The Indonesian Journal of Computer Science
www.ijcs.net
Volume 13, Issue 5, October 2024
https://doi.org/10.33022/ijcs.v13i5.4384

Implementasi Game Hangman Multiplayer Menggunakan Socket dan Multithreading Berbasis Protokol TCP

Mastura Diana Marieska[1], Harisatul Aulia[2], Muhammad Agung Hikmatullah[3], Rini Wahyuningtyas[4], Aras Maulana[5]
mastura.diana@ilkom.unsri.ac.id[1], haris.aulia404@gmail.com[2], 09021181722006@student.unsri.ac.id[3], 09021281722062@student.unsri.ac.id[4], 09021281722060@student.unsri.ac.id[5]
[1,2,3,4,5] Program Studi Teknik Informatika, Universitas Sriwijaya

Informasi Artikel

Diterima : 26 Agu 2024
Direvisi : 2 Okt 2024
Disetujui : 29 Okt 2024

Kata Kunci

Socket, Multithreading, Permainan multiplayer, Hangman

Abstrak

Permainan multiplayer cenderung lebih menarik dan dapat memberikan pengalaman yang lebih menyenangkan dibandingkan permainan single player. Permainan tebak kata atau yang biasa disebut dengan Hangman merupakan permainan yang umum dimainkan bersama-sama. Apabila pemain berada di tempat berbeda, diperlukan mekanisme koneksi yang reliable agar permainan dapat berjalan dengan baik. Pada penelitian ini diimplementasikan game Hangman dengan menggunakan socket dan multithreading. Socket digunakan sebagai mekanisme koneksi antara pemain dengan server. Protokol yang digunakan adalah protokol TCP yang bersifat reliable untuk menjaga integritas komunikasi antara server dan client. Agar permainan dapat melibatkan banyak pemain dalam satu waktu, maka digunakan multithreading. Pada server, multithreading digunakan dengan menugaskan thread khusus untuk tiap client yang terkoneksi. Dengan cara ini, komunikasi antara server dengan tiap client dapat dilakukan secara independen dan paralel. Dari hasil pengujian blackbox testing, seluruh fitur dari game Hangman berhasil diimplementasikan dan dapat digunakan dengan baik.

Keywords

Socket, Multihreading, Multiplayer game, Hangman

Abstract

Multiplayer games tend to be more engaging and can offer a more enjoyable experience compared to single-player games. Hangman, a word-guessing game, is commonly played in groups. When players are in different locations, a reliable connection mechanism is crucial to maintain uninterrupted gameplay. In this study, the Hangman game was implemented using socket programming and multithreading. Sockets were used as the connection mechanism between the players and the server. TCP protocol, which is reliable, was used in this implementation to maintain integrity of communication between clients and server. To allow multi players to participate simultaneously, multithreading was implemented. On the server side, each connected client was assigned a dedicated thread, allowing independent and parellel communication between server and each client. Blackbox testing result indicate that all features of the Hangman game were succesfully implemented and functioned properly.

A. Pendahuluan

Game *multiplayer* telah menjadi bagian penting dalam dunia *gaming* modern. Aspek sosial yang ditawarkan game *multiplayer* tidak hanya meningkatkan keseruan bermain, tapi juga membuat pemain lebih terlibat dalam pengalaman *gaming*. Game *multiplayer* secara alami mendorong interaksi antar pemain. Ketika bermain bersama secara *realtime*, pemain bisa saling terhubung dan menciptakan momen-momen seru yang sulit dilupakan. Hal ini tentu membuat pengalaman bermain jadi lebih menyenangkan dan memuaskan. Game *multiplayer* juga bisa jadi sarana belajar yang efektif. Melalui interaksi dengan pemain lain, kita bisa mendapat wawasan dan keterampilan baru. Ini menunjukkan pentingnya aspek sosial dalam meningkatkan pengalaman *gaming* secara keseluruhan [1]. Banyak game *multiplayer* sengaja dirancang untuk mendorong kerja sama tim. Strategi ini terbukti ampuh dalam meningkatkan kesenangan bermain dan mempererat hubungan antar pemain. Bahkan untuk pemain yang lebih tua, game *multiplayer* yang menekankan kolaborasi bisa membantu mempererat ikatan sosial [2].

Permainan Hangman merupakan sebuah permainan *multiplayer* klasik di mana satu pemain memikirkan sebuah kata, sementara pemain lain mencoba menebaknya dengan menyarankan huruf-huruf. Meskipun terlihat sederhana, permainan ini sangat popular dan telah terbukti efektif dalam berbagai konteks pendidikan. Sejumlah penelitian mengungkapkan dampak positif Hangman terhadap peningkatan kosakata para pelajar. Permainan ini terbukti dapat meningkatkan pemahaman, pengejaan, pengucapan, dan penggunaan kata-kata dalam konteks yang tepat [3]. Studi lain juga menunjukkan bahwa Hangman efektif dalam memperkuat kemampuan kosakata siswa, terutama yang masih muda [4]. Selain itu, Hangman juga dinilai mampu meningkatkan keterampilan berbahasa Inggris secara umum, khususnya terkait akuisisi kosakata [5]. Permainan Hangman juga diadopsi dalam strategi gamifikasi untuk mengajarkan algoritma dan konsep-konsep dasar ilmu komputer [6].

Evolusi teknologi dan strategi jaringan dalam pengembangan game *multiplayer* sangat dipengaruhi oleh tujuan untuk meningkatkan pengalaman pemain. Tantangan terbesarnya adalah memastikan bahwa semua pemain bisa memainkan permainan dengan lancar. Salah satu aspek utama yang mendorong evolusi ini adalah pemanfaatan pemrograman *socket* dan *multithreading* untuk memfasilitasi komunikasi yang lancar dan pemrosesan yang efisien dalam game *multiplayer* [7]. Dengan menerapkan pemrograman *socket*, game dapat membangun koneksi jaringan dan memungkinkan pertukaran data *realtime* antarpemain, yang berkontribusi pada pengalaman bermain game yang lebih mendalam dan menarik. Selain itu, penggunaan *multithreading* memungkinkan game untuk menangani beberapa tugas secara bersamaan, yang meningkatkan kinerja dan responsivitas dalam lingkungan *multiplayer* [8]. Pemakaian *socket* dan *multithreading* juga terbukti pada sistem lain seperti *online evaluation system* yang menunjukkan keefektifitasnya dalam memastikan keamanan dan efisiensi transmisi data antara *server* dan *client* [9]. Dalam hal pemrograman *socket*, *multithreading* perlu digunakan agar satu *server* dapat menerima koneksi dari beberapa *client* pada saat yang bersamaan.

Pada penelitian ini diimplementasikan aplikasi game Hangman *multiplayer* dengan menggunakan *multithreading* dan *socket programming* berbasis TCP. Keistimewaan dari aplikasi ini terletak pada pemanfaatan protokol dasar, yaitu TCP. Kebanyakan game sederhana *multiplayer* lainnya memanfaatkan protokol level aplikasi seperti HTTP. Terdapat dua bagian pada game Hangman ini, yaitu *server* sebagai pelaksana permainan dan *client* sebagai user yang berpartisipasi pada permainan. Jumlah user yang bermain dalam satu waktu dapat lebih dari satu. Dengan game Hangman *multiplayer* ini, didemonstrasikan bagaimana prinsip-prinsip dasar pemrograman jaringan dapat diterapkan untuk meningkatkan interaktivitas dan *engagement* dalam memainkan game sederhana.

B. Metode Penelitian

Pada bagian ini, dijelaskan metode dalam melaksanakan penelitian, meliputi skema koneksi, arsitektur, serta flowchart yang diimplementasikan pada game Hangman.

Skema Koneksi Client-Server Menggunakan TCP

Pada skema koneksi *client-server*, *client* dapat langsung mengirimkan permintaannya ke *server* untuk meminta layanan yang sesuai, dan kemudian *server* mengerjakannya dan segera mengembalikan data atau kode kesalahan sebagai respons kepada *client* [10]. Keunggulan skema ini adalah mendukung berbagai protokol komunikasi, yang dapat disesuaikan untuk memenuhi kebutuhan aplikasi tertentu, yang selanjutnya meningkatkan fleksibilitas dan kemampuan beradaptasinya [11]. Satu *socket* mendengarkan permintaan koneksi (ini menjadi tanggung jawab utama *server*) dan *socket* lainnya meminta koneksi (ini menjadi tanggung jawab utama client) untuk bergabung. Setelah koneksi terbentuk, *socket* tersebut dapat digunakan untuk mengirimkan data ke kedua arah [12].

Transmission Control Protocol (TCP) adalah protokol pada Internet, yang bertanggung jawab untuk memastikan komunikasi yang *reliable* antar perangkat dalam jaringan. Beroperasi pada lapisan *transport*, TCP dicirikan oleh sifatnya yang berorientasi pada koneksi - artinya, koneksi harus dibuat terlebih dahulu sebelum data dapat ditransmisikan. Proses ini dikenal sebagai "*three-way handshake*" TCP. Pertama, *client* memulai koneksi dengan mengirim paket SYN (Sinkronisasi) ke *server*. Lalu *server* merespons dengan paket SYN-ACK (Sinkronisasi-Pengakuan) untuk mengonfirmasi penerimaan SYN dan menyatakan kesiapannya untuk membuat koneksi. Terakhir, *client* mengirimkan paket ACK (Pengakuan) kembali ke *server*, mengonfirmasi pembentukan koneksi. Proses tiga langkah ini sangat penting untuk memastikan *client* dan *server* tersinkronkan dan siap untuk bertukar data. Dengan demikian, kehilangan data dapat dicegah dan komunikasi yang *reliable* dapat dijamin[13].

Penerapan *client-server* dan TCP di penelitian ini terletak pada implementasi *socket*. TCP menjadi *backbone* dari implentasi koneksi *socket*. Socket TCP digolongkan sebagai *stream socket*, memanfaatkan protokol TCP untuk memfasilitasi aliran data berkelanjutan antara *client* dan *server*. Hal ini penting untuk aplikasi yang membutuhkan pengiriman data yang terjamin, seperti server

web atau koneksi basis data. Sifat berorientasi koneksi dari TCP memungkinkan pembentukan saluran komunikasi yang stabil. Fitur ini sangat bermanfaat bagi aplikasi yang perlu mempertahankan status dan memastikan semua paket diterima dengan urutan yang tepat [14]. Hal ini menjadi keunggulan penggunaan TCP dalam implementasi *socket*.

Pada bahasa Java, implementasi *socket* TCP dilakukan dengan membuat objek dari kelas ServerSocket. Kelas ServerSocket merupakan salah satu kelas dari *library* java.net. Pembuatan objek ini menerima masukan berupa nomor *port*. Objek ServerSocket dibuat saat aplikasi server dimulai, yaitu pada prosedur start(). Setelah objek ServerSocket berhasil dibuat, maka selanjutnya akan melakukan listen untuk memantau apakah ada *client* yang ingin membuat koneksi. Berikut adalah code pemakaian TCP pada pembuatan objek ServerSocket.

```
public void start() {
      try {
          keepGoing = true;
          // Pembuatan socket server
          serverSocket = new ServerSocket(port);
          serverSocket.setReuseAddress(true);
          // Memulai thread untuk listen client
          new ListenForClient(serverSocket).start();
      } catch (IOException e) {
          String msg = sdf.format(new Date()) + " Exception on new
ServerSocket: " + e;
          display(msg);
      }
}
```

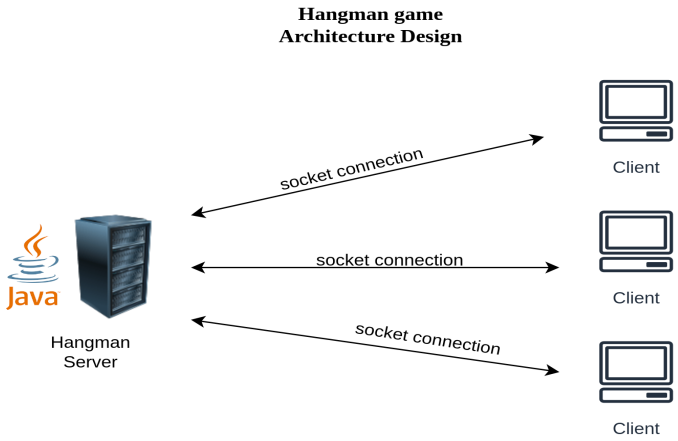
Mutithreading

Multithreading adalah teknik yang memungkinkan satu proses memiliki beberapa thread eksekusi sekaligus. Membuat pembagian suatu proses menjadi unit-unit kerja yang lebih kecil yang dapat dieksekusi secara bersamaan. Dalam konteks pengembangan game, prinsip *multithreading* yang efektif dapat meningkatkan kinerja mesin game secara signifikan dengan memanfaatkan komputasi paralel [15] Implementasi pemrograman *socket* tanpa *multithreading* menjadikan koneksi hanya bersifat *one to one* [16]. Apabila ada *client* yang sedang terkoneksi pada *server*, maka *client* lain yang ingin bergabung tidak akan diproses, melainkan akan ditempatkan pada antrian. Antrian baru dapat diproses apabila *client* sebelumnya telah memutuskan koneksi. Dalam hal penerapan pada game Hangman, tanpa *multithreading* maka game tidak dapat dimainkan secara *multiplayer*.

Pada bahasa Java, *multithreading* diimplementasikan melalui kelas *Thread* dan *Runnable*, yang memungkinkan eksekusi paralel tugas-tugas yang berbeda dalam satu aplikasi [17]. Pemakaian *multithreading* dilakukan pada aplikasi *client* maupun *server.* Di sisi *server*, *multithreading* digunakan untuk menerima koneksi dari banyak *client*, sehingga permainan dapat dilakukan secara *multiplayer*. Sementara pada *client*, *multithreading* digunakan untuk mendengarkan pesan dari *server* dan menampilkan status permainan terkini.

Arsitektur Game Hangman

Terdapat dua kompenen pada implementasi game Hangman, yaitu *Client* dan *Server*. *Client* merupakan komponen aplikasi dimana user berinteraksi dengan game, sedangkan s*erver* berperan sebagai pusat kendali untuk mengelola sesi-sesi permainan. *Server* bertanggung jawab untuk menjalankan logika game Hangman, termasuk menangani status permainan, mengatur pemilihan kata, melacak tebakan pemain, serta menentukan hasil akhir setiap putaran permainan. *Server* mengatur dan mempertahankan koneksi berkelanjutan dengan setiap *client*.



Gambar 1. Arsitektur Game Hangman

Seperti yang terlihat pada Gambar 1, koneksi *socket* merupakan saluran komunikasi dua arah yang memungkinkan para *client* berinteraksi dengan *server*. Ketika seorang *client* bergabung dalam permainan, mereka akan membuat koneksi *socket* dengan *server*. Melalui koneksi ini, *client* dapat mengirimkan tebakan, meminta informasi status permainan terkini, serta menerima pembaruan dari *server*. Sebaliknya, *server* juga dapat mengirimkan pembaruan permainan, tanggapan atas permintaan *client*, serta informasi relevan lainnya kepada semua *client* yang terhubung.

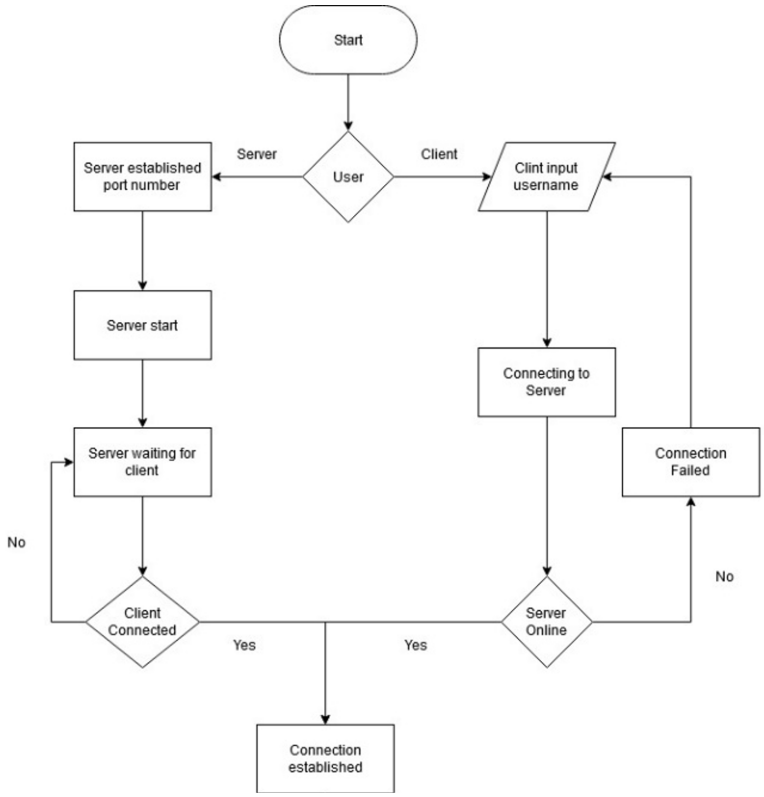
Dengan menggunakan arsitektur *client-server* seperti ini, game Hangman dapat melayani banyak *client* secara bersamaan. Setiap *client* terhubung ke *server* secara independen, sehingga mereka dapat berpartisipasi dalam permainan secara konkuren. Desain ini memungkinkan adanya fitur *realtime multiplayer*, dimana pada satu waktu beberapa pemain dapat saling bersaing dalam sesi permainan Hangman yang sama.

Prosedur Pembuatan Koneksi

Sesuai dengan sifat skema *client-server*, untuk membuat koneksi pada game Hangman, *server* harus aktif terlebih dahulu mengaktifkan *socket* pada suatu *IP Address* dan nomor *port* tertentu [10]. Nomor *port* yang digunakan merupakan nomor *port* khusus yang dipilih, karena nomor ini yang kemudian diinformasikan kepada *client* game Hangman. Setelah *socket* aktif, *server* kemudian menunggu sampai ada *client* yang secara aktif menghubungi *server*.

Client yang ingin bergabung pada game Hangman terlebih dahulu harus memasukkan *username*. Kemudian *client* memasukkan data *IP Address* serta *port server* yang dituju. Tidak ada pemilihan nomor *port* untuk koneksi dari *client*, sehingga dapat menggunakan nomor *port* acak yang diberikan oleh sistem operasi. Apabila saat itu *server* tidak aktif, maka pembuatan koneksi gagal. Apabila *server* telah dalam keadaan aktif, maka pembuatan koneksi berhasil dan *client* siap bergabung permainan.

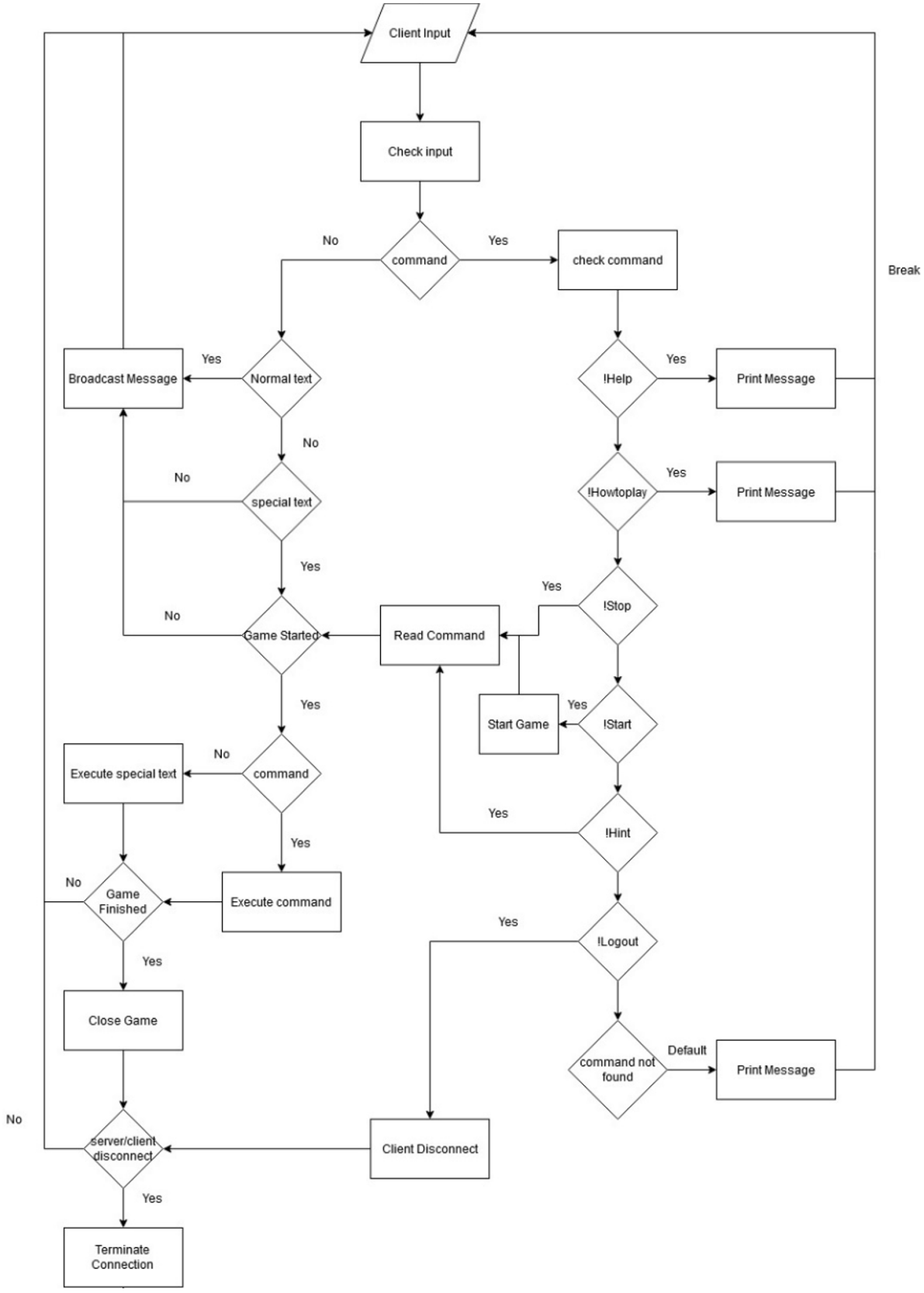
Server akan mencatat *username* dari *client* yang berhasil membuat koneksi. Selanjutnya *server* akan membuat satu *thread* khusus yang ditugaskan untuk *handling* koneksi dari *client* tersebut. Apabila saat itu ada sesi permainan yang sedang aktif, maka *client* akan dikirimi informasi mengenai kata menjadi soal permainan dan *server* mengirim informasi *broadcast* kepada *client* lain yang sedang terkoneksi pada sesi tersebut mengenai bergabungnya *user* baru. *Flowchart* pembuatan koneksi antara *client-server* dapat dilihat pada Gambar 2 berikut ini.



Gambar 2. Flowchart Pembuatan Koneksi

Implementasi Hangman Server

Server bertugas mengendalikan jalannya game dan merespon setiap input yang dikirimkan user. *Multithreading* pada *server* diimplementasikan dengan membuat *thread* untuk *handling* koneksi tiap client. *Thread* utama pada *server* berfungsi untuk mengatur jalannya/logika dari game Hangman. *Flowchart* implementasi Hangman Server dapat dilihat pada Gambar 3.



Gambar 3. Flowchart Hangman Server

Terdapat 2 jenis input yang dapat dikirimkan oleh user, yaitu *command* dan *non command.* Input berupa *command* diawali dengan tanda "!". Ada 6 jenis *command* yang dapat dikirimkan oleh user, yaitu :

!Help	: Menampilkan daftar *command* yang tersedia
!Howtoplay	: Menampilkan aturan permainan
!Start	: Memulai permainan
!Stop	: Menghentikan permainan
!Hint	: Memberikan petunjuk dari kata yang ditebak
!Logout	: Keluar dari permainan dan memutus koneksi

Saat user menjalankan *command* !Start, maka *server* akan memilih secara acak kata yang akan ditebak dari list kata pada file words.txt. Kata ini yang kemudian dijadikan soal pada sesi permainan tersebut. *Server* akan *broadcast* ke seluruh user berupa jumlah kata dan jumlah huruf yang harus ditebak. Apabila ada user yang menjalankan *command* !Stop, maka *server* akan menghentikan permainan dan *broadcast* informasi berhentinya permainan ke seluruh user.

Apabila input yang dikirimkan user berupa *non command*, maka *server* akan mengecek apakah input tersebut berupa tebakan jawaban dari permainan tersebut atau berupa pesan chat untuk seluruh user. Pesen chat untuk seluruh user akan langsung dibroadcast oleh *server*, dengan terlebih dahulu menambahkan *username* dari user yang mengirim pesan. Tebakan jawaban dari permainan yang berlangsung diawali dengan tanda "~". Saat user mengirimkan tebakan jawaban, *server* akan mengecek apakah ada sesi permainan yang sedang berlangsung. Jika ada sesi permainan yang berlangsung, maka *server* akan mengecek soal pada sesi tersebut.

Jika user menebak dengan benar, maka *server* akan mengirim *feedback* berupa ucapan selamat kepada user yang menebak. Kepada user lainnya, *server* mengirimkan pesan berupa informasi permainan berakhir beserta nama user yang berhasil menebak. Jika tebakan user salah, maka "nyawa" user tersebut akan dikurangi. Apabila "nyawa" seluruh user telah habis dan belum ada yang memberikan tebakan benar, maka *server* akan *broadcast* pesan berisi jawaban pertanyaan dan informasi tidak ada pemenang dari sesi permainan tersebut.

Implementasi Hangman Client

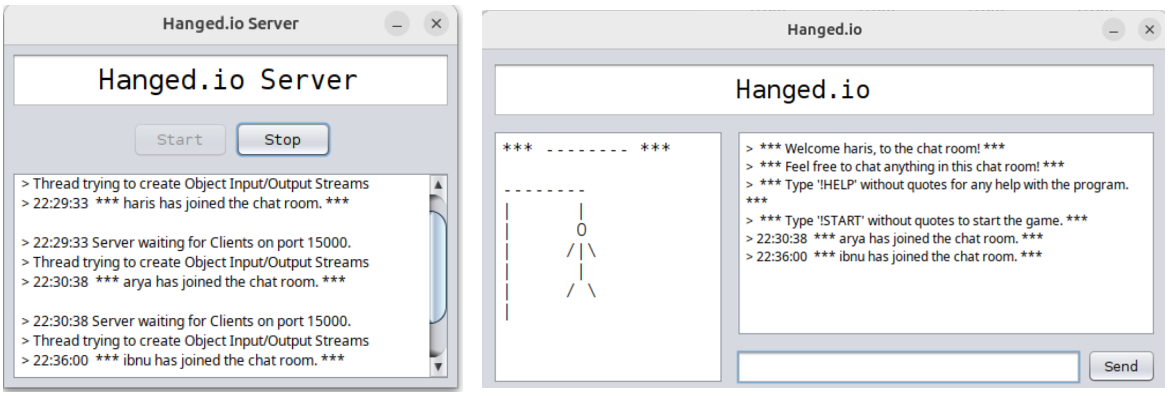
Implementasi *client* lebih sederhana dari *server*. Hangman Client terdiri atas 2 *thread*, yaitu *thread* utama untuk interaksi user dan *thread* untuk mendengarkan pesan dari *server*. Tampilan utama berisi soal yang harus ditebak dan gambaran status Hangman pada saat tersebut. Dari gambaran status Hangman, user dapat mengetahui jumlah sisa "nyawa" yang dimilikinya. User dapat mengirimkan pesan berupa *command* maupun *non command* kepada *server*. Setiap ada pesan dari *server*, maka tampilan utama akan diperbarui sesuai isi pesan tersebut.

C. Hasil dan Pembahasan

Game Hangman diimplementasikan menggunakan bahasa Java dan memakai kelas java.net untuk pembuatan *socket*. Pada bagian ini dijelaskan tampilan layar dari Game Hangman yang dibuat pada penelitian ini dan hasil dari *blackbox testing*.

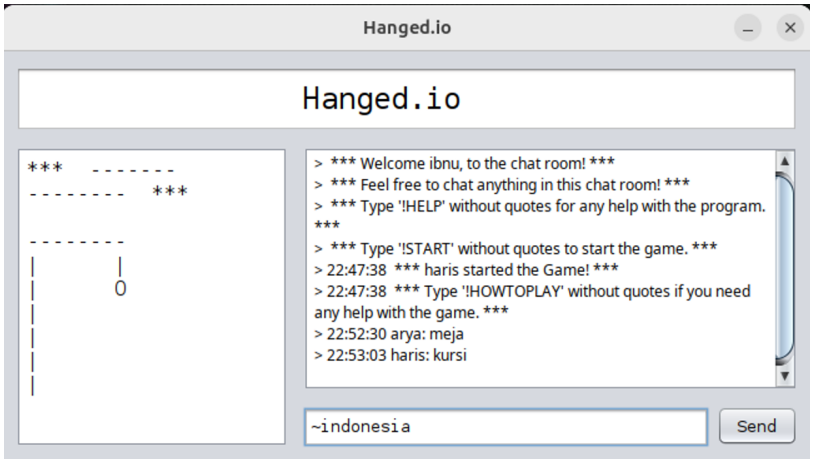
Tampilan Game Hangman

Game ini dimulai dengan mengaktifkan *server* pada *IP Address* dan nomor *port* yang telah ditentukan. Setelah *server* aktif, selanjutnya berjalan prosedur pembuatan koneksi sesuai *flowchart* yang telah dirancang. *Client* akan menginisiasi koneksi dengan terlebih dahulu memasukkan data *username*. Setiap kali ada user baru yang berhasil terkoneksi, *server* akan menampilkan pesan pada tampilan utamanya dan mengirimkan pesan *broadcast* kepada semua user yang terkoneksi saat itu. Gambar 4 menunjukkan tampilan *server* dan *client* ketika ada user baru yang terkoneksi.



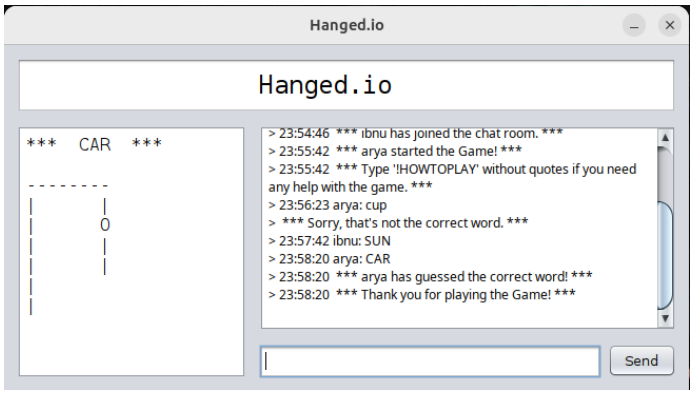
Gambar 4. Tampilan *client* berhasil terkoneksi

Selanjutnya *server* akan menunggu sampai salah satu user mengirim perintah !START untuk memulai permainan. Jika salah satu user memulai permainan, maka *server* akan memilih soal berupa kumpulan kata dari word.txt. Soal ini kemudian dikirimkan ke seluruh user yang aktif pada sesi tersebut. User dapat memulai menebak soal yang telah dikirimkan oleh *server*. Menebak soal dilakukan dengan cara mengirim pesan yang diawali dengan tanda"~". Apabila tebakan user salah, maka tampilan gambar status Hangman akan berubah sesuai dengan jumlah salah tebak yang telah dilakukan oleh user. Gambar 5 adalah tampilan *client* saat akan menebak soal.

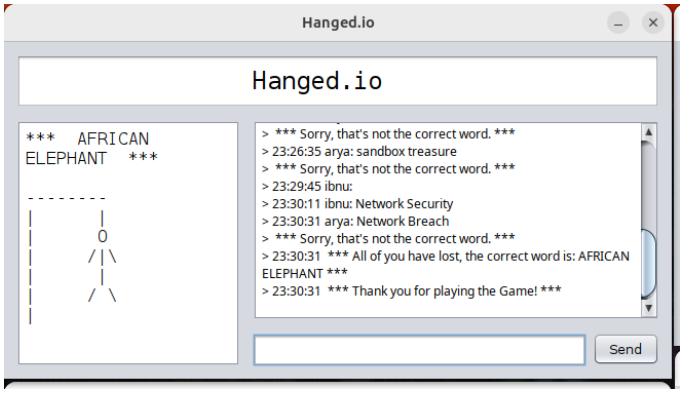


Gambar 5. Tampilan *client* menebak kata

Apabila ada user yang berhasil menebak soal, maka user tersebut memenangkan permainan. *Server* mengirimkan ucapan selamat kepada user yang menang dan mengirimkan pesan informasi kepada user lainnya mengenai berakhirnya permainan. Apabila sampai dengan kesempatan menebak habis dan tidak ada user yang berhasil menebak dengan benar, maka *server* mengirim pesan ke seluruh user mengenai jawaban yang benar dan berakhirnya permainan. Gambar 6 menunjukkan tampilan *client* ketika berhasil memenangkan permainan dan Gambar 7 menunjukkan tampilan *client* ketika tidak ada yang berhasil menang.



Gambar 6. Tampilan client memenangkan permainan



Gambar 7. Tampilan client tidak ada yang menang

Setelah berakhirnya satu sesi permainan, user dapat memilih untuk bermain lagi dengan mengirim *command* !START atau dapat meninggalkan game dengan mengirim *command* !LOGOUT. Apabila ada user yang meninggalkan permainan, maka *server* akan memutus koneksi terhadap user tersebut dan mengirimkan informasi kepada user lainnya bahwa user tersebut telah meninggalkan permainan.

Blackbox Testing

Untuk memastikan seluruh fitur pada game Hangman berjalan dengan baik, dilakukan test dengan metode *blackbox testing*. *Blackbox testing* merupakan metode pengujian yang fokus pada fungsional aplikasi tanpa melihat detil ke kode program. Kelebihan dari *blackbox testing* adalah dapat melakukan evaluasi dari sudut pandang user [18]. Terdapat 6 fitur utama aplikasi yang diujikan, yaitu mengaktifkan *server*, *client login*, memulai permainan, menebak kata, game selesai,

dan client *logout*. Hasil *blackbox testing* dapat dilihat pada Table 1. Dari hasi pengujian, didapatkan bahwa seluruh fitur berjalan dengan baik.

Tabel 1. Hasil Blackbox Testing

No	Fitur	Langkah Pengujian	Hasil
1	Mengaktifkan server	Pada tampilan server, klik start. Lalu, server akan berjalan pada port yang tertera	Berhasil
2	Client login	Pada kolom yang disediakan, masukan username yang diinginkan, sampai ada pesan koneksi berhasil	Berhasil
3	Memulai permainan	Pada kolom *chat,* user mengetik perintah !START, muncul pesan permainan dimulai dan soal ditampilkan	Berhasil
4	Menebak kata	Pada kolom *chat,* user mengetik perintah ~kata_yang _ditebak. Misal ~meja. Tampil respon dari server sesuai kebenaran tebakan kata	Berhasil
5	Game selesai	Permainan selesai dengan kemenangan apabila peserta bisa menebak kata. Namun, pemain dinyatakan kalah apabila tidak dapat menebak kata yang dimaksud.	Berhasil
6	Client logout	Pada kolom *chat,* user mengetik perintah !LOGOUT. Koneksi ke server berhasil diakhiri.	Berhasil

D. Simpulan

Penelitian ini telah berhasil mendemonstrasikan implementasi *socket programming* dan *multithreading* pada Java melalui pengembangan sebuah game Hangman yang dilengkapi dengan fitur *chat*. Implementasi ini menunjukkan bagaimana teknologi *socket* dapat digunakan untuk memungkinkan komunikasi *realtime* antara *client* dan *server*, sementara *multithreading* memungkinkan *server* untuk menangani banyak koneksi *client* secara bersamaan. Permainan yang dikembangkan juga telah diuji menggunakan metode *blackbox testing* untuk memastikan semua fungsi dan fitur berjalan sesuai dengan yang diharapkan. Hasil pengujian menunjukkan bahwa aplikasi ini bekerja dengan baik di berbagai skenario, termasuk ketika banyak user terhubung secara bersamaan. Hasil ini diharapkan dapat menjadi referensi bagi pengembang lain yang ingin mengimplementasikan solusi serupa dalam aplikasi mereka.

E. Referensi

[1] I. Iacovides, A. L. Cox, A. Avakian, and T. Knoll, “Player strategies: Achieving breakthroughs and progressing in single-player and cooperative games,” *CHI PLAY 2014 - Proceedings of the 2014 Annual Symposium on Computer-Human Interaction in Play*, pp. 131–140, 2014, doi: 10.1145/2658537.2658697.

[2] S. Osmanovic and L. Pecchioni, “Beyond Entertainment,” *Games Cult*, vol. 11, no. 1–2, pp. 130–149, 2016, doi: 10.1177/1555412015602819.

[3] I. N. Weda Dharmayasa, “Implementing the Hangman Game in Teaching English Vocabulary to Elementary School Students,” *Jurnal Pendidikan Bahasa Inggris undiksha*, vol. 10, no. 3, pp. 291–297, 2023, doi: 10.23887/jpbi.v10i3.58140.

[4] M. Munikasari, S. Sudarsono, and D. Riyanti, "the Effectiveness of Using Hangman Game To Strengthen Young Learners' Vocabulary," *Journal of English Education Program*, vol. 2, no. 1, pp. 57–65, 2021, doi: 10.26418/jeep.v2i1.43328.

[5] N. Nabilah, "Using Hangman Game Application For The EFL Classroom: It Efficacy for Learners to Master Vocabulary," *Proceeding of International Conference on Language Pedagogy (ICOLP)*, vol. 1, no. 1, pp. 136–143, 2021, doi: 10.24036/icolp.v1i1.31.

[6] L. R. Begosso, L. C. Begosso, D. Cunha, J. V. Pinto, L. Lemos, and M. Nunes, "The Use of Gamification for Teaching Algorithms," *Communication Papers of the 2018 Federated Conference on Computer Science and Information Systems*, vol. 17, pp. 225–231, 2018, doi: 10.15439/2018f165.

[7] I. P. A. E. Pratama and I. W. G. Arisna, "Fighter Plane Online Game Based on Design Science Research Methodology Using Socket.Io and Nodejs," *Indonesian Journal of Engineering and Science*, vol. 3, no. 2, pp. 029–037, 2022, doi: 10.51630/ijes.v3i2.38.

[8] M. Samak and M. K. Ramanathan, "Multithreaded test synthesis for deadlock detection," *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA*, pp. 473–489, 2014, doi: 10.1145/2660193.2660238.

[9] P. Jiang, K. Yan, H. Chen, and H. Sun, "Building of Online Evaluation System Based on Socket Protocol," *Computer Science and Information Systems*, vol. 19, no. 1, pp. 185–204, 2022, doi: 10.2298/CSIS210201047J.

[10] T. Duong-Ba and T. Nguyen, "Distributed client-server assignment," *Proceedings - Conference on Local Computer Networks, LCN*, no. October 2012, pp. 296–299, 2012, doi: 10.1109/LCN.2012.6423633.

[11] A. Hasibuan and E. Dalimunthe, "Implementasi Metode Client Server pada Penerapan Aplikasi Simulasi Ujian Akhir," *Jurnal Informatika Universitas Pamulang*, vol. 5, no. 2, p. 152, 2020, doi: 10.32493/informatika.v5i2.5614.

[12] P. Deitel and H. Deitel, *Java: How To Program, Early Objects*, 11th ed. Pearson, 2018.

[13] G. Fairhurst, B. Trammell, and M. Kuehlewind, "RFC 8095: Services Provided by IETF Transport Protocols and Congestion Control Mechanisms," 2017, doi: https://doi.org/10.17487/RFC8095.

[14] M. VM, "TCP Concurrent Echo Program using Fork and Thread," *International Journal on Recent and Innovation Trends in Computing and Communication*, vol. 3, no. 4, pp. 2225–2229, 2015, doi: 10.17762/ijritcc2321-8169.1504100.

[15] T. Russo, R. R. Ribeiro, A. Araghi, R. de M. Lameiras, J. Granja, and M. Azenha, "Continuous Monitoring of Elastic Modulus of Mortars Using a Single-Board Computer and Cost-Effective Components," *Buildings*, vol. 13, no. 5, 2023, doi: 10.3390/buildings13051117.

[16] R. Rustagi and V. Kumar, "Experiential Learning of Networking Technologies: Evolution of Socket Programming – Part II," *Advanced Computing and Communications*, vol. 4, no. 1, 2020, doi: 10.34048/ACC.2020.1.F4.

[17] oracle, "javadoc," docs.oracle.com. [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/lang/Thread.html

[18] D. I. Pirdaus and R. A. Hidayana, “Analysis Testing Black Box and White Box on Application To-Do List Based Web,” *International Journal of Mathematics, Statistics, and Computing*, vol. 2, no. 2, pp. 68–75, 2024.