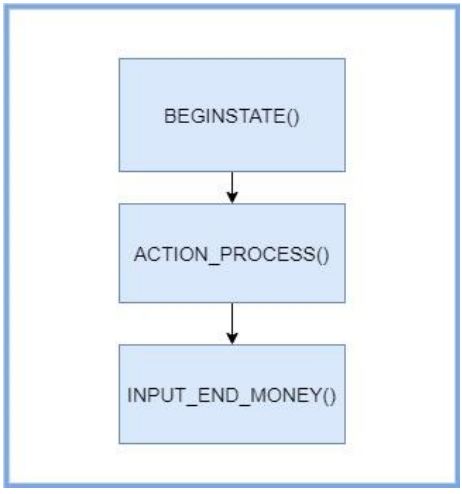
Nama	Tipe	Penjelasan
small_blinds	*integer*	*small blind*
big_blinds	*integer*	*big blind*
ante	*integer*	*ante*
number_player	*integer*	jumlah pemain
dealer_position	*integer*	posisi *dealer*
bot_position	*integer*	posisi *bot*
module_street	*string*	*street* dalam *module*
current_bet	*integer*	*current bet* pada *street* ini
player_name	*array of string*	nama pemain
current_money	*array of integer*	jumlah keping awal pemain
status	*array of integer*	status pemain
player_current_bet	*array of integer*	*current bet* pemain
action	*array of integer*	*flag* untuk menentukan pemain sudah melakukan aksi atau belum pada *street* ini
sb	*integer*	indeks pemain yang bertaruh *small blind*
bb	*integer*	indeks pemain yang bertaruh *big blind*
player_now	*integer*	indeks pemain yang akan dicek dalam *module*
number_action	*integer*	jumlah aksi yang diketahui *bot*
check_string	*string*	*variable* pembantu untuk menentukan bahwa aksi merupakan *dealing* atau tidak

Tabel 3.1B Tabel Penjelasan *Global Variable*

player_index	*integer*	indeks pemain yang melakukan aksi
player_action	*string*	aksi pemain yang dilakukan oleh pemain dengan indeks = player_index
bet_amount	*integer*	nilai keping yang dipertaruhkan pada saat pemain melakukan *bet*
call_amount	*integer*	nilai keping yang dipertaruhkan pada saat pemain melakukan *call*
raise_amount	*integer*	nilai keping yang dipertaruhkan pada saat pemain melakukan *raise*
raise_to	*integer*	nilai hasil *raise* sehingga *current bet* sebesar nilai ini
end_money	*array of integer*	jumlah keping akhir pemain saat *module* dijalankan
result	*integer*	jumlah aksi yang terjadi dalam *module*
module_flag	*integer*	*flag* untuk menentukan *module* sudah selesai atau belum
change_street_flag	*integer*	*flag* untuk menentukan apakah saatnya terjadi *dealing* atau belum
text_output	*array of string*	kumpulan string yang merupakan aksi-aksi yang terjadi dalam *module*
bet_difference	*integer*	selisih antara *current bet* dengan player_current_bet [player_now]
money_difference	*integer*	selisih antara current_money[player_now] dengan end_money[player_now]
string_number_1	*string*	*variable string* pembantu
string_number_2	*string*	*variable string* pembantu
position_flag	*integer*	*flag* untuk menentukan posisi pemain setelah *dealing*

3.5 Desain Pemrosesan *Input*

Pemrosesan *input* dilakukan pada tahap pertama dari sistem. Pemrosesan *input* dilakukan untuk mendapatkan kondisi yang selanjutnya akan diproses untuk mendapatkan *output*. Terdapat beberapa fungsi yang digunakan pada pemrosesan *input*. Gambar 3.13 merupakan *flowchart* dari pemrosesan *input*. Terdapat tiga fungsi yang digunakan yang akan dijelaskan selanjutnya.



Gambar 3.13 *Flowchart* Pemrosesan *Input*

3.5.1 Desain Fungsi BEGINSTATE

Fungsi ini digunakan untuk membaca beberapa *input* awal, memberi nilai pada beberapa *variable* serta menentukan pemain mana yang bertaruh *small blinds* maupun *big blinds*. *Input* pada *pseudocode* ini adalah:

- small_blinds, yaitu *variable* bertipe *integer* untuk menunjukkan nilai *small blind*

- big_blinds, yaitu *variable* bertipe *integer* untuk menunjukkan nilai *big blinds*
- ante, yaitu *variable* bertipe *integer* untuk menunjukkan nilai *ante*
- number_player, yaitu *variable* bertipe *integer* untuk menunjukkan jumlah pemain
- dealer_position, yaitu *variable* bertipe *integer* untuk menunjukkan posisi *dealer*, bot_position, yaitu *variable* bertipe *integer* untuk menunjukkan posisi dari *bot*
- current_money, yaitu *variable* bertipe *array of integer* untuk menunjukkan jumlah keping setiap pemain saat ini
- player_name, yaitu *variable* bertipe *array of string* untuk menunjukkan nama setiap pemain.

Tidak ada *output* pada *pseudocode* ini. *Pseudocode* dari fungsi ini dapat dilihat pada *pseudocode* 3.1A, *pseudocode* 3.1B, dan *pseudocode* 3.1.C.

Pseudocode 3.1A Fungsi BEGINSTATE(bagian 1)

```
INPUT small_blinds, big_blinds, ante
INPUT number_player, dealer_position,
bot_position
INPUT street_now
module_street ← "preflop"
current_bet ←big_blinds
FOR i = 0 to number_player:
    INPUT current_money[i], player_name[i]
```

Pseudocode 3.1B Fungsi BEGINSTATE(bagian 2)

```
    current_money[i] ←current_money[i] - ante
    status[i] ←1
    player_current_bet[i] ←0
    action[i] ←0
ENDFOR
IF number_player = 2:
    sb ← dealer_position - 1
    bb ← (sb+1) modulo jumlah_pemain

    player_now ← sb
ELSE:
    sb ← dealer_position
    bb ← (sb+1) modulo number_player
    player_now ← (dealer_position+2) modulo number_player
ENDIF
IF player_current_bet[sb] < small_blinds:
    player_current_bet[sb] ←current_money[sb]
    current_money[sb] ←0
    status[sb] ←0
ELSE:
    player_current_bet[sb] ←small_blinds
    current_money[sb] ← current_money[sb] - small_blinds
ENDIF
IF player_current_bet[bb] < big_blinds:
    player_current_bet[bb] ←current_money[bb]
    current_money[bb] ←0
```

Pseudocode 3.1C Fungsi BEGINSTATE(bagian 3)

```
    status[bb] ←0
ELSE:
    player_current_bet[bb] ←big_blinds
    current_money[bb] ←current_money[bb]-
big_blinds
ENDIF
```

3.5.2 Desain Fungsi ACTION_PROCESS

Fungsi ini digunakan untuk memproses *input* aksi yang diketahui oleh *bot* saat modul dijalankan. *Input* dari *pseudocode* ini adalah:

- number_action, yaitu *variable* bertipe *integer* untuk menunjukkan jumlah aksi yang terjadi.
- check_string, yaitu *variable* bertipe *string* yang membantu *pseudocode* untuk menentukan apakah aksi itu *dealing* atau tidak.
- module_street, yaitu *variable* bertipe *string* untuk menunjukkan *street* dalam modul.
- player_action, yaitu *variable* bertipe *string* untuk menunjukkan aksi dari pemain.
- bet_amount, yaitu *variable* bertipe *integer* untuk menunjukkan jumlah keping yang dipertaruhkan saat pemain *bet*.
- call_amount, yaitu *variable* bertipe *integer* untuk menunjukkan jumlah keping yang dipertaruhkan saat pemain *call*.

- raise_amount, yaitu *variable* bertipe *integer* untuk menunjukkan jumlah keping yang dipertaruhkan saat pemain *raise*.
- raise_to, yaitu *variable* bertipe *integer* untuk menunjukkan nilai hasil *raise* sehingga *current bet* sebesar ini.

Tidak ada *output* dari *pseudocode* ini. *Pseudocode* dari fungsi ini dapat dilihat di *pseudocode* 3.2A, *pseudocode* 3.2B, dan *pseudocode* 3.2C.

Pseudocode 3.2A Fungsi ACTION_PROCESS(bagian 1)

```
INPUT number_action
FOR a = 0 to number_action:
    INPUT check_string
    IF check_string = "dealing":
    INPUT module_street
      player_now ← dealer_position
    current_bet ← 0
    FOR i = 0 to number_player:
      player_current_bet[i] ← 0
        action[i] ← 0
      ENDFOR
    ELSE:
      FOR i = 0 to number_player:
        IF player_name[i] = check_string:
          player_index ← i
        ENDIF
      ENDFOR
      player_now ← (player_index+1) modulo n
```

Pseudocode 3.2B Fungsi ACTION_PROCESS(bagian 2)

```
      INPUT player_action
      IF player_action = "folds":
        status[player_index] ← 0
      ELSE IF player_action = "bets":
        INPUT bet_amount
        current_money[player_index] ← current_money[player_index]-bet_amount
        current_bet ← bet_amount
        player_current_bet[player_index] ← bet_amount
      ELSE IF player_action = "calls":
        INPUT call_amount
        current_money[player_index] ← current_money[player_index] - call_amount
        player_current_bet[player_index] ← player_current_bet[player_index] + call_amount
      ELSE IF player_action = "raises":
        INPUT raise_amount
        INPUT raise_to
        current_money[player_index] ← current_money[player_index] - (raise_to - player_current_bet[player_index])
        current_bet ← raise_to
        player_current_bet[player_index] ← raise_to
      ENDIF
      action[player_index] ← 1
      IF current_money[player_index] = 0 :
        status[player_index] ← 0
```

Pseudocode 3.2C Fungsi ACTION_PROCESS(bagian 3)

```
41:       ENDIF
42:    ENDIF
43: ENDFOR
```

3.5.3 Desain Fungsi INPUT_END_MONEY

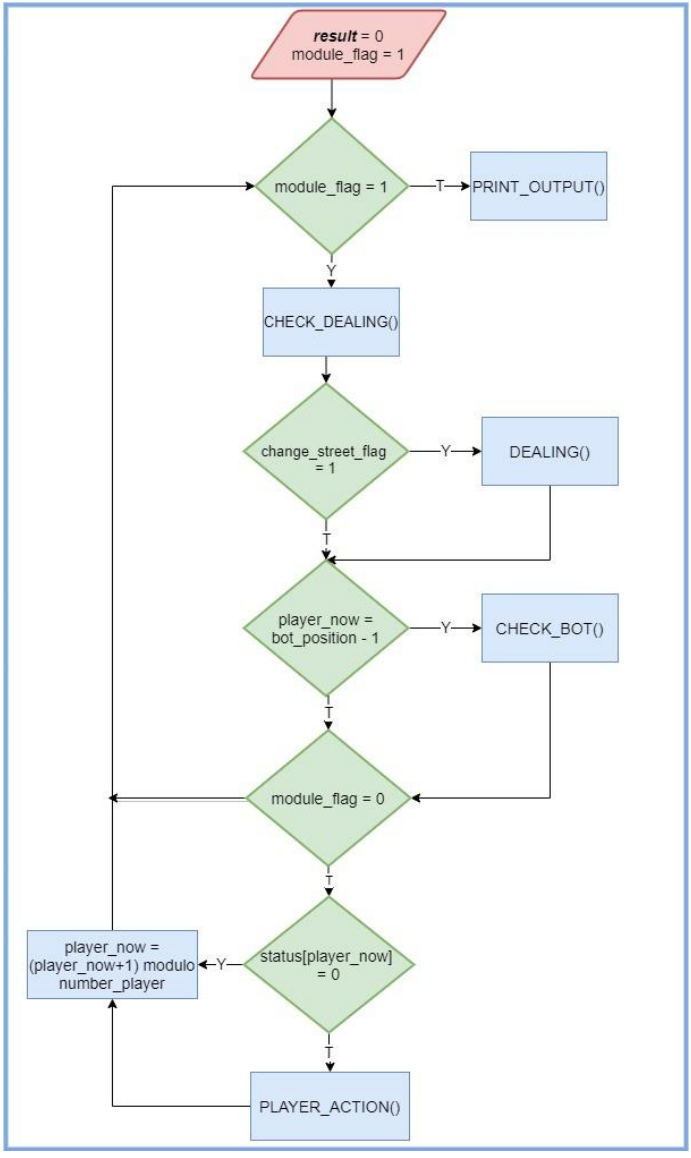
Fungsi ini digunakan untuk menerima *input* jumlah keping akhir setiap pemain. *Input* dari *pseudocode* ini adalah end_money, yaitu *variable* bertipe *array of integer* untuk menunjukkan jumlah keping akhir setiap pemain saat modul dijalankan. Tidak ada *output* dari *pseudocode* ini. *Pseudocode* dari fungsi ini dapat dilihat di *pseudocode* 3.3.

Pseudocode 3.3 Fungsi INPUT_END_MONEY

```
1: FOR i = 0 to number_player:
2:   INPUT end_money[i]
3: ENDFOR
```

3.6 Desain Pemrosesan *Output*

Pemrosesan *output* dilakukan setelah pemrosesan *input* sistem. Pemrosesan *output* dilakukan untuk mendapatkan hasil yang diinginkan sesuai yang dijelaskan pada subbab 2.2.3 Terdapat beberapa fungsi yang digunakan pada pemrosesan *output*. Gambar 3.14 merupakan *flowchart* dari pemrosesan *output*. Terdapat beberapa fungsi yang digunakan yang akan dijelaskan selanjutnya.



Gambar 3.14 *Flowchart* Pemrosesan *Output*

3.6.1 Desain Fungsi CHECK_DEALING

Fungsi ini digunakan untuk mengecek apakah kondisi sekarang sudah memenuhi syarat untuk melakukan *dealing* yang dijelaskan pada subbab 2.1.2. Tidak ada *input* maupun *output* pada *pseudocode* ini. *Pseudocode* dari fungsi ini dapat dilihat di *pseudocode* 3.4.

Pseudocode 3.4 Fungsi CHECK_DEALING

```
1: change_street_flag ← 1
2: FOR i = 0 to number_player:
3:   IF status[i] = 1 AND player_current_bet  ←
curr_bet:
4:     change_street_flag ← 0
5:   ELSE IF status[i] = 1 AND action[i] = 0:
6:     change_street_flag ← 0
7:   ENDIF
8: ENDFOR
9: IF module_street = street_now:
10:    change_street_flag ← 0
11: ENDIF
```

3.6.2 Desain Fungsi DEALING

Fungsi ini digunakan untuk melakukan proses *dealing*. Tidak ada *input* maupun *output* pada *pseudocode* ini. *Pseudocode* dari fungsi ini dapat dilihat di *pseudocode* 3.5.

Pseudocode 3.5 Fungsi DEALING

```
IF module_street = "preflop":
  module_street ← "flop"
ELSE IF module_street = "flop":
  module_street ← "turn"
ELSE IF module_street = "turn":
  module_street ← "river"
ELSE IF module_street = "river":
  module_flag ← 0
  RETURN // EXIT FROM FUNCTION
ENDIF
current_bet ← 0
player_now ← dealer_position
FOR i = 0 to number_player:
   player_current_bet[i] ← 0
   action[i] ← 0
ENDFOR
text_output[result] ← "dealing " + module_street
result ← result+1
```

3.6.3 Desain Fungsi CHECK_BOT

Fungsi ini digunakan untuk mengecek apakah modul sudah selesai atau belum. Tidak ada *input* maupun *output* pada *pseudocode* ini. Pengecekan dilakukan pada giliran *bot*. *Pseudocode* dari fungsi ini dapat dilihat di *pseudocode* 3.6.

Pseudocode 3.6 Fungsi CHECK_BOT

```
bot_check ← 1
FOR i = 0 to number_player:
  IF(current_money[i] > end_money[i]):
    bot_check ← 0
  ENDIF
ENDFOR
IF module_street != street_now:
  bot_check ← 0
ENDIF
IF bot_check = 1:
    module_flag ← 0
ENDIF
```

3.6.4 Desain Fungsi PLAYER_ACTION

Fungsi ini digunakan untuk menentukan aksi apa yang dilakukan suatu pemain dalam modul. Tidak ada *input* maupun *output* pada *pseudocode* ini. *Pseudocode* dari fungsi ini dapat dilihat di *pseudocode* 3.7A, *pseudocode* 3.7B, *pseudocode* 3.7C, dan *pseudocode* 3.7D. Terdapat beberapa fungsi pada *pseudocode* yang akan dijelaskan selanjutnya.

Pseudocode 3.7A Fungsi PLAYER_ACTION(bagian 1)

```
money_difference ← current_money[player_now] - end_money[player_now]
bet_difference ←current_bet - player_current_bet[player_now]
IF current_bet = 0:
    IF module_street = street_now:
    IF current_money[player_now] = end_money[player_now]:
      CHECK()
    ELSE IF current_money[player_now] > end_money[player_now]:
      BET()
    ENDIF
    ELSE IF module_street != street_now:
      POSITION_AFTER_DEALING();
      IF position_flag =  -1:
        IF current_money[player_now] = end_money[player_now]:
          CHECK()
        ELSE IF current_money[player_now] > end_money[player_now]:
          BET()
        ENDIF
      ELSE IF position_flag = 1:
        CHECK()
      ENDIF
 ELSE IF player_current_bet[player_now] = current_bet:
```

Pseudocode 3.7B Fungsi PLAYER_ACTION(bagian 2)

```
22:       IF module_street = street_now:
23:         IF current_money[player_now] =
end_money[player_now]:
24:           CHECK()
25:         ELSE IF current_money[player_now]  >
 end_money[player_now]:
26:           RAISE()
27:         ENDIF
28:    ELSE IF module_street != street_now:
29:       POSITION_AFTER_DEALING();
30:       IF position_flag = -1:
31:         IF current_money[player_now] =
end_money[player_now]:
32:           CHECK()
33:         ELSE IF current_money[player_now] >
end_money[player_now]:
34:           RAISE()
35:         ENDIF
36:       ELSE IF position_flag = 1:
37:         CHECK()
38:       ENDIF
39:  ELSE IF player_current_bet[player_now] <
current_bet:
40:       IF module_street = street_now:
41:         IF current_money[player_now] =
end_money[player_now]:
42:           FOLD()
43:         ELSE IF current_money[player_now] >
end_money[player_now]:
```

Pseudocode 3.7C Fungsi PLAYER_ACTION(bagian 3)

```
44:            IF money_difference  ≤
bet_difference:
45:              CALL_ALL()
46:            ELSE IF money_difference  >
bet_difference:
47:              RAISE()
48:            ENDIF
49:          ENDIF
50:       ELSE IF module_street != street_now:
51:          POSITION_AFTER_DEALING();
52:          IF position_flag = -1:
53:            IF current_money[player_now] =
end_money[player_now]:
54:              FOLD()
55:            ELSE IF
current_money[player_now] >
end_money[player_now]:
56:              IF money_difference  ≤
bet_difference:
57:                CALL_ALL()
58:              ELSE IF money_difference  >
bet_difference:
59:                RAISE()
60:              ENDIF
61:            ENDIF
62:          ELSE IF position_flag = 1:
63:            IF current_money[player_now] =
end_money[player_now]:
64:              FOLD()
```

Pseudocode 3.7D Fungsi PLAYER_ACTION(bagian 4)

```
65:         ELSE IF current_money[player_now] >
end_money[player_now]:
66:           IF money_difference ≤
bet_difference:
67:             CALL_ALL()
68:           ELSE IF money_difference >
bet_difference:
69:             CALL()
70:           ENDIF
71:         ENDIF
72:       ENDIF
73:    ENDIF
74: ENDIF
75: action[player_now]←1
76: result←result + 1
```

3.6.4.1 Desain Fungsi POSITION_AFTER_DEALING

Fungsi ini digunakan untuk mengecek apakah suatu pemain melakukan aksi sebelum atau setelah *bot*. Tidak ada *input* maupun *output* pada *pseudocode* ini. *Pseudocode* dari fungsi ini dapat dilihat di *pseudocode* 3.8.

Pseudocode 3.8 Fungsi POSITION_AFTER_DEALING

```
1: check_position ← dealer_position modulo
number_player
2: position_flag ← 0
3: WHILE position_flag ←  DO:
4:   IF check_position = (bot_position - 1) OR
player_now = bot_position - 1:
5:     position_flag ←  -1
6:   ELSE IF check_position = player_now:
7:     position_flag ←  1
8:   ENDIF
9:   check_position ← check_position + 1 modulo
number_player
10: ENDWHILE
```

3.6.4.2 Desain Fungsi CHECK

Fungsi ini digunakan untuk melakukan aksi *check*. Tidak ada *input* maupun *output* pada *pseudocode* ini. *Pseudocode* dari fungsi ini dapat dilihat di *pseudocode* 3.9.

Pseudocode 3.9 Fungsi CHECK

```
1: text_output[result] ← player_name[player_now]
+ " checks"
```

3.6.4.3 Desain Fungsi FOLD

Fungsi ini digunakan untuk melakukan aksi *fold*. Tidak ada *input* maupun *output* pada *pseudocode* ini. *Pseudocode* dari fungsi ini dapat dilihat di *pseudocode* 3.10.

Pseudocode 3.10 Fungsi FOLD

```
text_output[result] ←player_name[player_now]
+ " folds"
status[player_now] ← 0
```

3.6.4.4 Desain Fungsi BET

Fungsi ini digunakan untuk melakukan aksi *bet*. Tidak ada *input* maupun *output* pada *pseudocode* ini. *Pseudocode* dari fungsi ini dapat dilihat di *pseudocode* 3.11.

Pseudocode 3.11 Fungsi BET

```
string_number_1 ← (STRING)money_difference
current_bet ← money_difference
player_current_bet[player_now] ← current_bet
current_money[player_now] ←
end_money[player_now]
text_output[result] ← player_name[player_now]
+ " bets " + string_number_1
IF current_money[player_now] = 0:
  status[player_now] ←0
ENDIF
```

3.6.4.5 Desain Fungsi RAISE

Fungsi ini digunakan untuk melakukan aksi *raise*. Tidak ada *input* maupun *output* pada *pseudocode* ini. *Pseudocode* dari fungsi ini dapat dilihat di *pseudocode* 3.12.

Pseudocode 3.12 Fungsi RAISE

```
1: string_number_1 ← (STRING)( money_difference -
bet_difference)
2: current_bet ←money_difference +
player_current_bet[player_now]
3: string_number_2 ← (STRING) current_bet
4: player_current_bet[player_now] ←current_bet
5: current_money[player_now] ←
 end_money[player_now]
6: text_output[result] ←player_name[player_now]
+ " raises " + string_number_1 + " to " +
string_number_2
7: IF current_money[player_now] = 0:
8:   status[player_now] ←0
9: ENDIF
```

3.6.4.6 Desain Fungsi CALL_ALL

Fungsi ini digunakan untuk melakukan aksi *call* di mana jumlah *call* yang dilakukan sebesar selisih jumlah keping saat ini dan jumlah keping akhir. *Call* yang dilakukan belum tentu sebesar selisih antara bet pemain dengan *current bet* karena jumlah keping pemain tidak cukup sehingga pemain melakukan *call all-in*. Tidak ada *input* maupun *output* pada *pseudocode* ini. *Pseudocode* dari fungsi ini dapat dilihat di *pseudocode* 3.13.

Pseudocode 3.13 Fungsi CALL_ALL

```
1: string_number_1 ← (STRING)money_difference
2: player_current_bet[player_now] ←
 money_difference +
player_current_bet[player_now]
3: current_money[player_now] ←
 end_money[player_now]
4: text_output[result] ←player_name[player_now]
+ " calls " + string_number_1
5: IF current_money[player_now] = 0:
6:   status[player_now] ←0
7: ENDIF
```

3.6.4.7 Desain Fungsi CALL

Fungsi ini digunakan untuk melakukan aksi *call* di mana jumlah *call* yang dilakukan sebesar selisih *bet* pemain dengan *current bet*. Tidak ada *input* maupun *output* pada *pseudocode* ini. *Pseudocode* dari fungsi ini dapat dilihat di *pseudocode* 3.14.

Pseudocode 3.14 Fungsi CALL

```
1: string_number_1 ← (STRING)bet_difference
2: player_current_bet[player_now] ← current_bet
3: current_money[player_now] ←
current_money[player_now] - bet_difference
4: text_output[result] ← player_name[player_now]
+ " calls " + string_number_1
5: IF current_money[player_now] = 0:
6:   status[player_now] ← 0
7: ENDIF
```

3.6.5 Desain Fungsi PRINT_OUTPUT

Fungsi ini digunakan untuk mencetak hasil yang sudah didapatkan oleh sistem dengan format yang dijelaskan pada subbab 2.2.3. Tidak ada *input* pada *pseudocode* ini. *Output* dari *pseudocode* ini adalah result, yaitu *variable* bertipe *integer* untuk menunjukkan jumlah aksi yang terjadi yang belum diketahui *bot* saat modul dipanggil, dan text_output, yaitu *variable* bertipe *array of string* untuk menunjukkan aksi-aksi yang terjadi yang belum diketahui *bot* saat modul dipanggil. *Pseudocode* dari fungsi ini dapat dilihat di *pseudocode* 3.15.

Pseudocode 3.15 Fungsi PRINT_OUTPUT

```
1: PRINT result
2: FOR i = 0 to result:
3: PRINT text_output[i]
4: ENDFOR
```

BAB IV
IMPLEMENTASI

Bab ini menjelaskan mengenai implementasi algoritma dari rancangan sistem yang telah dibahas pada Bab 3 meliputi kode program.

4.1 Lingkungan Implementasi

Lingkungan implementasi dan pengembangan yang dilakukan adalah sebagai berikut.

1. Perangkat Keras:
   - Processor Intel(R) Core(TM) i3-5005U CPU @ 2.00GHz (4 CPUs), ~2.00GHz
   - *Random Access Memory* 12GB
2. Perangkat Lunak:
   - Sistem Operasi Windows 8.1 Pro 64-bit
   - DevC++
   - C++5.1.2

4.2 Implementasi Program Utama

Subbab ini menjelaskan implementasi program utama. Program ini merupakan program yang digunakan untuk menyelesaikan permasalahan 2082 – Poker. Program diimplementasikan dengan menggunakan bahasa pemrograman C++ karena dalam pengimplementasian algoritma solusi dibutuhkan batas waktu dan memori yang sangat terbatas untuk komputasi yang besar.

4.2.1 Penggunaan *Library*

Program ini menggunakan *library* seperti yang ditunjukkan pada Kode Sumber 4.1.

```
1. #include <iostream>
2. #include <cstdio>
3. #include <sstream>
4. #include <string>
5. #include <cstring>
6. #include <algorithm>
7. #include <vector>
```

Kode Sumber 4.1 *Header* Program Utama

4.2.2 Preprocessor

Preprocessor dapat dilihat pada Kode Sumber 4.2.

```
1. using namespace std;
2. int small_blinds, big_blinds, ante;
3. int number_player, dealer_position;
4. int bot_position;
4. string module_street, street_now;
5. int current_bet;
6. string player_name[10];
7. int current_money[10];
8. int status[10];
9. int player_current_bet[10];
10. int action[10];
11. int sb, bb, player_now, player_index;
12. int number_action;
13. string check_string, player_act;
14. int raise_amount, raise_to;
15. int bet_amount, call_amount;
16. int end_money[10];
17. int result;
18. int module_flag, change_street_flag;
19. int position_flag;
20. int check_position,bot_check;
21. int money_difference, bet_difference;
22. string space, temp;
23. string string_number_1, string_number_2;
24. stringstream ss;
25. vector<string> text_output;
```

Kode Sumber 4.2 *Preprocessor* Program Utama

using namespace std digunakan agar tidak perlu menulis *std::* di awal setiap baris. Hampir semua *Variable-variable* yang dideklarasikan pada Kode Sumber 4.2 sudah dijelaskan pada subbab 3.3. *Text_output* bertipe *vector of string* untuk memudahkan proses penyimpanan aksi yang dilakukan. *Stringstream ss* digunakan untuk mengubah *integer* menjadi *string*.

4.2.3 Implementasi Fungsi *Main*

Fungsi *Main* nantinya menjadi fungsi yang dijalankan pertama kali oleh sistem. Fungsi ini berisikan fungsi-fungsi lainnya serta pemrosesan yang telah dijelaskan oleh *flowchart* pada subbab 3.4 dan 3.5. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.3.

```
1. int main(){
2.    ios::sync_with_stdio(0);
3.    beginstate();
4.    action_process();
5.    input_end_money();
6.    result=0;
7.    module_flag=1;
8.    while(module_flag){
9.       ss.clear();
10.      ss.str("");
11.      temp="";
12.      string_number_1="";
13.      string_number_2="";
14.      check_dealing();
15.      if(change_street_flag==1) dealing();
16.      if(player_now==(bot_position-
1))check_bot();
17.      if(module_flag==0)break;
18.      if(status[player_now]!=0){
19.         player_action();
20.      }
21.   player_now=(player_now+1)%number_player;
22.   }
23.   print_output();
24.   return 0;
25. }
```

Kode Sumber 4.3 Implementasi Fungsi *Main*

ios::sync_with_stdio(0) digunakan untuk mempercepat waktu ketika menggunakan *cin* dan *cout*. *Variable-variable* string pembantu, yaitu *ss*, *temp*, *string_number_1*, dan *string_number_2*, dikosongkan terlebih dahulu sebelum dipakai kembali.

4.3 Implementasi Fungsi

Pada subbab ini akan dijelaskan implementasi dari fungsi-fungsi yang sesuai dengan desainnya pada bab 3.

4.3.1 Fungsi beginstate

Berdasarkan penjelasan pada subbab 3.4.1 fungsi ini digunakan untuk membaca beberapa *input* awal, memberi nilai pada beberapa *variable* serta menentukan pemain mana yang bertaruh *small blinds* maupun *big blinds*. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.4A dan Kode Sumber 4.4B.

```
1. void beginstate(){
2. cin>>small_blinds>>big_blinds>>ante;
3.   cin>> number_player >> dealer_position >>
bot_position;
4. cin>>street_now;
5. module_street="preflop";
6. current_bet=big_blinds;
7. for(int i=0;i<number_player;i++){
8.      cin>> player_name[i] >>  current_money[i] ;
9.    current_money[i]-=ante;
10.     status[i]=1;
11.     player_current_bet[i]=0;
12.     action[i]=0;
13.     if(current_money[i]<=0){
14.        status[i]=0;
15.        current_money[i]<=0;
16.     }
17.   }
```

Kode Sumber 4.4A Implementasi Fungsi beginstate (bagian 1)

```
18.   if(number_player==2){
19.      sb=(dealer_position - 1 ) % number_player;
20.      bb=(sb+1)%number_player;
21.      player_now=sb;
22.   }
23.   else{
24.      sb=dealer_position%number_player;
25.      bb=(sb+1)%number_player;
26.      player_now=(dealer_position + 2) %
number_player;
27.   }
28.   if(current_money[sb]<small_blinds){
29.      player_current_bet[sb] = current_money[sb];
30.      current_money[sb]=0;
31.      status[sb]=0;
32.   }
33.   else{
34.      player_current_bet[sb]=small_blinds;
35.      current_money[sb]-=small_blinds;
36.   }
37.   if(current_money[bb]<big_blinds){
38.      player_current_bet[bb] = current_money[bb];
39.      current_money[bb]=0;
40.      status[bb]=0;
41.   }
42.   else{
43.      player_current_bet[bb]=big_blinds;
44.      current_money[bb]-=big_blinds;
45.   }
46. }
```

Kode Sumber 4.4B Implementasi Fungsi beginstate (bagian 2)

4.3.2 Fungsi action_process

Berdasarkan penjelasan pada subbab 3.4.2 fungsi ini digunakan untuk memproses *input* aksi yang diketahui oleh *bot* saat modul dijalankan. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.5A dan Kode Sumber 4.5B.

```
1. void action_process(){
2. cin>>number_action;
3. for(int i=0;i<number_action;i++){
4.    cin>>check_string;
5.    if(check_string=="dealing"){
6.       cin>>module_street;
7.       player_now = dealer_position %
number_player;
8.       current_bet=0;
9.       for(int j = 0;j < number_player;j++){
10.           player_current_bet[j]=0;
11.           action[j]=0;
12.        }
13.     }
14.     else {
15.        for( int j = 0;j < number_player;j++){
16.           if(player_name[j] == check_string){
17.              player_index=j;
18.              break;
19.           }
20.        }
21.        cin>>player_act;
22.        player_now = (player_index + 1) %
number_player;
23.        if(player_act=="folds"){
24.           status[player_index]=0;
25.        }
26.        else if(player_act=="bets"){
27.           cin>>bet_amount;
28.           current_money [player_index] -=
bet_amount;
29.           current_bet=bet_amount;
30.           player_current_bet [player_index] =
bet_amount;
31.        }
```

Kode Sumber 4.5A Implementasi Fungsi action_process (bagian 1)

```
32.        else if(player_act=="calls"){
33.           cin>>call_amount;
34.           current_money [player_index] -=
call_amount;
35.           player_current_bet [player_index] +=
call_amount;
36.        }
37.        else if(player_act=="raises"){
38.           cin >> raise_amount >> space >>
raise_to;
39.           current_money [player_index]-= (raise_to
- player_current_bet[player_index]);
40.           current_bet=raise_to;
41.           player_current_bet [player_index] =
raise_to;
42.        }
43.        action[player_index]=1;
44.        if(current_money[player_index] == 0){
45.           status[player_index]=0;
46.        }
47.     }
48.   }
49. }
```

Kode Sumber 4.5B Implementasi Fungsi action_process (bagian 2)

4.3.3 Fungsi input_end_money

Berdasarkan penjelasan pada subbab 3.4.3 fungsi ini digunakan untuk menerima *input* jumlah keping akhir setiap pemain. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.6.

```
1. void input_end_money(){
2.   for(int i=0;i<number_player;i++){
3.      cin>>end_money[i];
4.   }
5. }
```

Kode Sumber 4.6 Implementasi Fungsi input_end_money

4.3.4 Fungsi check_dealing

Berdasarkan penjelasan pada subbab 3.5.1 fungsi ini digunakan untuk mengecek apakah kondisi sekarang sudah memenuhi syarat untuk melakukan *dealing* yang dijelaskan pada subbab 2.1.2. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.7.

```
1. void check_dealing(){
2. change_street_flag=1;
3. for(int i=0;i<number_player;i++){
4.    if(status[i]==1 &&
player_current_bet[i]<current_bet){
5.       change_street_flag=0;
6.       break;
7.    }
8.    else if(status[i]==1 && action[i]==0){
9.       change_street_flag=0;
10.         break;
11.      }
12.   }
13.   if(module_street==street_now){
14.      change_street_flag=0;
15.   }
16. }
```

Kode Sumber 4.7 Implementasi Fungsi check_dealing

4.3.5 Fungsi dealing

Berdasarkan penjelasan pada subbab 3.5.2 fungsi ini digunakan untuk melakukan proses *dealing*. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.8.

```
1. void dealing(){
2. if(module_street=="preflop")
module_street="flop";
3. else if(module_street=="flop")
module_street="turn";
4. else if(module_street=="turn")
module_street="river";
5. else if(module_street=="river") {
6.    module_flag=0;
7.    return;
8. }
9. current_bet=0;
10.   player_now=dealer_position%number_player;
11.   for(int i=0;i<number_player;i++){
12.      player_current_bet[i]=0;
13.      action[i]=0;
14.   }
15.   temp="dealing "+module_street;
16.   text_output.push_back(temp);
17.   result++;
18. }
```

Kode Sumber 4.8 Implementasi Fungsi dealing

4.3.6 Fungsi check_bot

Berdasarkan penjelasan pada subbab 3.5.3 fungsi ini digunakan untuk mengecek apakah modul sudah selesai atau belum. Pengecekan dilakukan pada giliran *bot*. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.9.

```
1. void check_bot(){
2.    bot_check=1;
3. for(int i=0;i<number_player;i++){
4.    if(current_money[i]>end_money[i]){
5.       bot_check=0;
6.       break;
7.    }
8. }
9. if(module_street!=street_now)bot_check=0;
10.   if(bot_check) module_flag=0;
11. }
```

Kode Sumber 4.9 Implementasi Fungsi check_bot

4.3.7 Fungsi player_action

Berdasarkan penjelasan pada subbab 3.5.4 fungsi ini digunakan untuk menentukan aksi apa yang dilakukan suatu pemain dalam modul. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.10.

```
1. void player_action(){
2. money_difference = current_money[player_now]-
end_money[player_now];
3. bet_difference = current_bet -
player_current_bet[player_now];
4. if(current_bet==0){
5.   if(module_street==street_now){
6.     if( current_money[player_now] ==
end_money[player_now])check();
7.     else if ( current_money [player_now] >
end_money[player_now])bet();
8.   }
9.   else if(module_street!=street_now){
10.       position_after_dealing();
11.       if(position_flag==-1){
12.         if(current_money [player_now] ==
end_money[player_now])check();
13.         else if (current_money[player_now]
> end_money[player_now])bet();
14.       }
15.       else if(position_flag==1) check();
16.     }
17.  }
18.  else if(player_current_bet[player_now] ==
current_bet){
```

Kode Sumber 4.10A Implementasi Fungsi player_action (bagian 1)

```
19.     if(module_street==street_now){
20.       if(current_money[player_now] ==
end_money[player_now])check();
21.       else if(current_money [player_now] >
end_money[player_now])raise();
22.     }
23.     else if(module_street!=street_now){
24.       position_after_dealing();
25.       if(position_flag==-1){
26.         if(current_money [player_now] ==
end_money[player_now])check();
27.         else if(current_money [player_now] >
end_money[player_now])raise();
28.       }
29.       else if(position_flag==1) check();
30.     }
31.   }
32.   else if(player_current_bet
[player_now]<current_bet){
33.     if(module_street==street_now){
34.       if(current_money
[player_now]==end_money[player_now])fold();
35.       else if(current_money [player_now] >
end_money[player_now]){
36.         if(money_difference <=
bet_difference)call_all();
37.         else if(money_difference >
bet_difference)raise();
38.       }
39.     }
40.     else if(module_street!=street_now){
41.       position_after_dealing();
```

Kode Sumber 4.10B Implementasi Fungsi player_action (bagian 2)

```
42.       if(position_flag==-1){
43.         if(current_money
[player_now]==end_money[player_now])fold();
44.         else if(current_money
[player_now]>end_money[player_now]){
45.           if( money_difference <=
bet_difference)call_all();
46.           else if
(money_difference>bet_difference)raise();
47.         }
48.       }
49.       else if(position_flag==1) {
50.         if(current_money
[player_now]==end_money[player_now])fold();
51.         else if(current_money
[player_now]>end_money[player_now]){
52.           if (money_difference
<=bet_difference)call_all();
53.           else if
(money_difference>bet_difference)call();
54.         }
55.       }
56.     }
57.  }
58.  action[player_now]=1;
59.  result++;
60. }
```

Kode Sumber 4.10C Implementasi Fungsi player_action (bagian 3)

4.3.8 Fungsi position_after_dealing

Berdasarkan penjelasan pada subbab 3.5.4.1 fungsi ini digunakan untuk mengecek apakah suatu pemain melakukan aksi sebelum atau setelah *bot*. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.11.

```
1. void position_after_dealing(){
2. check_position = dealer_position %
number_player;
3. position_flag=0;
4. while(position_flag==0){
5.   if(check_position==(bot_position-1) ||
player_now==(bot_position-1)){
6.     position_flag=-1;
7.   }
8.   else if(check_position==player_now){
9.     position_flag=1;
10.    }
11.    check_position =(check_position+1) %
number_player;
12.  }
13. }
```

Kode Sumber 4.11 Implementasi Fungsi position_after_dealing

4.3.9 Fungsi *check*

Berdasarkan penjelasan pada subbab 3.5.4.2 fungsi ini digunakan untuk melakukan aksi *check*. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.12.

```
1. void check(){
2. temp = player_name[player_now]+" checks";
3. text_output.push_back(temp);
4. }
```

Kode Sumber 4.12 Implementasi Fungsi check

4.3.10 Fungsi *fold*

Berdasarkan penjelasan pada subbab 3.5.4.3 fungsi ini digunakan untuk melakukan aksi *fold*. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.13.

```
1. void fold(){
2. temp=player_name[player_now]+" folds";
3. text_output.push_back(temp);
4. status[player_now]=0;
5. }
```

Kode Sumber 4.13 Implementasi Fungsi fold

4.3.11 Fungsi *bet*

Berdasarkan penjelasan pada subbab 3.5.4.4 fungsi ini digunakan untuk melakukan aksi *bet*. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.14.

```
1. void bet(){
2. ss<<money_difference;
3. string_number_1=ss.str();
4. current_bet=money_difference;
5. player_current_bet[player_now] =
current_bet;
6. current_money[player_now] =
end_money[player_now];
7. temp=player_name[player_now]+" bets
"+string_number_1;
8. text_output.push_back(temp);
9. if(current_money[player_now]==0){
10.     status[player_now]=0;
11.  }
12. }
```

Kode Sumber 4.14 Implementasi Fungsi bet

4.3.12 Fungsi *raise*

Berdasarkan penjelasan pada subbab 3.5.4.5 fungsi ini digunakan untuk melakukan aksi *raise*. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.15.

```
1. void raise(){
2. ss<<(money_difference-bet_difference);
3. string_number_1=ss.str();
4. ss.clear();
5. ss.str("");
6. current_bet = money_difference +
player_current_bet[player_now];
7. ss<<(current_bet);
8. string_number_2=ss.str();
9. ss.clear();
10.  ss.str("");
11.  player_current_bet[player_now] =
current_bet;
12.  current_money[player_now] =
end_money[player_now];
13.  temp=player_name[player_now]+" raises
"+string_number_1+" to "+string_number_2;
14.  text_output.push_back(temp);
15.  if(current_money[player_now]==0){
16.    status[player_now]=0;
17.  }
18. }
```

Kode Sumber 4.15 Implementasi Fungsi raise

4.3.13 Fungsi call_all

Berdasarkan penjelasan pada subbab 3.5.4.6 fungsi ini digunakan untuk melakukan aksi *call* di mana jumlah *call* yang dilakukan sebesar selisih jumlah keping saat ini dan jumlah keping akhir. *Call* yang dilakukan belum tentu sebesar selisih antara bet pemain dengan *current bet* karena jumlah keping pemain tidak cukup sehingga pemain melakukan *call all-in*. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.16.

```
1. void call_all(){
2. ss << (money_difference);
3. string_number_1=ss.str();
4. player_current_bet[player_now] +=
money_difference;
5. temp=player_name[player_now]+" calls
"+string_number_1;
6. text_output.push_back(temp);
7. current_money[player_now] =
end_money[player_now];
8. if(current_money[player_now]==0){
9.   status[player_now]=0;
10.  }
11. }
```

Kode Sumber 4.16 Implementasi Fungsi call_all

4.3.14 Fungsi *call*

Berdasarkan penjelasan pada subbab 3.5.4.6 fungsi ini digunakan untuk melakukan aksi *call* di mana jumlah *call* yang dilakukan sebesar selisih *bet* pemain dengan *current bet*. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.17.

```
1. void call(){
2. ss << (bet_difference);
3. string_number_1=ss.str();
4. player_current_bet[player_now] =
current_bet;
5. current_money[player_now]-=bet_difference;
6. temp=player_name[player_now]+" calls
"+string_number_1;
7. text_output.push_back(temp);
8. if(current_money[player_now]==0){
9.   status[player_now]=0;
10.  }
11. }
```

Kode Sumber 4.17 Implementasi Fungsi call

4.3.15 Fungsi print_output

Berdasarkan penjelasan pada subbab 3.5.5 fungsi ini digunakan untuk mencetak hasil yang sudah didapatkan oleh sistem dengan format yang dijelaskan pada subbab 2.2.3. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.18.

```
1.void print_output(){
2.  cout<<result<<endl;
3.  if(result>0){
4.    for(int i=0;i<text_output.size(); i++){
5.      cout<<text_output[i]<<endl;
6.    }
7.  }
8.}
```

Kode Sumber 4.18 Implementasi Fungsi print_output

BAB V
UJICOBA DAN EVALUASI

Pada bab ini dijelaskan tentang uji coba dan evaluasi dari implementasi yang telah dilakukan pada tugas akhir ini.

5.1 Lingkungan Uji Coba

Lingkungan uji coba yang digunakan untuk uji coba kebenaran adalah sebuah *virtual machine* yang digunakan pada situs penilaian daring Timus Online Judge dengan spesifikasi sebagai berikut:

1. Perangkat Keras:
   - *Processor* Intel Xeon E5-2665 v5 2400 MHz

2. Perangkat Lunak:
   - Kompiler G++ 7.1
   - Sistem Operasi Windows 10 64-bit

Lingkungan uji coba yang digunakan untuk uji coba kinerja menggunakan komputer pribadi milik penulis dengan spesifikasi sebagai berikut:

3. Perangkat Keras:
   - Processor Intel(R) Core(TM) i3-5005U CPU @ 2.00GHz (4 CPUs), ~2.00GHz
   - *Random Access Memory* 12GB
4. Perangkat Lunak:
   - Sistem Operasi Windows 8.1 Pro 64-bit
   - DevC++
   - C++5.1.2

5.2 Skenario Uji Coba

Pada bagian ini akan dijelaskan skenario yang akan digunakan untuk melakukan pengujian terhadap implementasi yang dibuat untuk permasalahan 2082 – Poker.

Uji coba kebenaran akan dilakukan dengan melihat umpan balik yang diberikan oleh Timus Online Judge setelah sumber kode dikirimkan. Timus Online Judge akan mengecek kebenaran dari sumber kode yang dikirim dengan memasukkan kasus uji dengan parameter yang sudah dijabarkan pada subbab 2.2.1.

Uji coba kinerja akan dilakukan dengan mengirimkan kode sumber hasil implementasi program ke situs penilaian Timus Online Judge sebanyak 10 kali kemudian menganalisa performa dari umpan balik yang diberikan.

5.2.1 Evaluasi Kebenaran

Evaluasi dilakukan dengan mengecek *input* yang diberikan dengan *output* yang dihasilkan dari implementasi program yang sudah dibuat sama dengan contoh *output* yang ada pada permasalahan Timus Online Judge 2082 - Poker. Terdapat empat buah contoh *input* dan *output*. Kasus uji dapat dilihat pada Tabel 5.1.

Tabel 5.1 Tabel Uji Coba

Input	*Output*
10 20 2 5 3 4 flop PhilIvey 500 TomDwan 500 ViktorBlom 500 Grobot 500 Ziigmund 500 3 PhilIvey folds TomDwan calls 20 ViktorBlom raises 40 to 60 498 478 438 438 478	4 Grobot calls 50 Ziigmund folds TomDwan folds dealing flop

Pada program implementasi yang dibuat, sistem akan membaca 3 buah angka dari masukan yaitu 10(*sb*) sebagai *small blinds*, 20(*bb*) sebagai *big blinds*, dan 2(*a*) sebagai ante. Kemudian, sistem akan membaca 3 buah angka lagi yaitu 5(*n*) sebagai jumlah pemain, 3(*d*) sebagai posisi *dealer*, dan 4(*h*) sebagai posisi *bot*. Setelah itu, sistem akan membaca sebuah *string* sebagai *street* saat modul dijalankan. Lalu, sistem akan masuk ke dalam perulangan sebanyak 5 kali untuk mendapatkan 5 nama pemain berserta jumlah keping masing-masing pemain. Nama pemain pertama yang dimasukkan adalah PhilIvey dengan jumlah keping 500. Nama pemain kedua yang dimasukkan adalah TomDwan dengan jumlah keping 500. Nama pemain ketiga yang dimasukkan adalah ViktorBlom dengan jumlah keping 500. Nama pemain keempat yang dimasukkan adalah GroBot dengan jumlah keping 500. Nama pemain kelima yang dimasukkan adalah Ziigmund dengan jumlah keping 500. Selama perulangan ini, jumlah keping masing-masing pemain akan dikurangi sebesar *a*. Setelah itu, pemain keempat bertaruh *small blinds* dan pemain

kelima bertaruh *big blinds*. Jumlah keping masing-masing pemain setelah *blinds* ditunjukkan pada Tabel 5.2

Tabel 5.2 Jumlah Keping Pemain setelah *Blinds*

Nama Pemain	PhilIvey	TomDwan	ViktorBlom	GroBot	Ziigmund
Jumlah Keping	498	498	498	488	478

Sistem akan membaca sebuah angka yaitu 3(*k*) sebagai jumlah aksi yang terjadi. Setelah itu, sistem akan masuk ke dalam perulangan sebanyak 3 kali untuk mendapatkan 3 aksi yang terjadi. Aksi-aksi ini sudah dijelaskan pada subbab 2.1.1. Kalimat pertama adalah "PhilIvey folds", berarti PhilIvey melakukan aksi berupa *fold*. Kalimat kedua adalah "TomDwan calls 20", berarti TomDwan melakukan aksi berupa *call* sebesar 20. Kalimat ketiga adalah "ViktorBlom raises 40 to 60", berarti ViktorBlom melakukan *raise* sebesar 40 menjadi 60. Ketiga aksi ini diproses sehingga didapatkan kondisi jumlah keping setiap pemain yang ditunjukkan pada Tabel 5.3.

Tabel 5.3 Jumlah Keping Pemain setelah Aksi

Nama Pemain	PhilIvey	TomDwan	ViktorBlom	GroBot	Ziigmund
Jumlah Keping	498	478	438	488	478

Lalu, sistem akan masuk perulangan lagi sebanyak jumlah pemain, yaitu 5 kali, untuk mendapatkan jumlah keping akhir setiap pemain. Pemain pertama, PhilIvey, memiliki jumlah keping 498. Pemain kedua, TomDwan, memiliki jumlah keping 478. Pemain ketiga, ViktorBlom, memiliki jumlah keping 438. Pemain keempat, GroBot, memiliki jumlah keping 438. Pemain kelima, Ziigmund, memiliki jumlah keping 478. Tabel 5.4 menunjukkan jumlah keping saat ini dan jumlah keping akhir pemain.

Tabel 5.4 Jumlah Keping Pemain Saat Ini dan Akhir

Nama Pemain	PhilIvey	TomDwan	ViktorBlom	GroBot	Ziigmund
Jumlah Keping Saat ini	498	478	438	488	478
Jumlah Keping Akhir	498	478	438	438	478

Setelah selesai memproses masukan, sistem akan memproses data yang dimiliki sesuai dengan subbab 2.4. Sistem akan melakukan pengecekan pada pemain setelah aksi yang diketahui, yaitu GroBot. Sistem akan menentukan bahwa GroBot akan melakukan *call* sebesar 60-10, yaitu 50. Sistem akan menyimpan *string* "Grobot calls 50". Kondisi jumlah keping setelah pengecekan pertama ditunjukkan pada Tabel 5.5.

Tabel 5.5 Jumlah Keping Pemain Setelah Pengecekan Pertama

Nama Pemain	PhilIvey	TomDwan	ViktorBlom	GroBot	Ziigmund
Jumlah Keping Saat ini	498	478	438	488	478
Jumlah Keping Akhir	498	478	438	438	478

Pengecekan selanjutnya dilakukan pada Ziigmund. Sistem akan menentukan bahwa Ziigmund akan melakukan *fold*. Sistem akan menyimpan *string* "Ziigmund folds". Kemudian, dilakukan pengecekan pada PhilIvey. Karena PhilIvey telah *fold*, pengecekan dilanjutkan pada TomDwan. Sistem akan menentukan bahwa TomDwan akan melakukan *fold*. Sistem akan menyimpan *string* "TomDwan *fold*". Setelah itu, sistem akan

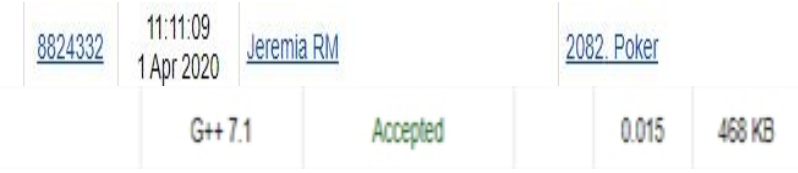
menentukan bahwa terjadi pergantian *street*. Sistem akan menyimpan *string* “dealing flop”. Lalu, sistem akan menghentikan proses pengecekan karena kondisi telah terpenuhi. Sistem akan memberikan keluaran berupa sebuah angka 4, yaitu jumlah *string* yang disimpan sistem. Akhirnya, sistem akan mengeluarkan *string* yang telah disimpan secara urut. Keluaran ini ditunjukkan pada Tabel 5.6.

Tabel 5.6 Keluaran Sistem

4 Grobot calls 50 Ziigmund folds TomDwan folds dealing flop

5.2.2 Uji Coba Kebenaran

Uji coba kebenaran dilakukan dengan mengirimkan kode hasil implementasi program ke situs Timus Online Judge. Permasalahan yang diselesaikan adalah 2082 – Poker. Setelah mengirimkan kode sumber maka akan mendapatkan umpan balik dari Timus Online Judge seperti yang ada pada **Gambar 5.1**.



Gambar 5.1 Hasil umpan balik dari hasil uji kebenaran di Timus Online Judge

Dari hasil uji coba yang dilakukan kode sumber mendapatkan umpan balik *Accepted*. Waktu yang diperlukan program adalah 0,015 detik dan memori yang dibutuhkan program adalah 468 KB.

5.2.3 Uji Coba Kinerja

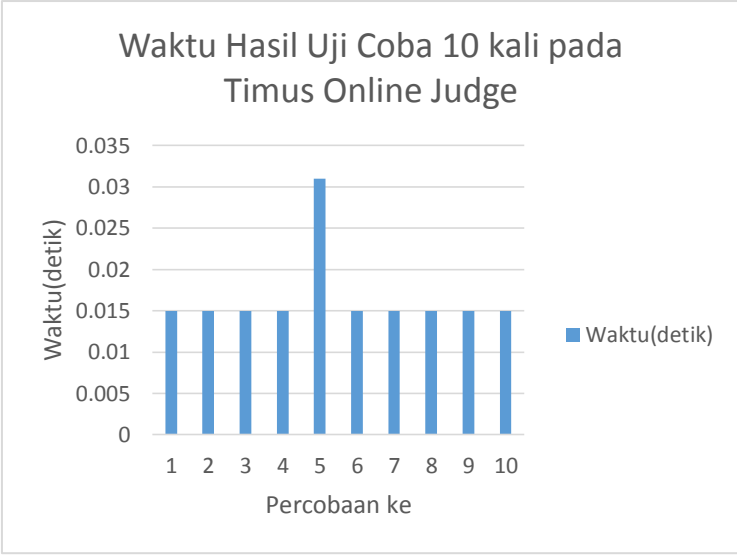
Setelah itu kode sumber yang sama akan dikirimkan sampai 10 kali untuk melihat variasi waktu dan memori yang dibutuhkan. Hasil uji coba dengan mengirimkan kode sumber sebanyak 10 kali, dapat dilihat pada Gambar 5.2, Tabel 5.7, Gambar 5.3, dan Gambar 5.4.

Date	Author	Problem	Language	Judgement result	Test #	Execution time	Memory used
19:01:12 1 Apr 2020	Jeremia RM	2082_Poker	G++ 7.1	Accepted		0.015	468 KB
18:54:11 1 Apr 2020	Jeremia RM	2082_Poker	G++ 7.1	Accepted		0.015	468 KB
18:49:23 1 Apr 2020	Jeremia RM	2082_Poker	G++ 7.1	Accepted		0.015	472 KB
18:45:49 1 Apr 2020	Jeremia RM	2082_Poker	G++ 7.1	Accepted		0.015	472 KB
18:39:50 1 Apr 2020	Jeremia RM	2082_Poker	G++ 7.1	Accepted		0.015	468 KB
18:36:22 1 Apr 2020	Jeremia RM	2082_Poker	G++ 7.1	Accepted		0.031	476 KB
18:32:45 1 Apr 2020	Jeremia RM	2082_Poker	G++ 7.1	Accepted		0.015	468 KB
18:30:13 1 Apr 2020	Jeremia RM	2082_Poker	G++ 7.1	Accepted		0.015	468 KB
18:27:40 1 Apr 2020	Jeremia RM	2082_Poker	G++ 7.1	Accepted		0.015	468 KB
18:22:22 1 Apr 2020	Jeremia RM	2082_Poker	G++ 7.1	Accepted		0.015	480 KB

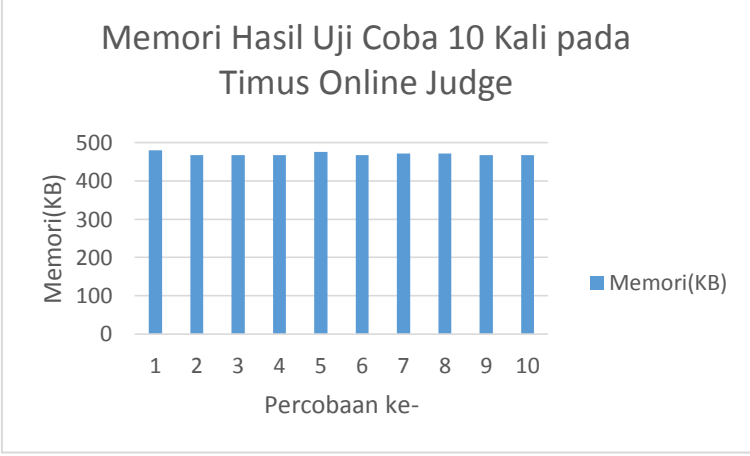
Gambar 5.2 Hasil umpan balik dari uji kebenaran 10 kali pada Timus Online Judge

Tabel 5.7 Tabel Waktu dan Memori dari Uji Coba Kebenaran 10 Kali pada Timus Online Judge

Percobaan ke-	Waktu (detik)	Memori (KB)
1	0,015	480
2	0,015	468
3	0,015	468
4	0,015	468
5	0,031	476
6	0,015	468
7	0,015	472
8	0,015	472
9	0,015	468
10	0,015	468



Gambar 5.3 Grafik Waktu Hasil Uji Kebenaran Sebanyak 10 kali pada Timus Online Judge



Gambar 5.4 Grafik Memori (KB) Hasil Uji Kebenaran Sebanyak 10 kali pada Timus Online Judge

Dari hasil uji coba kebenaran sebanyak 10 kali pada Timus Online Judge, maka dapat dilihat bahwa rata-rata memori yang dibutuhkan oleh program adalah 470,8 KB, sedangkan waktu rata-rata yang dibutuhkan program adalah 0,0166 detik.

[*Halaman ini sengaja dikosongkan*]

BAB VI
KESIMPULAN DAN SARAN

Pada bab ini akan dijelaskan kesimpulan dari hasil uji coba yang telah dilakukan serta saran-saran tentang pengembangan yang dapat dilakukan terhadap tugas akhir ini di masa yang akan datang.

6.1 Kesimpulan

Dari analisis dan uji coba yang dilakukan terhadap implementasi algoritma untuk menyelesaikan permasalahan 2082 – Poker, dapat diambil kesimpulan sebagai berikut:

1. Permasalahan 2082 – Poker di situs Timus Online Judge dapat diselesaikan dengan memodelkan permasalahan tersebut dengan menggunakan metode *greedy*.
2. Rata-rata waktu dari implementasi algoritma *greedy* kurang dari batas waktu yang diberikan oleh soal dengan waktu rata-rata 0,0166 detik dan memori rata-rata yang digunakan 470,8 KB.

6.2 Saran

Pada tugas akhir ini tentunya terdapat kekurangan serta nilai-nilai yang dapat penulis ambil. Berikut adalah saran-saran yang dapat diambil melalui tugas akhir ini:

- Untuk ke depannya, materi yang ada pada tugas akhir ini dapat menjadi bahan riset untuk mencari optimasi yang lebih lanjut.

[*Halaman ini sengaja dikosongkan*]

DAFTAR PUSTAKA

[1] Cardschat, “Texas Hold'em Rules: Limit and No Limit,” 14 October 2007. [Online]. Available: https://www.cardschat.com/texas-holdem-poker-rules.php. [Accessed 11 March 2019].

[2] I. Chevdar, “2082. Poker,” 3 May 2016. [Online]. Available: https://acm.timus.ru/problem.aspx?space=1&num=2082. [Accessed 11 March 2019].

[3] S. Jain, “Greedy Algorithms - Geeksforgeeks,” 7 May 2013. [Online]. Available: https://www.geeksforgeeks.org/greedy-algorithms/. [Accessed 19 March 2019].

[*Halaman ini sengaja dikosongkan*]

LAMPIRAN A: DATA UJI

Berikut merupakan data uji coba yang digunakan untuk uji coba kebenaran.

Input	10 20 2 5 3 4 flop PhilIvey 500 TomDwan 500 ViktorBlom 500 Grobot 500 Ziigmund 500 3 PhilIvey folds TomDwan calls 20 ViktorBlom raises 40 to 60 498 478 438 438 478
Output	4 Grobot calls 50 Ziigmund folds TomDwan folds dealing flop

[*Halaman ini sengaja dikosongkan*]

BIODATA PENULIS

Penulis bernama Jeremia Ronaldo Manurung, putra keempat dari 4 bersaudara yang lahir pada tanggal 22 Agustus 1998 di Palangkaraya, Kalimantan Tengah. Penulis melaksanakan pendidikan dasar di Sekolah Dasar Katolik Don Bosco Palangkaraya pada tahun 2004 hingga 2006, Sekolah Dasar Katolik Santa Clara Surabaya pada tahun 2006 hingga 2010, Sekolah Menengah Pertama Katolik Santa Clara Surabaya pada tahun 2010 hingga 2013, dan Sekolah Menengah Atas Kristen Petra 2 Surabaya pada tahun 2013 hingga 2016. Pada masa penulisan tugas akhir ini, penulis sedang menempuh studi S1 di Institut Teknologi Sepuluh Nopember, Surabaya di Departemen Teknik Informatika.

Selama masa studi, penulis memiliki ketertarikan dalam bidang rancang bangun aplikasi web, keamaan informasi dan jaringan, pemograman berorientasi objek, dan robotika.

Selain kesibukan akademik, penulis juga berperan aktif dalam beberapa kegiatan dan kepanitiaan. Beberapa diantaranya adalah staf dari divisi pengabdian profesi pada Himpunan Mahasiswa Teknologi Computer pada tahun 2017 dan menjadi staf dari divisi *National Logic Competition* pada kegiatan Schematics pada tahun 2017 dan 2018.

[*Halaman ini sengaja dikosongkan*]