Universidade de Brasília – UnB
Faculdade UnB Gama – FGA
Engenharia de Software

Evolução e Manutenção do SimpleMUD

Autor: Iuri de Souza Severo Alves
Orientador: Prof. Dr. Edson Alves da Costa Júnior

Brasília, DF
2023

Iuri de Souza Severo Alves

Evolução e Manutenção do SimpleMUD

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Orientador: Prof. Dr. Edson Alves da Costa Júnior

Brasília, DF

2023

Iuri de Souza Severo Alves
Evolução e Manutenção do SimpleMUD/ Iuri de Souza Severo Alves. – Brasília, DF, 2023-
145 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Edson Alves da Costa Júnior

Trabalho de Conclusão de Curso – Universidade de Brasília – UnB
Faculdade UnB Gama – FGA , 2023.

1. Multi-user Dungeon. 2. Evolução e Manutenção de Software. I. Prof. Dr. Edson Alves da Costa Júnior. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Evolução e Manutenção do SimpleMUD

CDU 02:141:005.6

Iuri de Souza Severo Alves

Evolução e Manutenção do SimpleMUD

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, 11 de julho de 2023:

Prof. Dr. Edson Alves da Costa Júnior
Orientador

Prof. Dr. Maurício Serrano
Convidado 1

Prof. Dr. Renato Coral Sampaio
Convidado 2

Brasília, DF
2023

Agradecimentos

Gostaria de expressar minha sincera gratidão aos meus pais, que sempre me apoiaram, ofereceram suporte em todas as minhas escolhas e nunca mediram esforços para suprir todas as minhas necessidades. À minha mãe, Diene Nobre, que trabalhou muito para que eu pudesse ter a oportunidade de concluir minha graduação e sempre supriu todas as necessidades que tive durante esses anos. Da mesma forma, meu pai, Ricardo Severo, que sempre esteve presente, dando apoio e me incentivando durante essa jornada.

Agradeço aos meus irmãos, primos e outros familiares, que me acompanharam durante todo esse processo. Em especial, meu primo, Lucas Severo, que me inspirou a escolher esse curso e me ajudou diversas vezes durante meus semestres.

Agradeço à minha melhor amiga e namorada, Larissa Costa, que esteve ao meu lado durante quase todo meu tempo na faculdade, me apoiando, ajudando, aguentando minhas chatices. Mostrou-me que a faculdade é mais do que só a sala de aula, o que me proporcionou diversas experiências que foram essenciais para que eu me tornasse quem sou hoje. Obrigado por tudo, meu bem, te amo.

Agradeço também aos meus amigos, aqueles que trouxe do ensino médio e aqueles que conheci durante a graduação, que compartilharam vários momentos comigo e contribuíram para o meu crescimento como pessoa. Especialmente meus amigos Joaquim Virgilio, Lucas Manoel, Camila Rodrigues, Daniel Correia e João Pedro Guedes e Hugo Sobral.

Por fim, expresso meus agradecimentos a todo o corpo docente e demais funcionários da FGA, cujo os serviços tornaram possível a realização da minha graduação. Em especial, sou grato ao professor Edson Alves, que me orientou na elaboração deste trabalho e compartilhou seu vasto conhecimento, proporcionando-me uma valiosa oportunidade de aprendizado. Também agradeço aos professores Maurício Serrano e Renato Sampaio, que aceitaram participar da banca de avaliação deste trabalho, contribuindo significativamente para o seu resultado final.

"Virtual worlds are places where the imaginary meets the real."
(Richard Bartle)

Resumo

Multi-User Dungeon (MUD) é um gênero de jogos que se originou na década de 1980 com o jogo MUD1, que deu nome ao gênero. Os MUDs consistem em mundos virtuais baseados em texto que podem ser explorados por múltiplos jogadores em um ambiente *online*. Embora tenham sido amplamente explorados até o início da década de 2000, com o avanço tecnológico foram substituídos por jogos gráficos e serviram como base para o desenvolvimento dos *Massively Multiplayer Online Role-Playing Games* (MMORPGs). No entanto, os códigos utilizados para desenvolvê-los ainda são valiosos para estudantes das áreas de tecnologia, que podem utilizá-los para explorar assuntos multidisciplinares. Dentre os diversos MUDs existentes, dois foram analisados neste trabalho: o Dyrt, desenvolvido por Valentin Popescu e Katie Mowry, e o SimpleMUD, desenvolvido por Ron Penton. Após a análise, o último foi escolhido como base de código para o trabalho em questão, tendo como objetivo evoluí-lo de acordo com os padrões atuais da tecnologia utilizada em seu desenvolvimento, o C++. Dessa forma, foi possível disponibilizá-lo para que outros estudantes possam explorá-lo. A evolução do código foi adaptativa, visando atualizá-lo sem alterar suas funcionalidades, seguindo as diretrizes principais de C++ estabelecidas por Bjarne Stroustrup e Herb Sutter, bem como outros padrões encontrados na comunidade. O resultado alcançado por meio do desenvolvimento deste trabalho consiste em uma versão funcional do SimpleMUD, disponibilizada em um ambiente conteinerizado, acompanhada por testes unitários e análise estática do código, juntamente com um fluxo de trabalho de integração contínua. Além disso, foi elaborado e disponibilizado um *backlog* do produto, contendo atividades de aprimoramento tanto para o MUD quanto para o repositório, servindo como um guia para orientar outros estudantes em trabalhos futuros.

Palavras-chave: Multi-user Dungeon. MUD. Evolução e Manutenção de Software. C++. MMORPG.

Abstract

Multi-User Dungeon (MUD) is a genre of games that originated in the 1980s with the game MUD1, which gave the genre its name. MUDs consist of text-based virtual worlds that can be explored by multiple players in an online environment. Although they were widely explored until the early 2000s, they were replaced by graphical games with technological advancements and served as a foundation for the development of Massively Multiplayer Online Role-Playing Games (MMORPGs). However, the source codes that were the result of their development are still valuable for students in the technology field, who can use them to explore interdisciplinary subjects. Among the various existing MUDs, two were analyzed in this work: Dyrt, developed by Valentin Popescu and Katie Mowry, and SimpleMUD, developed by Ron Penton. After the analysis, the latter was chosen as the codebase for the work at hand, aiming to evolve it according to the current standards of the technology used in its development, C++. With that in mind, these enhancements will be made available for other students to explore. The code evolution was adaptive, seeking to update it without altering its functionalities, following the main C++ guidelines established by Bjarne Stroustrup and Herb Sutter, as well as other patterns found in the community. The result achieved through the development of this work consists of a functional version of SimpleMUD, provided in a containerized environment, accompanied by unit tests and static code analysis, along with a continuous integration workflow. Additionally, a product backlog was elaborated and made available, containing enhancement activities for both the MUD and the repository, serving as a guide to guide other students in future work.

Key-words: Multi-user Dungeon. MUD. Software Evolution and Maintenance. C++. MMORPG.

Lista de ilustrações

Lista de tabelas

Lista de quadros

Lista de códigos

Lista de abreviaturas e siglas

COTS	*Commercial off-the-shelf*
DBM	*Database Manager*
GCC	GNU *Compiler Collection*
GDB	GNU *Project Debugger*
GDBM	GNU *Database Manager*
IEEE	Instituto de Engenheiros Eletricistas e Eletrônicos
IP	*Internet Protocol*
ISO	*International Organization for Standardization*
MMORPG	*Massively Multiplayer Online Role-Playing Game*
MUD	*Multi User Dungeon*
NDBM	*New Database Manager*
STL	*Standard Template Library*
TCC	Trabalho de Conclusão de Curso
TCP	*Transmission Control Protocol*
UML	*Unified Modeling Language*
UDP	*User Datagram Protocol*
UnB	Universidade de Brasília

Sumário

1 Introdução

Multi-user Dungeon (MUD) é um gênero de jogos que surgiu em meados dos anos 80 e deu origem a sofwares complexos, diversificados e adaptáveis. Os MUDs foram os primeiros mundos virtuais a fazerem sucesso e serviram como base para o desenvolvimento de diversos outros programas, como MMORPGs (*Massively Multiplayer Online Role-Playing Game*) e alguns simuladores (BARTLE, 2004; PENTON, 2003).

Devido à sua natureza, os MUDs requerem o uso de tecnologias distintas para sua operação e abrangem uma ampla gama de disciplinas da Engenharia de Software. O componente *multiplayer* exige habilidades na programação de redes, enquanto a gestão dos jogadores necessita de conhecimentos em bancos de dados ou manipulação de arquivos. Além disso, é indispensável ter familiaridade com *multithreading*, estruturas de dados e outros aspectos para o desenvolvimento do jogo propriamente dito.

Com a evolução das capacidades gráficas dos computadores, tanto os jogadores quanto os desenvolvedores migraram dos MUDs baseados em texto para mundos virtuais com gráficos bidimensionais e tridimensionais (BARTLE, 2004), porém a forma como o software era estruturado se manteve. Até o ano de 2003, quase todos os jogos estilo mundo virtual possuem estruturas similares internamente (PENTON, 2003).

1.1 Justificativa

Apesar dos MUDs terem se tornado ultrapassados, o código base desses jogos ainda pode ser considerado material valioso para fins educacionais. Disponibilizar um MUD em funcionamento em uma instituição acadêmica permitiria aos alunos a oportunidade de jogar, testar e analisar o jogo, a fim de compreender mais profundamente seu desenvolvimento, arquitetura e outros aspectos relevantes.

Além disso, também seria possível evoluir o MUD, de maneira que ele se torne mais seguro, escalável e até mesmo mais lúdico. Em sua conclusão, Penton (2003) discute as possíveis melhorias que podem ser feitas nos MUDs que ele desenvolveu e destaca em vários momentos como o código pode ser personalizado pelo desenvolvedor para que o jogo possua um estilo único.

Existem muitos aspectos que precisam ser considerados na administração de um MUD, e gerenciá-los pode ser uma excelente oportunidade de desenvolvimento para estudantes de cursos relacionados à tecnologia. É importante planejar as questões relacionadas à implantação do jogo, incluindo a conexão com a internet e a garantia da segurança dessa conexão com os jogadores, a implementação de novas funcionalidades, a realização de ma-

nutenção e correção de erros, e assim por diante. No entanto, para que isso seja possível, é necessário que o software tenha sido atualizado e esteja funcionando com as tecnologias atuais.

> *This book isn't the be-all and end-all of MUDs. I've only scratched the surface of what you can do... but there's absolutely no reason why MUD technologies can't be extended infinitely*(PENTON, 2003) [1].

1.2 Objetivos

Este trabalho tem como objetivo principal tornar a base de códigos de um MUD implantável, para que outros estudantes possam explorá-lo e evoluí-lo conforme desejarem. Para viabilizar este objetivo, é necessário atingir os seguintes objetivos específicos:

- selecionar uma base de códigos de um MUD;
- analisar a base selecionada;
- atualizá-la para que ela funcione utilizando tecnologias atuais;
- criar um ambiente de desenvolvimento onde seja possível realizar um desenvolvimento contínuo da base;
- realizar manutenções necessárias.

Deve-se destacar que este trabalho foi desenvolvido em paralelo com um projeto de iniciação científica, que tem como orientador o professor Maurício Serrano, cujo objetivo é modernizar e implementar um MUD utilizando uma base de dados relacional. A iniciação científica também foi desenvolvida por mim, sem envolvimento de outros estudantes.

1.3 Estrutura do Trabalho

O trabalho em questão está estruturado em cinco capítulos. O primeiro capítulo, Introdução, traz uma visão geral sobre o tema abordado, além de expor a motivação e os objetivos desta pesquisa, finalizando com uma explicação breve sobre sua estrutura. Já o segundo capítulo, intitulado Fundamentação Teórica, apresenta os conceitos essenciais para compreender o trabalho, com enfoque na história e desenvolvimento dos MUDs, bem como na evolução e manutenção de software. O terceiro capítulo, Metodologia, exibe o planejamento metodológico utilizado para a evolução e manutenção da base de códigos

[1] Este livro não é o princípio e o fim de todos os MUDs. Eu apenas arranhei a superfície do que você pode fazer... mas não há absolutamente nenhuma razão para que as tecnologias MUD não possam ser estendidas infinitamente (tradução livre).

selecionada, assim como o planejamento da metodologia que será adotada para a próxima etapa do desenvolvimento desse trabalho. O quarto capítulo, Resultados Obtidos, ilustra os resultados obtidos durante o desenvolvimento desta monografia. Por fim, o capítulo Considerações Finais apresenta uma breve conclusão a respeito do trabalho, assim como possibilidades de trabalhos futuros.

2 Fundamentação Teórica

Neste capítulo, é apresentada a fundamentação teórica necessária para compreender o trabalho em questão. Dividido em duas seções, a primeira aborda a história e o desenvolvimento dos Multi-User Dungeons (MUDs), desde sua origem em 1978, até as variantes que se proliferaram ao longo dos anos. A segunda seção trata sobre a evolução e manutenção de software, com ênfase nas definições sugeridas por Chapin et al. (2001).

2.1 Multi-user Dungeons

A relação entre *Multi-user Dungeons* (MUDs) e mundos virtuais surge desde antes da internet, na época de 1970, quando o primeiro MUD foi lançado. Os mundos virtuais costumam ser chamados de MUDs porque MUD foi o nome do primeiro mundo virtual a prosperar. No entanto, enquanto mundos virtuais são ambientes considerados autocontidos, MUDs são, principalmente, mundos virtuais baseados em texto (BARTLE, 2004; BARTLE, 2015). Em outras palavras, todo MUD é um mundo virtual, porém nem todo mundo virtual é um MUD.

De acordo com o artigo "Multi-User Dungeons (MUDs)" de Bartle (BARTLE, 2015), há oito critérios estabelecidos para determinar se um mundo virtual é considerado um MUD, os quais são:

1. Ele opera por meio de um conjunto de regras automáticas que permitem aos jogadores efetuar alterações no ambiente virtual; essas regras são conhecidas como sua física. Salas de bate-papo e Dungeons & Dragons não são MUDs porque não possuem física.

2. Os jogadores interagem com o mundo através do canal de um único objeto que eles controlam sozinhos e que está "dentro" do mundo; este objeto é seu personagem. Jogos de estratégia não são MUDs porque os jogadores não agem por meio de um personagem.

3. A interação ocorre em tempo real. Jogos baseados em turnos não são MUDs porque o tempo entre os movimentos é perceptível.

4. Vários jogadores podem compartilhar o mesmo ambiente virtual, no qual qualquer um deles pode fazer alterações. Jogos para um jogador não são MUDs porque o mundo não é compartilhado

5. As mudanças no mundo devem ser capazes de durar mesmo que todos jogadores encerrem suas sessões; isso é conhecido como persistência. Jogos de tiro em primeira pessoa não são MUDs porque não são persistentes
6. O mundo não deve ser o mundo real. A realidade não é um MUD porque é o mundo real.
7. Tem mecânica de jogo codificada em sua física. O Second Life não é um MUD porque não tem jogabilidade.
8. Tem uma interface principalmente textual. World of Warcraft não é um MUD porque é gráfico.

Atualmente, os critérios 1 a 6 são abrangidos pelo termo mundo virtual, que é imparcial em relação aos critérios 7 e 8 (BARTLE, 2015).

No aspecto do desenvolvimento, além de todo o trabalho de comunicação, geralmente há três partes em uma *game engine* de MUD: a física, a lógica e a de dados. A parte física é a que controla a existência e movimentação de itens, personagens, salas, etc. Basicamente tudo que pode ser representado como um objeto físico, os quais normalmente são definidos como entidades. A parte lógica é responsável pelo que acontece na camada física, como o que um personagem faz quando atacado ou quando recebe um item. Personagens precisam tomar decisões, items precisam realizar ações quando são equipados, entre outras coisas. Por fim, a parte de dados é a que define todas entidades físicas do jogo, por exemplo, sempre que você carrega um mapa na camada física, o mapa é carregado da camada de dados.

A forma como essas três partes são desenvolvidas nos MUDs variou bastante com tempo. Inicialmente elas eram armazenadas dentro do código, o que dificultava alterações no jogo, uma vez que era necessário ir no código, modificá-lo, recompilá-lo, reiniciar o MUD e o executar novamente.

A camada de dados foi a primeira a se tornar flexível. Salvar os dados em arquivos possibilitava alterar as informações das entidades sem precisar reiniciar o MUD. Após isso veio a flexibilização da camada lógica, que, essencialmente, passou a permitir a alteração de como todo o jogo funciona enquanto ele está rodando. A possibilidade de uma camada física flexível também existe, porém não é algo que se popularizou, visto que a necessidade de uma camada física flexível não existe, uma vez que a inserção de novos tipos de entidade é algo raro de acontecer (PENTON, 2003).

2.1.1 Evolução dos mundos virtuais

Bartle, em seu livro “Design Virtual Words” (BARTLE, 2004), divide a evolução dos mundos virtuais em cinco eras, que vão desde 1978 até 2004, quando o livro foi

publicado.

Durante a Primeira Era, que aconteceu entre 1978 e 1985, foi feito o desenvolvimento do primeiro jogo virtual chamado MUD (*Multi-User Dungeon*) criado por Roy Trubshaw, com o auxílio de Richard Bartle, na Universidade de Essex na Inglaterra no final de 1978. O jogo foi escrito originalmente em MACRO-10 assembler e depois dividido em duas partes: a *game engine*, escrita em BCPL, e o mundo do jogo escrito em uma linguagem criada por Trubshaw chamada MUDDL. A ideia era criar um jogo com múltiplos mundos que funcionaria no mesmo motor. O sucesso do jogo foi tão grande que seu nome acabou virando a definição de todo o gênero e, posteriormente, ele passou a ser chamado de MUD1.

A Segunda Era dos mundos virtuais, entre 1985 e 1989, foi marcada pelo experimento constante tanto na criação do mundo do jogo quanto na criação da *game engine*, com muitas contribuições originais vindo do grupo MirrorWorld no sistema IOWA (*Input/Output World of Adventure*). Foi decidido reescrever MUD1 do zero como MUD2 e uma nova linguagem, MUDDLE (*Multi-User Dungeon Definition LanguagE*), foi desenvolvida especificamente para escrever MUDs. A maioria dos MUDs da segunda era foram programados por entusiastas em casa, pois poucas instituições acadêmicas no Reino Unido forneciam recursos de computação como Universidade de Essex. A exceção foi AberMUD, escrito na Universidade de Wales, em Aberystwyth, por Alan Cox, em 1987, que foi posteriormente portado para C, o que gerou oportunidades de grandes avanços no gênero, pois permitiu a execução em ambientes Unix. Praticamente todas as questões-chave do design de MUDs foram identificadas na primeira e na segunda era.

A Terceira Era, entre 1989 e 1995, foi marcada por um período de grande crescimento no número de pessoas experimentando mundos virtuais. AberMUD se espalhou rapidamente entre departamentos de Ciência da Computação universitários, gerando cópias idênticas em milhares de máquinas Unix. Isso resultou em vários imitadores, sendo os principais TinyMUD, LPMUD e DikuMUD. De acordo com uma pesquisa sobre o tráfego na rede NSFnet em 1993, cerca de 10% dos bits pertenciam a MUDs, ou seja, antes da chegada da World Wide Web, os MUDs representavam cerca de 10% da Internet.

A Quarta Era (1995 - 1997) e a Quinta Era (1997 - Presente) foram marcadas por avanços na forma como a internet era comercializada e, consequentemente, a forma como os MUDs eram comercializados. Além disso também houve avanços na tecnologia relacionada a interfaces e gráficos, o que permitiu o surgimento dos primeiros mundos virtuais 2D e, posteriormente, 3D. No final de 1979, o primeiro mundo virtual gráfico totalmente funcional foi lançado, Avatar. No início dos anos 1990, a Kesmai Corporation fez um jogo de simulador de voo *multiplayer*, Air Warrior, que tinha clientes para PC, Atari ST, Commodore Amiga e Apple Macintosh. O Meridian 59 foi projetado por Mike Sellers e Damion Schubert, e pretendia se tornar o primeiro “3D MUD”, objetivo que foi

alcançado (BARTLE, 2004).

2.2 Evolução e Manutenção de Software

A manutenção e evolução de software são atividades inevitáveis no trabalho com software – quase todo software útil e bem-sucedido estimula pedidos de mudança e melhorias gerados pelo usuário (BENNETT; RAJLICH, 2000). No entanto, a definição precisa dessas atividades ainda é objeto de debate.

O Instituto de Engenheiros Eletricistas e Eletrônicos (IEEE) define, em seu glossário, a manutenção de software como sendo a modificação de um produto de software após a entrega para corrigir falhas, melhorar desempenho ou outros atributos, ou adaptar o produto a um ambiente alterado. O IEEE ainda define os termos 'manutenção corretiva', 'manutenção adaptativa' e 'manutenção perfectiva', apontados na tipologia de manutenção de software de Swanson (IEEE, 1990; SWANSON, 1976).

- Manutenção Adaptativa: realizada para tornar um programa de computador utilizável em um ambiente alterado;
- Manutenção Corretiva: realizada para corrigir falhas de hardware ou software;
- Manutenção Perfectiva: realizada para melhorar o desempenho, capacidade de manutenção ou outros atributos de um programa de computador.

Diversos pesquisadores adotaram a terminologia de Swanson, no entanto poucos utilizaram-na como foi definida, o que causa confusão quanto a seu significado, uma vez que eles podem variar de acordo com quem a está usando. O próprio glossário do IEEE é parcialmente inconsistente com a definição original desses termos (CHAPIN et al., 2001). As três intenções para a manutenção de software apontadas por Swanson foram, em resumo:

- Manutenção Adaptativa: para adaptar o sistema às mudanças no seu ambiente de dados ou ambiente de processamento
- Manutenção Corretiva: para corrigir falhas de processamento, desempenho ou implementação do sistema
- Manutenção Perfectiva: para aperfeiçoar o sistema em termos de desempenho, eficiência de processamento ou manutenibilidade

Chapin, por sua vez, define a manutenção de software como a aplicação proposital de atividades e processos, sejam eles completos ou não, em software existente, visando

modificar a forma como o software dirige o hardware do sistema. Isso é geralmente realizado no contexto da evolução de software e difere da definição convencional encontrada no glossário do IEEE, uma vez que o status de 'pós-implantação' ou 'após-entrega' para o software não faz parte da definição.

Além disso, Chapin também define a evolução de software como a aplicação de atividades e processos de manutenção de software que geram uma nova versão de software operacional com uma funcionalidade ou propriedades experimentadas pelo cliente alteradas em relação à versão operacional anterior (CHAPIN et al., 2001). No entanto, assim como outros pesquisadores e profissionais, ele também utiliza o termo como sinônimo de manutenção de software, uma vez que o mesmo carece de uma definição padrão (BENNETT; RAJLICH, 2000; CHAPIN et al., 2001).

Em conjunto com essas definições, Chapin também propõem uma classificação objetiva e mais precisa dos tipos de atividades envolvidas na evolução e manutenção de software, com o objetivo de basear-se em evidências objetivas verificáveis através de observação e/ou comparação do antes e do depois do software; e de proporcionar uma classificação realista e prática para facilitar a comunicação e gestão da evolução e manutenção de software entre pesquisadores, profissionais e seus gestores (CHAPIN et al., 2001). A proposta de Chapin será detalhada de forma resumida na Seção 2.2.1.

2.2.1 Classificação da evolução e manutenção de software

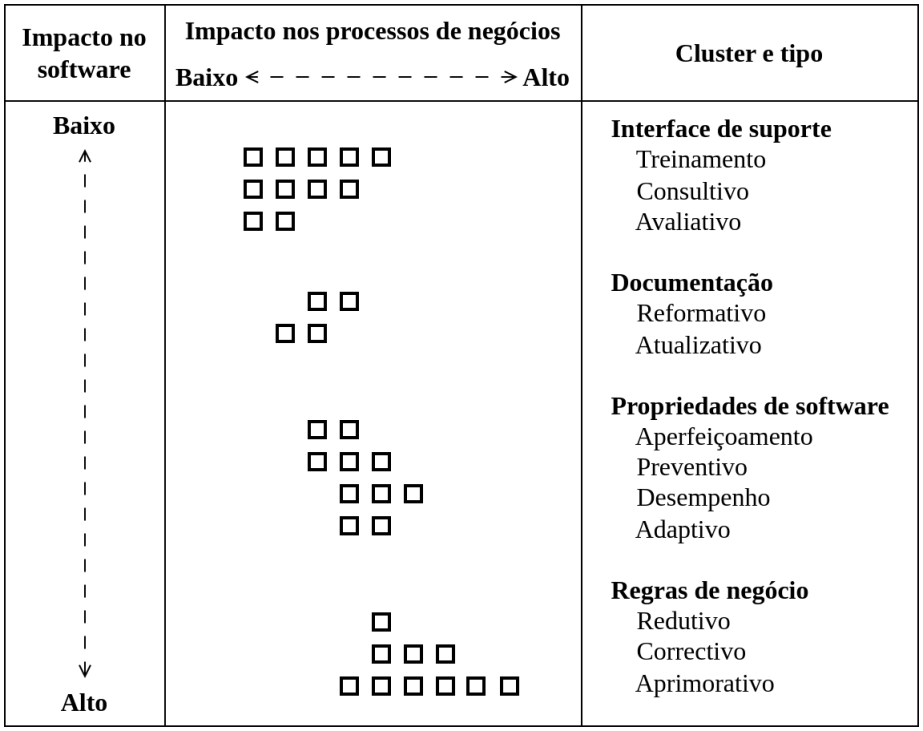
A classificação proposta por Chapin tem como base três critérios de decisão que se fundamentam sobre o local do sistema onde as atividades de manutenção ou evolução do software ocorrem, podendo essas serem realizadas no software (A), no código (B) ou nas funcionalidades experimentadas pelo cliente (C). Esses critérios são satisfeitos a partir da coleta de evidências objetivas, que são adquiridas por meio da comparação do software antes e depois das alterações.

A partir dos critérios atendidos, se define o *cluster* ao qual essa atividade pertence, entre as opções Interface de suporte, Documentação, Propriedades de software e Regas de negócio e, por fim, são escolhidos os tipos de decisão, mutuamente exclusivos, condizentes com a atividade realizada, dentre as opções sugeridas no *cluster* definido. Pode ser feita uma mistura ou agregação em qualquer ordem de alguns ou todos os tipos, mesmo que um tipo possa ser considerado ou observado como dominante.

A ordem dos tipos e seus *clusters* é significativa devido aos seus impactos diferentes, como ilustra a Figura 1. A dimensão horizontal representa o impacto da evolução ou manutenção na capacidade do usuário de utilizar o software de forma eficaz e na sua satisfação com o mesmo, da esquerda (baixo impacto) para a direita (alto impacto). O número de blocos sugere a faixa provável de impacto no cliente. A dimensão vertical

representa o impacto da evolução ou manutenção no próprio software, de cima (impacto baixo) para baixo (impacto alto).

Figura 1 – Impacto dos tipos de evolução e manutenção de software.



Fonte: Chapin (CHAPIN et al., 2001) (com adaptações)

Para definição do tipo de evolução ou manutenção realizada, Chapin fornece uma árvore de decisão condensada, Figura 2, com perguntas de 'Sim' ou 'Não' que encaminham para a opção mais apropriada. A analise é iniciada pela parte do sistema que sofreu alteração, A, B ou C, que indica o *cluster* adequado. Dentro de cada *cluster* são definidas perguntas para cada tipo, que são mostrados em itálico à sua direita, e só são aplicáveis quando a resposta associada aquela pergunta é "Sim".

A leitura da árvore de decisão apresentada na Fig. 2 deve ser realizada da esquerda para direita, enquanto os tipos devem ser lidos de baixo para cima, ambos levando em consideração os padrões de significância ilustrados na Fig. 1, para que se obtenha um impacto crescente tanto no software quanto nos processos de negócios do cliente.

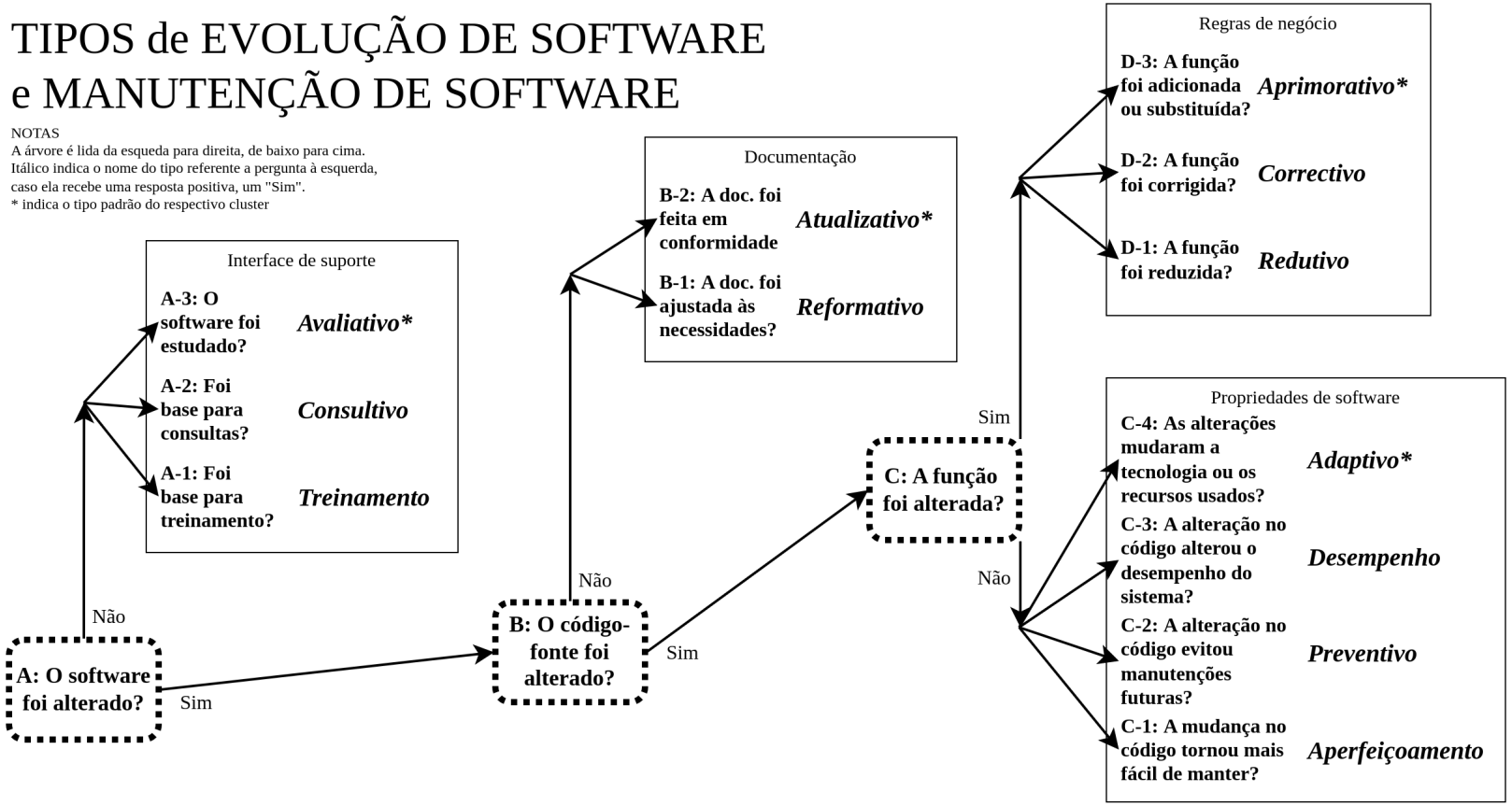
Dado que a evolução ou manutenção de software geralmente implica inúmeros processos ou atividades, a determinação do tipo requer a realização de várias perguntas na árvore de decisão. Progredir para a direita ou para cima na árvore de decisão deixa todos os tipos à esquerda e abaixo como candidatos ativos. E caso um tipo dominante não seja definido, a escolha padrão deve ser aquele com maior impacto que recebeu uma resposta positiva, ou seja, o "Sim" que estiver mais à direita e na posição mais alta.

O tipo de maior impacto dentro de um *cluster* também é a escolha padrão quando

a evidência para discriminar entre os tipos está faltando ou é discutível. Por conta disso, também não é fornecido um tipo "Outro", já que, no caso de um evidência ambígua, existe um padrão a ser seguido.

Caso exista falta de evidências para definir os critérios de decisão A, B, e C, ou caso elas sejam discutíveis, a escolha padrão para responder a árvore de decisão deve ser "Não". Por fim, se a observação e outras evidências indicarem que não foram realizadas atividades ou processos além dos relacionados à execução do software, não houve ocorrência de evolução ou manutenção de software.

Figura 2 – Árvore de decisão para os tipos.



Fonte: Chapin (CHAPIN et al., 2001) (com adaptações)

Diante da ampla variedade de atividades executadas no âmbito da evolução e manutenção de software, é possível observar uma variedade de tipos, mesmo quando um tipo particular é predominante. A Fig. 3 ilustra de maneira simples a relação encontrada entre os tipos definidos, onde os superiores são mais significativos porque têm mais impacto no cliente ou no software ou em ambos.

A chave para entender essa relação retoma os três critérios de decisão A, B e C. Responder "Não" para um desses critérios identifica de imediato um *cluster* respectivo, onde terá pelo menos um tipo de decisão que será escolhido. No entanto, ao responder com "Não" um critério mais impactante, ou seja, que esteja mais a direita, também corresponde a responder com "Não" os critérios menos impactantes, adicionando os respectivos *clustes* à analise.

Apesar da explicação ser complicada, na prática, ao selecionar um tipo de evolução ou manutenção de software, todos os tipos que estiverem listados abaixo dele na Fig. 3 também serão observados durante a realização das atividades, mesmo quem em pequena quantidade. Na definição das classes, a figura utiliza o "E" para indicar evolução de software e o "M" para indicar manutenção de software.

Figura 3 – Relação entre os tipos e classe

Tipo		**Classe**	
Cluster	**Tipo específico**	**E**	**M**
Regras de negócio	Aprimorativo	E	M
	Correctivo		
	Redutivo		
Propriedades de software	Adaptivo	E	M
	Desempenho		
	Preventivo	M	
	Aperfeiçoamento		
Documentação	Atualizativo	M	
	Reformativo		
Interface de suporte	Avaliativo	M	
	Consultivo		
	Treinamento		

Fonte: Chapin (CHAPIN et al., 2001) (com adaptações)

Os tipos de evolução e manutenção de software propostos por Chapin são definidos como:

- *Cluster* de interface de suporte

  - Treinamento (*Training*): envolve atividades como ministrar aulas para o pessoal do cliente, realizar treinamentos, on-site na empresa do cliente ou ambulantes, e utilizar materiais de treinamento existentes;
  - Consultivo (*Consultive*): envolve atividades como atender perguntas sobre o software em uma central de ajuda para o pessoal do cliente, discutir com o pessoal ou gerentes do cliente sobre o uso eficaz de um sistema ou sobre a realização de (propostas ou possíveis) alterações no software, e aconselhar clientes, gerentes ou outros interessados (ou mesmo pesquisadores) sobre o tempo, custo, atividades ou esforço necessários para realizar trabalhos de evolução ou manutenção solicitados ou contemplados.
  - Avaliativo (*Evaluative*): envolve atividades como auditoria, busca, examinação, teste de regressão, estudo, teste diagnóstico, fornecimento de ambientes de teste protegidos, cálculo de métricas sobre, ou criação de uma compreensão do software sem alterá-lo.

- *Cluster* de Documentação
  - Reformativo (*Reformative*): altera a forma da documentação não-código, sem alterar o código. Isso inclui atividades como alterar o formato ou cobertura de um manual do usuário.
  - Atualizativo (*Updative*): envolve atualizar, polir e aumentar a cobertura de documentação não-código, como substituir documentação não-código obsoleta por documentação atual ou preencher lacunas na documentação não-código, incluindo planos de teste, visões gerais narrativas, modelos UML, etc., sem alterar o código.

- *Cluster* de propriedades de software
  - Aperfeiçoamento (*Groomative*): envolve atividades de cuidado com o código fonte, como recompilações, substituição de componentes ou algoritmos por outros mais elegantes, criação de backups de código, alteração de autorizações de acesso pessoais ou de nível, alteração de comentários ou linhas de anotação de código fonte, alteração de convenções de nomenclatura de dados, alteração da legibilidade ou compreensibilidade do código e fornecimento de mensagens de erro menos crípticas.
  - Preventivo (*Preventive*): envolve alterações que não afetam as funcionalidades experimentada pelo cliente ou a tecnologia ou recursos utilizados. Os resultados dessas atividades preventivas raramente são diretamente perceptíveis pelo cliente.

  - Desempenho (*Performance*): envolve processos como a redução do uso de armazenamento interno, melhoria do tempo de atividade do sistema, redução da duração de períodos fora de serviço, aumento da velocidade de execução, substituição de componentes ou algoritmos por outros mais rápidos e melhoria da confiabilidade e robustez do sistema. A mudança em tais propriedades geralmente é percebível pelo cliente.
  - Adaptivo (*Adaptive*): envolve processos como adaptar o sistema para funcionar em uma plataforma diferente, com um sistema operacional modificado ou com um tipo diferente de banco de dados, redistribuir as funções entre componentes ou sub-sistemas, aumentar a utilização de COTS (*Commercial off-the-shelf*), alterar os protocolos de comunicação suportados, mudar a interoperabilidade do sistema e mudar práticas de design e implementação.

- *Cluster* de regras de negócio
  - Redutivo (*Reductive*): de particular importância para a evolução de software, envolve limitar ou eliminar algumas regras de negócios do repertório implementado, como substituir, restringir ou remover parte ou todos os componentes, algoritmos ou sub-sistemas. Fluxos de dados podem ser eliminados ou reduzidos.
  - Corrective (*Corrective*): envolve aprimorar e especificar a implementação das regras de negócios existentes para lidar melhor com exceções e casos mal gerenciados, geralmente adicionando precisão adicional à lógica interna de alguns componentes, algoritmos ou sub-sistemas, ou adicionando mais programação defensiva. Este tipo costuma ser reativo, para remoção de defeitos (*'bugs'*) como aqueles originados de falhas de design ou erros de codificação.
  - Aprimorativo (*Enhancive*): é particularmente significativo para a evolução de software, pois implementa mudanças e adições ao repertório de regras de negócios implementadas pelo sistema, inserindo novos componentes, algoritmos ou sub-sistemas e alterando os existentes para ampliar ou estender seu escopo. Novos fluxos de dados podem ser adicionados e os fluxos de dados existentes podem ser redirecionados.

3 Metodologia

Nesse trabalho serão utilizadas as definições e tipos de manutenção e evolução de software propostos por Chapin, em "*Types of software evolution and software maintenance*" (CHAPIN et al., 2001). A partir da instruções listadas em seu artigo, é possível definir que será realizada uma evolução adaptativa, uma vez que o software será evoluído para uma versão mais moderna de C++, o que irá alterar as práticas de design e implementação, porém sem alterar suas funcionalidades. Por "C++ moderno" entende-se o uso efetivo do padrão ISO C++, que atualmente se encontra na versão C++20, mas que se aplica a todas versões a partir da C++11 (STROUSTRUP; SUTTER et al., 2022).

Apesar do tipo de evolução dominante ser a adaptativa, também será possível ver nas alterações realizadas graus de manutenção avaliativa e para aperfeiçoamento, como explicado na Seção 2.2.1 (Figura 3). Para realizar a evolução proposta será necessário analisar o código original e, após a alterações, o código resultante será mais seguro e manutenível, abrangendo os dois tipos citados (CHAPIN et al., 2001).

> *Upgrading old systems is hard. However, we do believe that a program that uses a rule is less error-prone and more maintainable than one that does not* (STROUSTRUP; SUTTER et al., 2022) [1].

A Seção 3.1 apresenta a abordagem adotada para realização da primeira etapa deste TCC, enquanto a Seção 3.2 será responsável por expor o planejamento realizado para segunda etapa.

3.1 Primeira etapa

O desenvolvimento da primeira fase deste projeto não seguiu uma metodologia estabelecida. Três bases de MUDs foram avaliadas para o desenvolvimento desse projeto: Dirt, de 1993; Dyrt, de 1999; e SimpleMUD de 2003. Tanto o Dirt quanto o Dyrt foram desenvolvidos em C, enquanto o SimpleMUD foi desenvolvido em C++. Mais detalhes sobre as bases de código serão apresentado no Capítulo 4.

Como as fontes de código avaliadas eram antigas, o objetivo principal definido foi fazer com que compilassem corretamente. Para tal, foi realizado um processo iterativo de tentativas de compilação e correção de erros, continuado até que o funcionamento do MUD fosse alcançado, como foi o caso com o SimpleMUD, ou até que não houvesse mais progresso nas correções, como foi observado com o Dyrt. O código do Dirt foi descartado

[1] A atualização de sistemas antigos é difícil. No entanto, acreditamos que um programa que usa uma regra é menos sujeito a erros e mais fácil de manter do que um que não usa. (tradução livre)

do processo iterativo por ser um código mais antigo que o do Dyrt e ter servido como base para o desenvolvimento do mesmo, logo concluiu-se que sua compilação falharia assim como a de seu sucessor.

A compilação e execução do Dyrt foram realizadas em um ambiente Manjaro, versão 5.6, com o compilador GCC, na versão 12.2. Já o SimpleMUD foi compilado e executado em uma máquina virtual, utilizando a imagem Ubuntu 20.04.5 LTS (Focal Fossa), com o compilador G++, na versão 12.2. A máquina virtual foi gerada e configurada com o uso do software Vagrant, na versão 2.3.4. O arquivo de configuração utilizado pelo Vagrant está disponível no Apêndice A.

Após o sucesso na compilação e execução do SimpleMUD, foi iniciado um novo ciclo de atividades para a sua evolução e manutenção. Este ciclo incluiu a pesquisa das diretrizes básicas de C++, conforme apresentadas por Bjarne Stroustrup e Herb Sutter (STROUSTRUP; SUTTER et al., 2022), assim como outras boas práticas recomendadas para o desenvolvimento de C++, como utilização de um ambiente de desenvolvimento contínuo, adição de *flags* de *warning* no compilador, utilização de ferramentas para análise estática e análise em tempo de execução do código, entre outras apresentadas pelo canal de Youtube de Jason Turner, intitulado C++ Weekly[2], nos episódios 350[3] e 355[4]. Além disso, foi realizada uma análise do código para identificar trechos que pudessem ser melhorados. Depois de realizada a análise, as atualizações foram implementadas e documentadas. A análise e o planejamento para atualização do código foram realizados em pareamentos com o professor orientador Edson Alves da Costa Júnior.

Durante a primeira etapa, a falta de uma metodologia formal de desenvolvimento resultou em uma evolução mal documentada do código. Não houve uma estratégia clara de melhoria, as modificações foram realizadas baseadas nas percepções e conhecimentos prévios dos desenvolvedores envolvidos, e não foram coletados dados quantitativos para comprovar a efetividade dessas alterações.

3.2 Segunda etapa

A segunda etapa deste trabalho teve como objetivo principal organizar o ambiente de desenvolvimento do SimpleMUD, de modo a garantir uma evolução e manutenção eficazes, assim como organizar o repositório do projeto e sua documentação. De acordo com Turner (2022), são cruciais para o desenvolvimento adequado em C++:

- ambiente de desenvolvimento contínuo: GitHub, GitLab, Jenkins, entre outros;

[2] <https://www.youtube.com/@cppweekly>
[3] <https://youtu.be/q7Gv4J3FyYE>
[4] <https://youtu.be/dSYFm65KcYo>

- utilização do máximo de compiladores possíveis: GCC, Clang, entre outros;
- estrutura de teste organizada: Doctest, Catch, Gtest, Boosttest, entre outros;
- análise, relatórios e rastreamento da cobertura de testes: Coveralls, Codecov, entre outros;
- máximo de análise estática possível: *flags* de alerta, como `-Wall -Wextra -Wshadow -Wconversion -Wpedantic -Werror`, a *flag* `-fanalyzer` do GCC, Cppcheck, Clang-tidy, entre outros;
- análise de tempo de execução durante os testes: AddressSanitizer, Undefined Behavior Sanitizer, ThreadSanitizer, entre outros;
- teste de Fuzz;
- lançamento com reforço de segurança habilitado: Control Flow Guard, Stack Protector, entre outros.

No entanto, suprir todas essas necessidades para um desenvolvimento adequado não é algo trivial e, para isso, foi necessário um planejamento prévio.

Com a implementação do ambiente de desenvolvimento, foi possível realizar uma avaliação quantitativa da base de código analisada e elaborar um plano de melhoria apropriado. Além disso, as modificações realizadas na primeira etapa foram avaliadas, uma vez que todo o código foi versionado com a ferramenta Git e armazenado no Github.

3.2.1 Scrum Solo e Kanban

A metodologia principal adotada para o desenvolvimento dessa segunda etapa do projeto é o Scrum Solo, que consiste em uma adaptação do Scrum voltada para microempresas de software com um único desenvolvedor (desenvolvedor solo), mas que também foi adotada por diversos estudantes para elaboração de seus trabalhos de conclusão de curso (PAGOTTO et al., 2016).

Em conjunto com o Scrum Solo foram adotados alguns princípios do Kanban (BOEG, 2010), com destaque para o princípio da visualização do fluxo de trabalho, que será representado por meio do Quadro Kanban, com os fluxos: *To do*, *Doing*, *Review* e *Done*.

3.2.2 Organização do repositório

A organização do repositório foi feita com base nos modelos apresentados por Brasileo (2022), que apresenta uma pesquisa sobre repositórios de código aberto e disponibiliza modelos de equipe, processos e boas práticas. Para que seja possível criar uma

comunidade ativa para o sistema desenvolvido, serão seguidas as propostas apresentadas no modelo de boas práticas nesta etapa de desenvolvimento, sendo elas:

- presença do README;
- definição da Licença;
- página de Apresentação do Projeto;
- integração Contínua;
- presença do Guia de Contribuição;
- presença do Modelo de Issues;
- presença do Código de Conduta;
- presença do Modelo de Pull Request;
- uso de Linguagem Universal.

As atividades propostas no modelo estão inclusas como histórias de usuário no *Backlog* do Produto e foram implementadas conforme suas prioridades.

3.2.3 Integração e Deploy

Como foi citado, o sistema desenvolvido foi armazenado no GitHub, o que possibilitou a utilização de diversas ferramentas para auxiliar na integração das modificações realizadas, assim como para realização do *deploy*. A ferramenta escolhida para realizar essas atividades foi o *GitHub Actions*, que fornece um servidor onde é possível integrar o código e executar testes unitários e outras verificações. Além da integração, a ferramenta possibilita a configuração de fluxos de trabalho para realizar o *deploy* do sistema.

3.2.4 Testes de Software

Para o desenvolvimento dos testes unitários do software foi elaborado um planejamento de testes, conforme as explicações apresentadas por Everett e Jr (2007) no capítulo 5, "*Test Planning*". Após o desenvolvimento do planejamento, os testes serão elaborados utilizando o *framework* GoogleTest e a cobertura de testes será analisada pela ferramenta `gcovr`.

A parte de análise estática foi desenvolvida com a adição das *flags* recomendadas em Cpp-Best-Practices (2021), que é um repositório mantido por Turner com o objetivo de construir uma coleção colaborativa de boas práticas de C++. Também foram utilizadas as ferramentas Cppcheck e Clang-Tidy.

3.2.5 Melhorias de Software

Para realização de melhorias no código, primeiramente foi elaborado um planejamento de melhoria. Esse planejamento consistiu na listagem e classificação dos problemas encontrados no sistema, bem como na determinação da ordem de resolução dessas questões. A lista de problemas foi baseada nos relatórios gerados pela execução dos softwares Cppcheck, Clang-Tidy e nos *warnings* emitidos pelo GCC. Já a classificação foi baseada no conhecimento que se tinha a respeito do problema, a estimativa de tempo necessário para solucioná-lo e o quão complexo seria aplicar essa solução.

3.2.6 Banco de Dados

A base de dados utilizada pela aplicação é o PostgreSQL. A modelagem dos dados seguiu a abordagem entidade-relacionamento e foi desenvolvido um Diagrama Entidade-Relacionamento (DER), assim como um Diagrama Lógico de Dados. Por fim, foi realizado o processo de normalização do dados, seguido da implementação do banco em si, por meio de *scripts* SQL.

A escolha do PostgreSQL foi feita pelo professor Maurício Serrano, para um melhor aproveitamento do projeto pelas matérias de Banco de Dados 1, que utilizam o mesmo banco de dados. O processos realizados para modelagem dos dados seguem os trabalhos de Barcelar (2012) e Bagui (2009).

3.3 Ferramentas Utilizadas

Nesta seção são detalhadas as ferramentas utilizadas no desenvolvimento desse trabalho. É importante destacar que ferramentas foram acrescidas no decorrer da segunda etapa, visto que os passos estabelecidos por Turner requerem o uso de vários recursos externos. *If you are developing blindly, without any tool guidance, you are doing C++ wrong* (TURNER, 2022) [5].

3.3.1 Clang-format

ClangFormat é uma ferramenta desenvolvida a partir da LibFormat, uma biblioteca que implementa formatação automática de código-fonte baseada no Clang. Essa ferramenta pode ser útil em diversos fluxos de trabalho, incluindo um utilitário autônomo e integrações com editores de texto (TEAM, 2023a).

[5] Se você está desenvolvendo às cegas, sem nenhuma orientação de ferramenta, você está utilizando C++ errado. (tradução livre)

3.3.2 Clang-tidy

O clang-tidy é uma ferramenta de "linter"[6] para C++ baseada em clang. Seu propósito é fornecer um framework extensível para diagnosticar e corrigir erros típicos de programação, como violações de estilo, uso incorreto de interface ou *bugs* que podem ser deduzidos por meio de análise estática. O clang-tidy é modular e oferece uma interface conveniente para escrever novas verificações (TEAM, 2023b).

3.3.3 Cppcheck

O Cppcheck é uma ferramenta de análise estática usada para analisar códigos C/C++. Ela fornece recursos de análise de código distintos para identificar erros de programação e se concentra na detecção de comportamentos indefinidos e práticas de codificação potencialmente perigosas. O objetivo é minimizar os falsos positivos. O Cppcheck foi desenvolvido para analisar códigos C/C++ com sintaxe não padrão, comumente encontrada em projetos embarcados. Cppcheck é oferecido como um software de código aberto e também como Cppcheck Premium, que oferece funcionalidades adicionais e suporte (CPPCHECK, 2023).

3.3.4 Docker e Docker Compose

O Docker é uma plataforma aberta que visa o desenvolvimento, distribuição e execução de aplicativos. Através do Docker, é possível separar os aplicativos da infraestrutura, garantindo uma entrega de software ágil. É possível empacotar e executar aplicativos em um ambiente isolado, denominado contêiner. Além disso, permite a execução de vários contêineres simultaneamente em um mesmo *host*, sem sobrecarregar o sistema. Os contêineres, por sua vez, são leves e incluem todos os componentes necessários para a execução do aplicativo, o que evita a necessidade de depender do que está atualmente instalado no *host* (DOCKER, 2023b).

O Docker Compose é uma ferramenta utilizada para definir e executar aplicativos Docker compostos por múltiplos contêineres. Por meio do Compose, é possível configurar os serviços do aplicativo por meio de um arquivo YAML e, com apenas um comando, é possível criar e iniciar todos os serviços com base na configuração definida. Além disso, o Compose é compatível com diferentes ambientes, tais como produção, *staging*, desenvolvimento, teste, assim como fluxos de trabalho de integração contínua (DOCKER, 2023a).

[6] Lint é o termo da ciência da computação para uma ferramenta de análise de código estático usada para sinalizar erros de programação, *bugs*, erros estilísticos e construções suspeitas.(Fonte: Wikipédia)

3.3.5 Docsify

O Docsify é uma ferramenta de geração de sites estáticos. A partir do carregamento e análise de arquivos Markdown, ele gera um *website* voltado para documentação de projetos e de fácil integração com o GitHub Pages (DOCSIFY, 2023).

3.3.6 GCC

A GNU *Compiler Collection* (GCC) é um compilador produzido pelo Projeto GNU que oferece suporte a várias linguagens de programação, arquiteturas de hardware e sistemas operacionais, como o C, C++, Objective-C, Fortran, Ada, Go, and D. Ela também inclui bibliotecas padrões para essas linguagens, como a `libstdc++` (GNU, 2023a).

3.3.7 gcovr

O `gcovr` é uma ferramenta que gerencia a utilização da ferramenta GNU gcov para gerar resultados resumidos de cobertura de código. Ela foi inspirada no pacote `coverage` do Python, que oferece funcionalidade similar para Python. O comando `gcovr` produz diferentes tipos de relatórios de cobertura, como Texto, HTML e XML e, por essa razão, pode ser uma alternativa de linha de comando ao utilitário `lcov`, que executa o `gcov` e gera um relatório HTML formatado. O desenvolvimento do `gcovr` foi motivado pela necessidade de resumos em texto e relatórios em XML (GCOVR, 2023).

3.3.8 Git

Git é uma ferramenta gratuita e de código aberto de controle de versão distribuído, desenvolvida por Linus Torvalds em 2005. Alguns dos objetivos desse sistema incluem velocidade, design simples, forte suporte ao desenvolvimento não linear (milhares de ramificações paralelas), totalmente distribuído e capaz de lidar eficientemente com projetos de grande porte, como o núcleo do Linux (velocidade e tamanho de dados). Desde o seu nascimento em 2005, o Git evoluiu e amadureceu para ser fácil de usar e ainda retêm essas qualidades iniciais (CHACON; STRAUB, 2014).

3.3.9 GitHub

GitHub é uma plataforma de gerenciamento de código baseada na web que fornece soluções completas para armazenamento, controle de versão e colaboração em projetos. Ela hospeda repositórios do Git e oferece aos desenvolvedores ferramentas eficientes como linha de comando, problemas (discussões encadeadas), *pull requests*, revisão de código e acesso a uma ampla coleção de aplicativos grátis e pagos no GitHub Marketplace. Oferece planos gratuitos e pagos para usuários pessoais e organizações, permitindo que

colaboradores trabalhem juntos em repositórios públicos ou privados com acesso a diferentes conjuntos de recursos. Com uma comunidade de 15 milhões de desenvolvedores, juntamente com integrações eficientes, GitHub revolucionou a forma como o software é construído (GITHUB, 2023b).

3.3.10 GitHub Actions

GitHub Actions é uma plataforma de integração contínua e entrega contínua (CI/CD) que permite a automatização de processos como compilação, testes e *pipeline* de implantação. A plataforma possibilita a criação de fluxos de trabalho para a realização de testes em cada *pull request* de um repositório ou para implantar *pull requests* mesclados em produção. A plataforma conta com máquinas virtuais do Linux, Windows e macOS para execução de fluxos de trabalho, além de permitir que executadores auto-hospedados sejam hospedados em infraestrutura de dados própria ou em nuvem (GITHUB, 2023a).

3.3.11 GoogleTest

O GoogleTest é um framework de testes desenvolvido pela equipe de Tecnologia de Testes da Google com o objetivo de atender aos requisitos e limitações específicos da empresa. O GoogleTest pode ser utilizado por desenvolvedores que trabalham em sistemas operacionais como Linux, Windows ou Mac e que escrevem código em C++. É importante ressaltar que o GoogleTest suporta diversos tipos de testes, além dos testes unitários (GOOGLE, 2023).

3.3.12 GNU Make

O GNU Make é uma ferramenta utilizada para controlar a geração de executáveis e outros arquivos não-fonte de um programa a partir dos arquivos-fonte do programa. O Make obtém seu conhecimento sobre como construir o programa a partir de um arquivo chamado *makefile*, que lista cada um dos arquivos não-fonte e como computá-los a partir de outros arquivos (GNU, 2023c).

3.3.13 PostgreSQL

O PostgreSQL é um sistema de banco de dados relacional orientado a objetos de código aberto, com mais de 35 anos de desenvolvimento ativo. Ele combina recursos avançados para armazenar e dimensionar com segurança cargas de trabalho complexas, e possui uma ampla comunidade de código aberto que oferece soluções inovadoras.

O PostgreSQL é amplamente suportado em diferentes sistemas operacionais e oferece recursos adicionais, como o PostGIS para dados geoespaciais. Ele se estabeleceu

como uma escolha popular para pessoas e organizações que buscam um banco de dados relacional de código aberto (POSTGRESQL, 2023).

3.3.14 SonarQube

O SonarQube é uma ferramenta de revisão de código automática auto-gerenciada, que auxilia de forma sistemática na entrega de código limpo. Na qualidade de elemento central da solução Sonar, o SonarQube integra-se ao fluxo de trabalho existente e identifica problemas no código para permitir a realização de inspeções de código contínuas em projetos. A ferramenta é capaz de analisar mais de 30 linguagens de programação distintas e se integra ao *pipeline* de CI e à plataforma DevOps para assegurar que o código atenda a elevados padrões de qualidade (SONARQUBE, 2023).

3.3.15 Vagrant

Vagrant é uma ferramenta destinada à construção e gestão de ambientes de desenvolvimento completos baseados em máquinas virtuais. Com uma abordagem fácil de usar e focada na automação, Vagrant contribui para a redução do tempo de configuração do ambiente de desenvolvimento e para o aumento da correspondência entre o desenvolvimento e a produção. As máquinas virtuais podem ser provisionadas por diversos provedores, tais como o VirtualBox, VMware, AWS, e outros. As ferramentas de provisionamento padrão da indústria, como *shell scripts*, Chef ou Puppet, podem ser empregadas para a instalação e configuração automatizadas do software na máquina virtual (VAGRANT, 2023).

4 Resultados Obtidos

Duas bases de código foram consideradas para realização da evolução e manutenção de software; os MUDs Dirt, desenvolvido por Alf Salte e G. Sorseth, versão 3.1.2, de 1993, e Dyrt, desenvolvido por Valentin Popescu e Katie Mowry, versão 1, de 1999. Ambos jogos foram desenvolvidos em C e tiveram bases semelhantes, uma vez que o Dirt se derivou do AberMUD, enquanto o Dyrt se derivou do próprio Dirt. Por ser um pouco mais novo, o código do Dyrt foi priorizado para análise e possível manutenção.

Após diversas tentativas de correção, não houve sucesso na execução do código do Dyrt e ele foi descartado. Por serem bases de código legado, diversos erros relacionados a bibliotecas e funções depreciadas foram encontrados, o que levou a consideração de uma terceira base de código: o SimpleMUD.

O SimpleMUD, desenvolvido por Ron Penton em seu livro "MUD Game Programming" (PENTON, 2003), é um exemplo simples e didático de MUD, que demonstra como combinar um sistema de dados com um sistema de reação de rede. Sua parte física e lógica foram desenvolvidas exclusivamente em C++, enquanto a camada de dados foi armazenada utilizando arquivos de dados ASCII simples (PENTON, 2003). Apesar de suas limitações, o SimpleMUD atende às expectativas das bases de código analisadas, em relação a ser um MUD, e, mesmo tendo sido desenvolvido em 2003, ele não apresentou problemas significativos relacionados a sua datação.

Esse capítulo está dividido em duas seções, referentes a cada uma dessas bases de código. A Seção 4.1 traz as correções realizadas no Dyrt, enquanto a Seção 4.2 apresenta as correções e melhorias feitas no SimpleMUD, bem como os testes unitários implementados e sua integração com o banco de dados.

4.1 Dyrt

Essa seção tem como objetivo apresentar os esforços realizados para correção do código do Dyrt.

4.1.1 Instalação do GDBM

O primeiro passo para instalação do Dyrt é a instalação do GDBM (GNU *Database Manager*), gerenciador de banco de dados utilizado pelo programa. A base de dados fornecia junto a seus arquivos a versão 1.7.3 desse gerenciador, lançada em 1995.

GNU DBM (*Database Manager*) é uma biblioteca de funções de banco de dados que

usa *hashing* extensível e funciona de maneira semelhante ao DBM padrão do UNIX. Ele permite armazenar pares chave/dado em um arquivo de dados, buscar e recuperar dados pela chave e deletar uma chave com seus dados. Também suporta a iteração sequencial sobre todos os pares chave/dado em um banco de dados. Possui compatibilidade com programas que usam as antigas funções DBM do UNIX (GNU, 2023b).

A instalação da versão 1.7.3 do GDBM consiste em três etapas: executar o programa de configuração das variáveis de instalação do gerenciador (`./configure`); compilar o GDBM (`make`); e instalar o pacote básico (`make install`). Tanto o segundo quanto o terceiro passos são realizados com o auxílio de um *Makefile*, que é gerado pelo script `configure` citado no primeiro passo. Ainda existem duas etapas opcionais para compilar os programas opcionais de teste e conversão (`make progs`) e para instalar os cabeçalhos opcionais de compatibilidade `dbm` e `ndbm` (`make install-compat`).

Não houveram problemas na realização da configuração e da compilação, porém ao executar a instalação dos pacotes básicos foram encontrados os erros apresentados nos Códigos 1 e 2. Para corrigí-los foi feita a criação das pastas `man3` e `info` nos caminhos de diretório `/usr/local/man` e `/usr/local`, respectivamente. Após isso também foram feitas alterações no *Makefile* do GDBM, para que esses erros não voltassem a acontecer (Código 3).

Código 1 – Erro: Pasta `man3` não encontrada.

```
1 /usr/bin/install -c -m 644 ./gdbm.3 /usr/local/man/man3/gdbm.3
2 /usr/bin/install: cannot create regular file '/usr/local/man/man3/gdbm.3': No
  ↪ such file or directory
```

Fonte: Autor

Código 2 – Erro: Pasta `info` não encontrada.

```
1 /usr/bin/install -c -m 644 ./gdbm.info /usr/local/info/gdbm.info
2 /usr/bin/install: cannot create regular file '/usr/local/info/gdbm.info': No
  ↪ such file or directory
```

Fonte: Autor

Código 3 – GDBM *Makefile* atualizado.

```
1 # GDBM Makefile, linha 96
2 install: libgdbm.a gdbm.h gdbm.info
3     $(INSTALL_DATA) libgdbm.a $(libdir)/libgdbm.a
4     $(INSTALL_DATA) gdbm.h $(includedir)/gdbm.h
```

```
5       @mkdir -p $(man3dir) # comando adicionado
6       $(INSTALL_DATA) $(srcdir)/gdbm.3 $(man3dir)/gdbm.3
7       @mkdir -p $(infodir) # comando adicionado
8       $(INSTALL_DATA) $(srcdir)/gdbm.info $(infodir)/gdbm.info
```

Fonte: Autor

Ambas partes opcionais da instalação resultaram em erros ao serem executadas. Por não serem obrigatórias para o funcionamento do sistemas, esses erros não foram corrigidos, mas podem ser vistos nos Códigos 4 e 5.

Código 4 – Erro ao compilar os programas opcionais de teste e conversão.

```
1  /usr/bin/ld: tndbm.o: in function 'main':
2  tndbm.c:(.text+0x1b5): warning: the 'gets' function is dangerous and should not
   ↪  be used.
3  /usr/bin/ld: tndbm.c:(.text+0x56): undefined reference to 'dbm_open'
4  /usr/bin/ld: tndbm.c:(.text+0x12c): undefined reference to 'dbm_firstkey'
5  /usr/bin/ld: tndbm.c:(.text+0x14b): undefined reference to 'dbm_fetch'
6  /usr/bin/ld: tndbm.c:(.text+0x188): undefined reference to 'dbm_close'
7  /usr/bin/ld: tndbm.c:(.text+0x1e3): undefined reference to 'dbm_fetch'
8  /usr/bin/ld: tndbm.c:(.text+0x2bd): undefined reference to 'dbm_store'
9  /usr/bin/ld: tndbm.c:(.text+0x386): undefined reference to 'dbm_delete'
10 /usr/bin/ld: tndbm.c:(.text+0x3c5): undefined reference to 'dbm_nextkey'
11 /usr/bin/ld: tndbm.c:(.text+0x3e0): undefined reference to 'dbm_fetch'
12 /usr/bin/ld: tndbm.c:(.text+0x42e): undefined reference to 'dbm_firstkey'
13 /usr/bin/ld: tndbm.c:(.text+0x44a): undefined reference to 'dbm_nextkey'
```

Fonte: Autor

Código 5 – Erro ao instalar os cabeçalhos opcionais de compatibilidade `dbm` e `ndbm`.

```
1  /usr/bin/install -c -m 644 ./dbm.h /usr/local/include/dbm.h
2  /usr/bin/install -c -m 644
3  /usr/bin/install: missing file operand
4  Try '/usr/bin/install --help' for more information.
```

Fonte: Autor

4.1.2 Instalando o Dyrt

Após a instalação do GDBM, os passos para instalar o Dyrt consistem em configurar as variáveis de instalação e compilar o programa, com auxílio do `Makefile`. Os dois primeiros passos são alterar os valores referentes ao ambiente onde o jogo irá ser rodado no

arquivo `config.h`, encontrado na pasta `../include/`, e rodar o comando `make depend`. Ambos os passos foram realizados sem problemas. Após isso, é necessário compilar o jogo com o comando `make`.

Diferente da etapa de configuração, a compilação resultou em diversos erros e alertas. A maioria desses erros eram consequência de atualizações em bibliotecas e até mesmo no próprio compilador de C, o que fez com que algum deles se repetissem diversas vezes, porém em arquivos diferentes.

O primeiro erro corrigido foi referente ao uso da variável `sys_errlist[errno]`, que está depreciada e foi substituída pela função `strerror(errno)` (Código 6). A correção foi realizada nos arquivos `boostrap.c`, `generate.c`, `log.c`, `main.c`, `misc.c` e `timing.c`, em 10 trechos diferentes do código.

Código 6 – Exemplo de erro resultante do uso da variável `sys_errlist[errno]`.

```
1 generate.c: In function 'make_verbs':
2 generate.c:104:18: error: 'sys_errlist' undeclared (first use in this function)
3   104 |             buff,sys_errlist[errno]);
4       |                  ^~~~~~~~~~
```

Fonte: Autor

O segundo erro encontrado foi resultado da definição de um caminho de diretório para o executável do compilador de C++ que não é mais o adotado (Código 7). A correção foi simples, bastou atualizar o caminho definido na constantes `CPP`, declarada no arquivo `/include/machine/linux.h` (Código 8).

Código 7 – Tentativa de acesso ao caminho de diretório `/lib/cpp`.

```
1 gcc -o ..//bin/generate -DDEBUG -g -O4 -I..//include/ -DLINUX -DREBOOT
  ↪  generate.o -lgdbm -lcrypt
2 Regenerating World....
3 ..//bin/generate data ../
4 (*) Generating dyrt userfiles...
5
6 sh: line 1: //lib/cpp: No such file or directory
7
8 Zone not found: limbo in file ..//data/WORLD/limbo.zone.
```

Fonte: Autor

Código 8 – Definição da constante CPP no arquivo linux.h.

```
1 // linux.h, linha 6
2 // Definição original
3 #define CPP "//usr/bin/cpp -P -traditional -I ../include %s"
4 // Caminho atualizado
5 #define CPP "//usr/bin/cpp -P -traditional -I ../include %s"
```

Fonte: The Dyrt Project (1995 - 1999) (com adaptações)

Após esse erro, apareceram diversos erros de variáveis e funções redeclaradas, declarações implícitas e referencias inválidas. O Código 9 apresenta um compilado de todos esses erros, na ordem em que apareceram.

Código 9 – Compilado de erros levantados ao executar o comando `make`.

```
1 main.c: In function 'get_options':
2 main.c:634:15: error: invalid storage class for function 'usage'
3   634 |   static void usage (void);
4       |               ^~~~~
5 main.c: In function 'main_loop':
6 main.c:767:15: error: invalid storage class for function 'new_connection'
7   767 |   static void new_connection (int fd);
8       |               ^~~~~~~~~~~~~~
9
10 sflag.c:565:1: error: static declaration of '_wiz' follows non-static
   ↪ declaration
11 sflag.c:522:3: note: previous implicit declaration of '_wiz' with type
   ↪ 'void(int,  char *)'
12   522 |   _wiz (LVL_APPRENTICE, wordbuf);
13       |   ^~~~
14
15 /usr/bin/ld: errno: TLS definition in /usr/lib/libc.so.6 section .tbss
   ↪ mismatches non-TLS reference in bootstrap.o
16 /usr/bin/ld: /usr/lib/libc.so.6: error adding symbols: bad value
17 collect2: error: ld returned 1 exit status
18 make: *** [Makefile:158: ..//bin/aberd] Error 1
19
20 gcc -o aberd -DDEBUG -g -O4 -I..//include/ -DLINUX -DREBOOT actions.o admin.o
   ↪ board.o bootstrap.o bprintf.o calendar.o change.o clone.o commands.o
   ↪ communicate.o condition.o fight.o flags.o frob.o game.o hate.o log.o
   ↪ magic.o mail.o main.o misc.o mobile.o move.o mud.o objsys.o parse.o party.o
   ↪ rooms.o s_socket.o sendsys.o sflag.o timing.o uaf.o utils.o wizard.o
   ↪ wizlist.o writer.o zones.o  -lgdbm -lcrypt
```

```
21 /usr/bin/ld:
   ↪ admin.o:/home/iuri/Desktop/pibic/Pibic/dyrt_19/dyrt/src/..//include/mud.h:15:
   ↪                   multiple definition of `a_new_player';
   ↪ actions.o:/home/iuri/Desktop/pibic/Pibic/dyrt_19/dyrt/src/..//include/mud.h:15:
   ↪ first defined here
22 /usr/bin/ld:
   ↪ timing.o:/home/iuri/Desktop/pibic/Pibic/dyrt_19/dyrt/src/timing.c:56:
   ↪                   multiple definition of `next_event';
   ↪ mud.o:/home/iuri/Desktop/pibic/Pibic/dyrt_19/dyrt/src/..//include/global.h:86:
   ↪ first defined here
```

Fonte: Autor

A funções `usage(void)` e `new_connection(int fd)` já haviam sido declaradas no arquivo `main.c`, nas linhas 40 e 45, e, para corrigir o erro, bastou comentar a re-declaração delas, nas linhas 634 e 767. A função `static void _wiz(int, char *)` era chamada diversas vezes antes da sua declaração, para correção desse erro foi adicionado um cabeçalho de declaração para função, no início do arquivo `sflag.c`. A definição da variável `errno` não estava correspondendo com a sua definição, pois estava sendo acessada a partir da declaração `extern int errno;`. A correção foi feita a partir da remoção da declaração de `errno` como uma variável externa e inclusão da biblioteca `errno.h`. Por fim, foram adicionadas as variáveis `extern Boolean a_new_player` na biblioteca `mud.h` e `extern time_t next_event;` no arquivo `timing.c`, e a biblioteca `mud.h` foi incluída ao arquivo `mud.c`.

Após a correção de todos os erros, o comando `make` foi executado sem erros. Foi necessário executar o script `verbgen` para geração dos verbos e essa ação se mostrou necessária em toda compilação após a limpeza dos arquivos (`make clean`).

4.1.3 Executando o Dyrt

Apesar da compilação ter sido bem sucedida, a linkagem do executável não aconteceu como deveria, impedindo sua execução pelo comando `aberd &`, como recomendado pelas instruções de instalação. Porém, a execução direta do binário criado funcionou.

A primeira execução do MUD gerou a saída apresentada no Código 10. O resultado final foi uma falha de segmentação de memória, erro que acontece quando o software tenta acessar uma área restrita de memória (uma violação de acesso à memória).

Para depuração do código foi utilizada a função `printf()` em conjunto com a execução do programa, com o objetivo de encontrar a função onde o erro acontecia. Essa opção foi escolhida devido a baixa familiaridade com a ferramenta GDB (GNU Project Debugger). Após várias execuções, foi descoberto que o erro era levantado pela função

`gdbm_open()`, da biblioteca GMDB, em sua versão 1.7.3. Para correção do erro, foi feita a atualização da biblioteca para sua versão mais atual, 1.23, lançada em 2022. Não houveram erros decorrentes da alteração realizada.

Código 10 – Saída apresentada pela execução do Dyrt.

```
1  data_dir = "../data/".
2  max_players = 40.
3  port = 6715.
4  Do not clear syslog file.
5  Debugging is on.
6  Kill other mud.
7  Bootstrap... ID-table & ID-counter used 8192 bytes.
8  Bootstrap... 'players' used 602880 bytes.
9  Bootstrap... A:actions used 61780 bytes.
10 Bootstrap... Z:zones used 7856 bytes.
11 Bootstrap... L:locations used 923949 bytes.
12 Bootstrap... C:mobiles used 192716 bytes.
13 Bootstrap... E:levels used 0 bytes.
14 Bootstrap... H:hours used 0 bytes.
15 Bootstrap... I:intermud.conf used 0 bytes.
16 Bootstrap... O:objects used 279263 bytes.
17 Bootstrap... P:pflags used 0 bytes.
18 Bootstrap... V:verbs used 3200 bytes.
19 Bootstrap... W:wizlist ---->Can`t open W file wizlist used 0 bytes.
20
21 A total of 2071644 bytes used.
22 Connected to port 6715 on lucas-tkp.
23 Segmentation fault (core dumped)
```

Fonte: Autor

Apesar da correção ter tido êxito, a segunda execução do programa teve o mesmo resultado, sendo finalizada por um erro de segmentação. No entanto, o erro estava acontecendo por outro motivo. A princípio foi feita uma depuração utilizando o `printf()` novamente, o que indicou que o erro ocorria na função `consid_move()`. A partir dessa função, a depuração com a função `printf()` se tornou inviável, uma vez que a função `consid_move()` era chamada dentro de um laço de repetição que iterava 338 vezes. Para contornar esse problema foi utilizado o GDB, que indicou que o erro partia da função `check_send_msg()`, localizada no arquivo `sendsys.c`, na linha 210, devido a uma conversão de um valor inteiro negativo para ponteiro do tipo `struct _send_msg_box` (Código 11).

Código 11 – Função `check_send_msg` original.

```
1 // sendsys.c, linha 210
2 char *check_send_msg(int plx, int a, char *t)
3 {
4     struct _send_msg_box *b = (struct _send_msg_box *)a;
5     if (test_rcv(plx, b->mode, b->min, b->max, b->x1, b->x2))
6         return t;
7     return NULL;
8 }
```

Fonte: The Dyrt Project (1995 - 1999) (com adaptações)

Ao realizar a depuração da função `check_send_msg()` foi observado que tanto a função quanto a variável convertida eram transmitidas como parâmetro para função `send_g_msg()` pela função `send_msg()`, ambas localizadas no arquivo `sendsys.c`, nas linhas 114 e 220, respectivamente (Códigos 12 e 13).

Código 12 – Cabeçalho da função `send_g_msg` original.

```
1 // sendsys.c, linha 114
2 /* Send general message.  */
3 void send_g_msg(int destination,                          /* Where to send to */
4                 char *func(int plx, int arg, char *t), /* Test function */
5                 int arg,                                 /* Argument to test */
6                 char *text)                              /* Text to send */
```

Fonte: The Dyrt Project (1995 - 1999) (com adaptações)

Código 13 – Função `send_msg` original.

```
1 // sendsys.c, linha 220
2 void send_msg(int destination,   /* Where to send to */
3               int mode,          /* Flags to control sending */
4               int min,           /* Minimum level of recipient */
5               int max,           /* Maximum level of recipient */
6               int x1,            /* Do not send to him */
7               int x2,            /* Nor to him */
8               char *format, ...) /* Format with args -> text to send */
9 {
10     struct _send_msg_box b;
11     b.mode = mode;
12     b.min = min;
13     b.max = max;
14     b.x1 = x1;
```

```
15      b.x2 = x2;
16      send_g_msg(destination, check_send_msg, (int)&b, bf);
17  }
```

Fonte: The Dyrt Project (1995 - 1999) (com adaptações)

Para corrigir o erro, foi retirada a conversão da área de memória para inteiro, feita na chamada da função `send_g_msg()` pela função `send_msg()`, e foram feitas alterações nos parâmetro das funções `send_g_msg()` e `gsendf()`, localizada na linha 262, do arquivo `sendsys.c`, que recebiam como argumento um `char *func(int plx, int arg, char *text)` e um `int arg`, e passaram a receber um `char *func(int plx, void *arg, char *text)` e um `void *arg`. Dessa forma, a área de memória começou a ser passada por completo para as outras funções, o que evitou o *overflow* que podia acontecer quando se tentava realizar a conversão para inteiro. Todas funções que foram afetadas pelas alterações também foram adaptadas (Código 14).

Código 14 – Atualização das funções que realizam conversão de ponteiro para inteiro.

```
1  // sendsys.c, linha 114
2  void send_g_msg(int destination,                        /* Where to send to */
3                  char *func(int plx, void * arg, char *t), /* Test function */
4                  void * arg,                               /* Argument to test
   ↪  */
5                  char *text)                              /* Text to send */
6
7  // sendsys.c, linha 210
8  char *check_send_msg(int plx, void * arg, char *t)
9  {
10     struct _send_msg_box *b = (struct _send_msg_box *)arg;
11
12     if (test_rcv(plx, b->mode, b->min, b->max, b->x1, b->x2))
13         return t;
14     return NULL;
15 }
16
17 // sendsys.c, linha 220
18 void send_msg(int destination,   /* Where to send to */
19               int mode,          /* Flags to control sending */
20               int min,           /* Minimum level of recipient */
21               int max,           /* Maximum level of recipient */
22               int x1,            /* Do not send to him */
23               int x2,            /* Nor to him */
24               char *format, ...) /* Format with args -> text to send */
25 {
26     struct _send_msg_box b;
```

```
27      send_g_msg(destination, check_send_msg, &b, bf);
28  }
29
30  // sendsys.c, linha 259
31  void sendf(int destination, char *format, ...)
32  {
33      send_g_msg(destination, NULL, NULL, b);
34  }
35
36  // sendsys.c, linha 262
37  void gsendf(int destination,
38          char *func(int plx, void * arg, char *text),
39          void * arg,
40          char *format, ...)
```

Fonte: The Dyrt Project (1995 - 1999) (com adaptações)

Não foram feitas mais alterações ao código do Dyrt. Após a correção da conversão de área de memória para inteiro, o jogo parou de dar erro ao ser executado, porém demonstrou erros com a tentativa de conexão de clientes. O acesso utilizando a ferramenta `telnet` quebra a aplicação de imediato. Não houve tentativa de depuração para descobrir o motivo desse novo erro.

4.2 SimpleMUD

Esta seção tem como objetivo apresentar o SimpleMUD e relatar o processo de evolução e manutenção de seu código. A Seção 4.2.1 descreve de maneira resumida o funcionamento do SimpleMUD, explicando como entrar no jogo e informando alguns comandos iniciais para os jogadores.

A Seção 4.2.2 apresenta as alterações realizadas para que o código do SimpleMUD pudesse ser executado. Após conseguir executar o SimpleMUD e validar que seria possível trabalhar com sua base de código, foi iniciado o planejamento de como seria realizada a melhoria do software, conforme apresentado na Seção 3.2. A Seção 4.2.4 apresenta os artefatos desenvolvidos para a realização do Scrum Solo. As seções 4.2.5 a 4.2.10 abordam os resultados alcançados na organização do repositório, integração, implantação, planejamento de testes, planejamento de melhorias e modelagem do banco de dados. A partir disso foi possível realizar as análises de código necessárias para comparar a qualidade do código antes e depois das alterações realizadas durante o desenvolvimento desse trabalho.

Os dados gerados por essas análises podem ser divididos em três etapas distintas. A primeira etapa, denominada “pré-melhorias”, refere-se aos dados gerados antes da implementação de quaisquer alterações destinadas a aprimorar o software. A segunda etapa,

chamada "melhorias não planejadas", diz respeito aos dados gerados após as alterações descritas nas Seções 4.2.3 e 4.2.11, as quais foram realizadas durante a primeira etapa deste trabalho (Seção 3.1). Por fim, a terceira etapa, denominada "melhorias planejadas", abrange os dados gerados após a conclusão da segunda etapa deste trabalho (Seção 3.2), a qual englobou as alterações descritas nas Seções 4.2.12 a 4.2.14. A Tabela 1 apresenta uma síntese desses dados, comparando cada uma das etapas mencionadas.

Tabela 1 – Síntese da análise de código do SimpleMUD e bibliotecas

		Pré-melhorias	**Melhorias não planejadas**	**Melhorias planejadas**
GCC	BasicLib	59	59	3
	SocketLib	63	63	6
	ThreadLib	4	4	0
	SimpleMUD	70	570	30
CppCheck	BasicLib	31	31	24
	SocketLib	21	17	17
	ThreadLib	1	1	3
	SimpleMUD	101	81	29
Clang-Tidy	BasicLib	448	152	444
	SocketLib	392	196	374
	ThreadLib	49	14	56
	SimpleMUD	1096	570	2760

Fonte: Autor

Para ser possível resumir as informações dos relatórios gerados pelas ferramentas de análise de código em uma única tabela, foi realizada uma contagem das palavras-chave específicas em cada relatório. Essas palavras-chave permitem ter uma noção da quantidade de problemas relatados em cada um. No caso do GCC, foi contabilizada a quantidade de ocorrências da palavra "`warning:`", que é emitida antes de cada *warning*. No Cppcheck, foi contabilizada a quantidade de ocorrências da palavra "`</error>`", e no Clang-Tidy foi utilizada a palavra "`- DiagnosticName:`". Para uma análise mais detalhada, é possível acessar esses relatórios no repositório do projeto pelo link: <https://github.com/UnMUD/UnMUD/tree/main/Logs>.

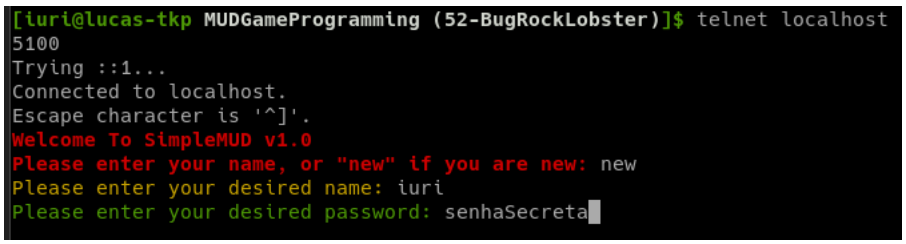
4.2.1 O jogo

O SimpleMUD, como o nome diz, é um MUD simples, que não possui história, mas que permite ao jogador explorar um pequeno mapa, enfrentar uma variedade de inimigos e encontrar ou comprar diversos items. Para se conectar ao jogo é necessário conexão com a rede onde o jogo está disponibilizado e que o computador possua uma ferramente de rede de computadores que permita a leitura e gravação de dados em conexões de rede utilizando os protocolos *Transmission Control Protocol* (TCP) ou *User Datagram Protocol*

(UDP), como a NetCat ou a Telnet. Tendo a ferramenta instalada é possível realizar a conexão utilizando o *Internet Protocol* (IP) e porta do servidor onde o jogo está rodando.

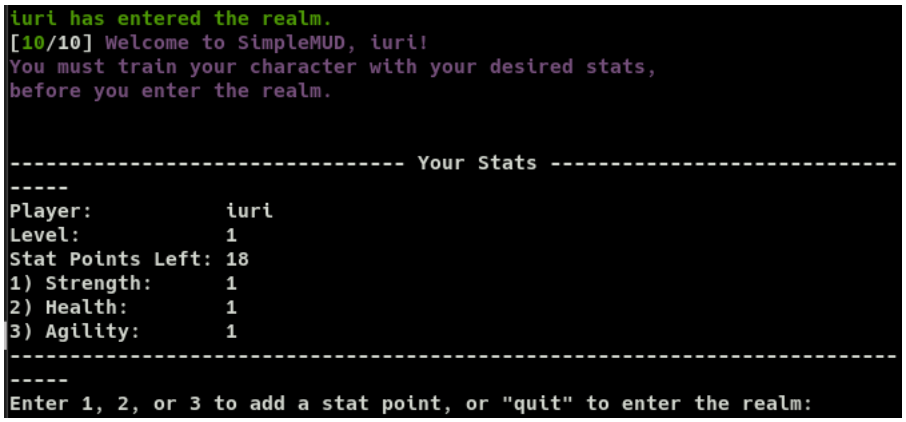
A Figura 4 apresenta a tela inicial do SimpleMUD. Ao se conectar ao jogo é possível acessar o mundo por meio de uma conta já existente ou criar uma conta nova, enviando a palavra *"new"*. Caso uma nova conta seja criada, é necessário adicionar pontos de atributos para o jogador, dividindo-os entre força, agilidade e vida (Figura 5). Esses atributos podem ser divididos igualmente em cenários onde o jogador não possui familiaridade com o estilo do jogo.

Figura 4 – Tela inicial do SimpleMUD.



Fonte: Autor

Figura 5 – Adição de atributos.

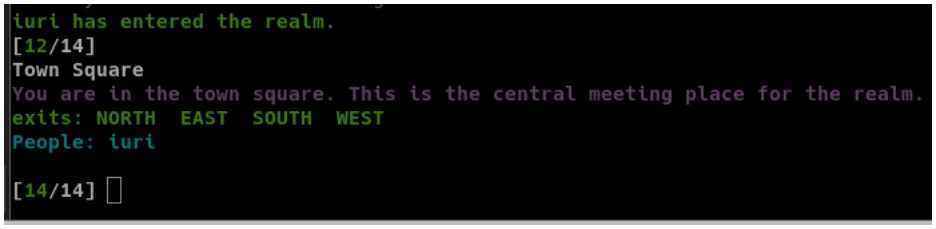


Fonte: Autor

Após distribuir os pontos de atributo o mundo virtual é apresentado para o jogador, que surge em *"Town Square"*, o centro da cidade inicial, ou na última posição onde esteve, caso ele esteja se conectando com uma conta já existente. Os locais do mapa são representados por um nome, seguido de uma descrição e as direções para onde o jogador pode andar (norte, sul, oeste ou leste). Também são listados os jogadores e inimigos que estão naquela posição do mapa. A Figura 6 apresenta o texto de descrição uma posição do mapa.

A partir desse ponto o jogador está livre para explorar o mundo conforme desejar. Para se movimentar basta enviar a direção que deseja ir, como *"North"* para ir para o

Figura 6 – Descrição de local do mapa.

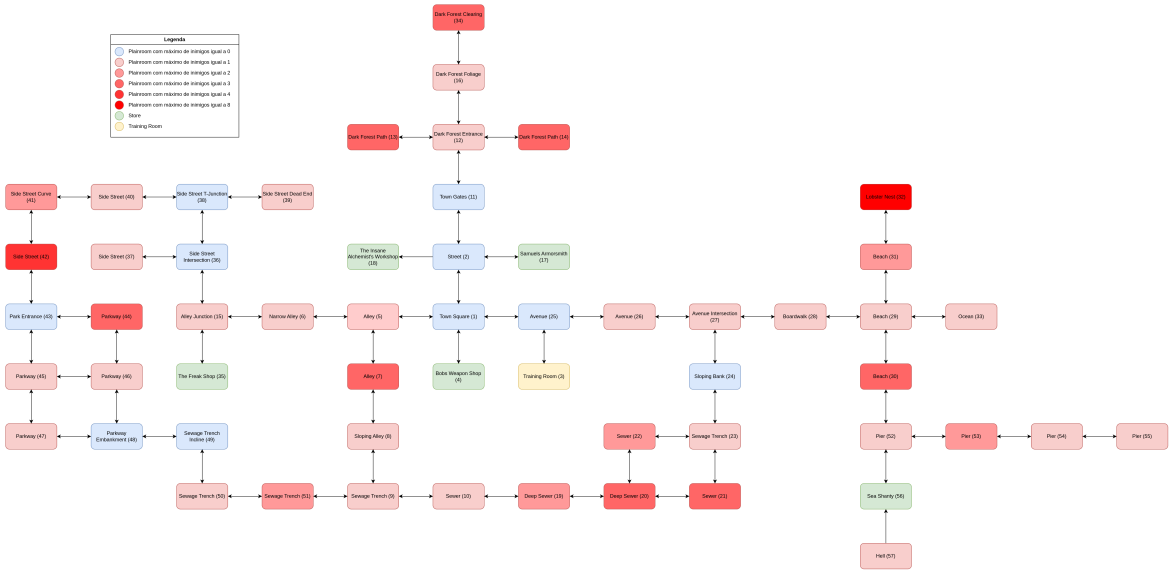


Fonte: Autor

norte. Outros comandos úteis são o *"attack"* que ataca inimigos próximos e o *"help"* que lista todos comandos do jogo. Por fim, caso o jogador queira encerrar sua sessão, basta enviar o comando *"quit"* que ele será desconectado do servidor.

Para auxiliar na exploração do jogo foi desenvolvido um mapa do mundo do SimpleMUD, que lista todos locais existentes (Figura 7). As áreas vermelhas indicam locais onde é possível encontrar inimigos, sendo que quanto mais forte o vermelho, mais inimigos podem ser encontrados naquela posição. As áreas roxas são referentes a locais seguros, sem inimigos. As lojas são representadas pela cor verde e a sala de treinamento é representada pela cor amarela.

Figura 7 – Mapa do SimpleMUD.



Fonte: Autor

4.2.2 Correções para compilação

As primeiras alterações realizadas no código foram feitas para corrigir erros de compilação. A documentação do SimpleMUD define três passos para compilação do programa:

1. `make libs`
2. `make simplemud`
3. `make link`

O primeiro passo serve para compilar as bibliotecas externas fornecidas pelo autor, o segundo para compilar o MUD e o terceiro para linkar os arquivos objeto gerados. Ao tentar executá-los diversos erros foram levantados.

4.2.2.1 make libs

O SimpleMUD utiliza três bibliotecas em sua codificação. Uma para funções básicas, `BasicLib`, uma para lidar com *sockets*, `SocketLib`, e uma para *threads*, `ThreadLib`. Ao executar o comando de compilação foram encontrados erros na biblioteca básica e na de *sockets*.

4.2.2.1.1 BasicLib

Os erros levantados na biblioteca básica se concentravam em torno das funções matemáticas de logaritmo, cosseno e seno, que são utilizadas no arquivo `BasicLibRandom.h` (Código 15).

Código 15 – Exemplo de erros da biblioteca básica.

```
../Libraries/BasicLib/BasicLibRandom.h: In member function 'double
↪ BasicLib::normal_generator<inclusive, generator>::operator()()':
../Libraries/BasicLib/BasicLibRandom.h:132:31: error: there are no arguments to
↪ 'log' that depend on a template parameter, so a declaration of 'log' must
↪ be available [-fpermissive]
  132 |         m_rho = sqrt(-2 * log(1- m_rho2));
      |                               ^~~
../Libraries/BasicLib/BasicLibRandom.h:132:31: note: (if you use
↪ '-fpermissive', G++ will accept your code, but allowing the use of an
↪ undeclared name is deprecated)
../Libraries/BasicLib/BasicLibRandom.h:140:36: error: there are no arguments to
↪ 'cos' that depend on a template parameter, so a declaration of 'cos' must
↪ be available [-fpermissive]
  140 | turn m_rho * ( m_valid ? cos(2 * pi * m_rho1) : sin( 2 * pi * m_rho1))
    ↪ * m_sigma + m_mean;
      |                            ^~~
../Libraries/BasicLib/BasicLibRandom.h:140:59: error: there are no arguments to
↪ 'sin' that depend on a template parameter, so a declaration of 'sin' must
↪ be available [-fpermissive]
  140 | ? cos(2 * pi * m_rho1) : sin( 2 * pi * m_rho1)) * m_sigma + m_mean;
```

```
11      |                                  ^~~
```

Fonte: Autor

Para corrigi-los foi necessário incluir a biblioteca `cmath`, que faz parte da biblioteca GNU ISO C++.

4.2.2.1.2 SocketLib

A biblioteca de *sockets* apresentou dois erros recorrentes. O primeiro foi com a função `memset()`, no arquivo `SocketLibSocket.cpp`, cujo a correção foi indicada pelo compilador (Código 16). Após a inclusão da biblioteca `cstring` os erros foram resolvidos.

Código 16 – Exemplo de erro levantado pela função `memset()`.

```
../Libraries/SocketLib/SocketLibSocket.cpp: In member function 'void
↪  SocketLib::DataSocket::Connect(SocketLib::ipaddress, SocketLib::port)':
../Libraries/SocketLib/SocketLibSocket.cpp:141:9: error: 'memset' was not
↪  declared in this scope
  141 |         memset( &(m_remoteinfo.sin_zero), 0, 8 );
      |         ^~~~~~
../Libraries/SocketLib/SocketLibSocket.cpp:9:1: note: 'memset' is defined in
↪  header '<cstring>'; did you forget to '#include <cstring>'?
    8 | #include "SocketLibSocket.h"
  +++ |+#include <cstring>
    9 |
```

Fonte: Autor

O segundo erro foi resultado da declaração do tipo `clistitr`, apresentado no Código 17. Como o próprio compilador aponta, a correção do erro é possível a partir da adição da palavra chave **`typename`** antes da definição de `clistitr` e após a palavra chave **`typedef`**. Essa correção foi necessária devido à dependência no escopo do tipo definido[1]. O Código 18 apresenta a revisão feita.

Código 17 – Exemplo de erro levantado pela declaração de `clistitr`.

```
In file included from ../Libraries/SocketLib/Connection.h:17,
                 from ../Libraries/SocketLib/Telnet.h:14,
                 from ../Libraries/SocketLib/Telnet.cpp:9:
../Libraries/SocketLib/ConnectionManager.h: At global scope:
```

[1] Uma explicação resumida desse erro pode ser encontrada no StackOverflow, Nested templates with dependent scope.

```
../Libraries/SocketLib/ConnectionManager.h:37:13: error: need 'typename' before
↪ 'std::__cxx11::list<SocketLib::Connection<protocol> >::iterator' because
↪ 'std::__cxx11::list<SocketLib::Connection<protocol> >' is a dependent scope
   37 |     typedef std::list< Connection<protocol> >::iterator clistitr;
      |             ^~~
      |             typename
../Libraries/SocketLib/ConnectionManager.h:110:17: error: 'clistitr' has not
↪ been declared
  110 |     void Close( clistitr p_itr );
      |                 ^~~~~~~
```

Fonte: Autor

Código 18 – Revisão da declaração de `clistitr`.

```
// ConnectionManager.h, linha 37
// Código com erro
typedef std::list< Connection<protocol> >::iterator clistitr;

// Código revisado
typedef typename std::list< Connection<protocol> >::iterator clistitr;
```

Fonte: Penton (2003) (com adaptações)

4.2.2.2 make simplemud

Assim como a compilação das bibliotecas, o comando para compilar o MUD também apresentou erros ao ser executado. Alguns semelhantes aos já resolvidos, como o exemplo apresentado no Código 19, o que tornou a correção mais rápida.

Código 19 – Exemplo de erro conhecido.

```
SimpleMUD/EntityDatabase.h:47:25: error: need 'typename' before
↪ 'SimpleMUD::EntityDatabase<datatype>::container::iterator' because
↪ 'SimpleMUD::EntityDatabase<datatype>::container' is a dependent scope
 47 |         iterator( const container::iterator& p_itr ) // copy constructor
    |                         ^~~~~~~~~
    |                         typename
```

Fonte: Autor

Todos erros levantados na primeira tentativa de utilização do comando se referiam ao arquivo `EntityDatabase.h` e, em sua maioria, aconteciam ao tentar declarar o iterador do tipo `container`, Código 20, definido pelo próprio autor.

Código 20 – Exemplo de erro ao declarar iterador do tipo `container`.

```
SimpleMUD/EntityDatabase.h: In constructor
↪  'SimpleMUD::EntityDatabase<datatype>::iterator::iterator(const int&)':
SimpleMUD/EntityDatabase.h:49:34: error: 'itr' was not declared in this scope
  49 |             container::iterator& itr = *this; // also needed because VC6
  ↪  sucks
     |                                  ^~~
```

Fonte: Autor

Dentre as diversas possibilidades de correção, a opção adotada foi a de definir um novo tipo para esse iterador, o tipo `containeritr`, seguindo o padrão adotado nas bibliotecas utilizadas. Essa escolha foi feita devido a quantidade de locais onde o iterador era utilizado no código analisado. Após a definição do novo tipo, todos acessos à `container::iterator` e `std::map<entityid,datatype>::iterator` foram atualizados. O resultado das alterações pode ser visto nos Códigos 21 e 22.

Código 21 – Acesso ao iterador do tipo `container` antes da correção.

```
// EntityDatabase.h
template< class datatype >
class EntityDatabase
{
public:
    typedef std::map<entityid, datatype> container;
    // --------------------------------------------------------------------
    //  The inner iterator class, used to iterate through the database.
    // --------------------------------------------------------------------
    class iterator : public container::iterator
    {
    public:

        // ----------------------------------------------------------------
        // NOTE: the constructors are needed as a result of VC6 sucking.
        // Have I mentioned that VC6 sucks yet?
        // ----------------------------------------------------------------
        iterator() {};  // default constructor
        iterator( const container::iterator& p_itr ) // copy constructor
        {
            container::iterator& itr = *this; // also needed because VC6 sucks
            itr = p_itr;
        }
```

Fonte: Penton (2003)

Código 22 – Correção do acesso ao iterador do tipo `container`.

```
1  // EntityDatabase.h
2  template< class datatype >
3  class EntityDatabase
4  {
5  public:
6
7      typedef std::map<entityid, datatype> container;
8      typedef typename std::map<entityid, datatype>::iterator containeritr;
9
10     // ---------------------------------------------------------------------
11     //  The inner iterator class, used to iterate through the database.
12     // ---------------------------------------------------------------------
13     class iterator : public containeritr
14     {
15     public:
16
17         // -----------------------------------------------------------------
18         // NOTE: the constructors are needed as a result of VC6 sucking.
19         // Have I mentioned that VC6 sucks yet?
20         // -----------------------------------------------------------------
21         iterator() {};  // default constructor
22         iterator( const containeritr& p_itr ) // copy constructor
23         {
24             containeritr& itr = *this; // also needed because VC6 sucks
25             itr = p_itr;
26         }
```

Fonte: Penton (2003) (com adaptações)

Após corrigidos os erros levantados pelo arquivo `EntityDatabase.h`, uma segunda tentatiza de executar o comando foi feita e resultou em novos erros (Código 23).

Código 23 – Erros da sintaxe de template.

```
1  g++ -I../Libraries *.cpp -c;
2  g++ -I../Libraries ./SimpleMUD/*.cpp -c;
3  ./SimpleMUD/EnemyDatabase.cpp:25:28: error: specializing member
   ↪  'SimpleMUD::EntityDatabaseVector<SimpleMUD::EnemyTemplate>::m_vector'
   ↪  requires 'template<>' syntax
4     25 | std::vector<EnemyTemplate>
      ↪  EntityDatabaseVector<EnemyTemplate>::m_vector;
5        |                            ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
./SimpleMUD/EnemyDatabase.cpp:28:27: error: specializing member
↪ 'SimpleMUD::EntityDatabase<SimpleMUD::Enemy>::m_map' requires 'template<>'
↪ syntax
   28 | std::map<entityid, Enemy> EntityDatabase<Enemy>::m_map;
   |                           ^~~~~~~~~~~~~~~~~~~
./SimpleMUD/ItemDatabase.cpp:20:26: error: specializing member
↪ 'SimpleMUD::EntityDatabase<SimpleMUD::Item>::m_map' requires 'template<>'
↪ syntax
   20 | std::map<entityid, Item> EntityDatabase<Item>::m_map;
      |                          ^~~~~~~~~~~~~~~~~~
./SimpleMUD/PlayerDatabase.cpp:24:28: error: specializing member
↪ 'SimpleMUD::EntityDatabase<SimpleMUD::Player>::m_map' requires 'template<>'
↪ syntax
   24 | std::map<entityid, Player> EntityDatabase<Player>::m_map;
      |                            ^~~~~~~~~~~~~~~~~~~~
./SimpleMUD/RoomDatabase.cpp:25:19: error: specializing member
↪ 'SimpleMUD::EntityDatabaseVector<SimpleMUD::Room>::m_vector' requires
↪ 'template<>' syntax
   25 | std::vector<Room> EntityDatabaseVector<Room>::m_vector;
      |                   ^~~~~~~~~~~~~~~~~~~~~~~~
./SimpleMUD/StoreDatabase.cpp:20:27: error: specializing member
↪ 'SimpleMUD::EntityDatabase<SimpleMUD::Store>::m_map' requires 'template<>'
↪ syntax
   20 | std::map<entityid, Store> EntityDatabase<Store>::m_map;
      |                           ^~~~~~~~~~~~~~~~~~~
make: *** [makefile:24: simplemud] Error 1
```

Fonte: Autor

Os novos erros foram decorrentes da forma como as variáveis estáticas das entidades do bancos de dados foram declaradas nas classes que possuíam herança com `EntityDatabase` ou com `EntityDatabaseVector`. Um mesmo erro para vários arquivos, como é possível observar no Código 23.

A solução adotada foi a adição da palavra chave `template<>`, utilizando como argumento a mesma classe passada para o template da classe base[2], antes da declaração da variável. O Código 24 apresenta o resultado da correção.

Código 24 – Correção da declaração das variáveis estáticas.

```
// EnemyDatabase.cpp, linha 27
// Código original
// declare the static map of the enemy instance database.
std::map<entityid, Enemy> EntityDatabase<Enemy>::m_map;
```

[2] Solução encontrada na documentação da IBM Static data members and templates (C++ only).

```
5
6 // Código corrigido
7 // declare the static map of the enemy instance database.
8 template< class Enemy >
9 std::map<entityid, Enemy> EntityDatabase<Enemy>::m_map;
```

Fonte: Penton (2003) (com adaptações)

O comando `make simplemud` funcionou corretamente após as alterações, assim como o comando `make link`. Não foram encontrados novos erros.

Após conseguir completar o processo de compilação do código e execução do MUD, foi iniciada uma análise em busca de trechos do código que poderiam ser atualizados para seguirem os padrões estabelecidos no *C++ Core Guidelines* (STROUSTRUP; SUTTER et al., 2022).

4.2.3 Alterações arquiteturais no consumo do Banco de Dados

O banco de dados do SimpleMUD consiste em um conjunto de arquivos que persistem os dados de jogadores, itens, monstros dentre outros. Para o consumo e manipulação das informações salvas nesses arquivos, Penton utiliza uma abordagem com dicionários, vetores, classes e funções estáticas, que gerenciam esses dados durante o jogo. Dessa forma ele conseguiu uma estrutura que o permitiu acessar os dados de forma global e sem a necessidade de gerar uma instância dos objetos das classes criadas(PENTON, 2003).

> *Because the classes are static, you should be able to access a single database within the game globally, without worrying about instantiating it* (PENTON, 2003) [3].

No entanto, essa é uma abordagem ingênua de implementação do padrão Singleton e desencadeia alguns problemas, como a ordem de inicialização de objetos estáticos globais em unidades de compilação diferentes ser indefinida, o que pode resultarem um objeto global referindo-se a outro quando este ainda não foi inicializado (NESTERUK, 2018).

A implementação clássica do padrão Singleton, sugerida por Nesteruk, é uma forma de melhorar a implementação feita por Peton, tornando-a mais segura e sem perder sua intenção inicial. Essa foi a opção adotada para melhorar o código analisado.

Foram feitas alterações significativas nas classes `EntityDatabase`, `EntityDatabaseVector`, `EnemyTemplateDatabase`, `EnemyDatabase`, `ItemDatabase`, `PlayerDatabase`, `RoomDatabase` e `StoreDatabase`. Os Códigos 25-26 apresentam um exemplo de como

[3] Como as classes são estáticas, você deve conseguir acessar um único banco de dados dentro do jogo globalmente, sem se preocupar em instanciá-lo. (tradução livre)

foram feitas as implementações de Penton, enquanto os Códigos 27-28 apresentam as modificações realizadas.

Código 25 – Cabeçalho das classes `EnemyTemplateDatabase` e `EnemyDatabase` originais.

```
1  // EnemyDatabase.h
2  class EnemyTemplateDatabase : public EntityDatabaseVector<EnemyTemplate>
3  {
4  public:
5      static void Load();
6  };  // end class EnemyTemplateDatabase
7
8
9  class EnemyDatabase : public EntityDatabase<Enemy>
10 {
11 public:
12     static void Create( entityid p_template, room p_room );
13     static void Delete( enemy p_enemy );
14     static void Load();
15     static void Save();
16
17 };  // end class EnemyDatabase
```

Fonte: Penton (2003)

Código 26 – Classes `EnemyTemplateDatabase` e `EnemyDatabase` originais.

```
1  // EnemyDatabase.cpp
2  // declare the static vector of the enemy template database.
3  std::vector<EnemyTemplate> EntityDatabaseVector<EnemyTemplate>::m_vector;
4
5  // declare the static map of the enemy instance database.
6  std::map<entityid, Enemy> EntityDatabase<Enemy>::m_map;
```

Fonte: Penton (2003)

A variável `m_map` é inicialmente declarada na classe `EntityDatabase`, que serve como classe base para o banco de dados, no entanto, por ser estática, é necessário redefini-la em toda classe derivada para evitar problemas de linkagem ao compilar os arquivos. O mesmo vale para variável `m_vector`, da classe `EntityDatabaseVector`. Outra peculiaridade dessa implementação original é a necessidade de todas funções serem estáticas, uma vez que não há instancia das classes. Ambas situações foram extinguidas com a nova implementação.

Código 27 – Cabeçalho das classes `EnemyTemplateDatabase` e `EnemyDatabase` atualizadas.

```
1  // EnemyDatabase.h
2  class EnemyTemplateDatabase : public EntityDatabaseVector<EnemyTemplate>
3  {
4  public:
5      static EnemyTemplateDatabase& GetInstance();
6      EnemyTemplateDatabase(EnemyTemplateDatabase const&) = delete;
7      EnemyTemplateDatabase(EnemyTemplateDatabase&&) = delete;
8      EnemyTemplateDatabase& operator=(EnemyTemplateDatabase const&) = delete;
9      EnemyTemplateDatabase& operator=(EnemyTemplateDatabase &&) = delete;
10
11     void Load();
12
13 private:
14     EnemyTemplateDatabase(){}
15 };  // end class EnemyTemplateDatabase
16
17
18 class EnemyDatabase : public EntityDatabase<Enemy>
19 {
20 public:
21     static EnemyDatabase& GetInstance();
22     EnemyDatabase(EnemyDatabase const&) = delete;
23     EnemyDatabase(EnemyDatabase&&) = delete;
24     EnemyDatabase& operator=(EnemyDatabase const&) = delete;
25     EnemyDatabase& operator=(EnemyDatabase &&) = delete;
26
27     void Create( entityid p_template, room p_room );
28     void Delete( enemy p_enemy );
29     void Load();
30     void Save();
31
32 private:
33     EnemyDatabase(){}
34 };  // end class EnemyDatabase
```

Fonte: Penton (2003) (com adaptações)

Código 28 – Classes `EnemyTemplateDatabase` e `EnemyDatabase` atualizadas.

```
1  // EnemyDatabase.cpp
2  EnemyTemplateDatabase& EnemyTemplateDatabase::GetInstance()
3  {
4      static EnemyTemplateDatabase enemyTemplateDatabase;
```

```
5     return enemyTemplateDatabase;
6 }
7
8 EnemyDatabase& EnemyDatabase::GetInstance()
9 {
10    static EnemyDatabase enemyDatabase;
11    return enemyDatabase;
12 }
```

Fonte: Penton (2003) (com adaptações)

Com a instanciação das classes, as funções e atributos deixaram de ser estáticos e os acessos as entidades do banco de dados passou ser feito a partir da chamada `GetInstance()` (Código 29). Diversas alterações pequenas foram feitas em outros arquivos para adaptá-los ao novo padrão adotado.

Código 29 – Exemplo de deleção de um inimigo antes e depois da atualização das entidades.

```
1 // Game.cpp, linha 1296
2 // Código original
3 EnemyDatabase::Delete( p_enemy );
4
5 // Código pós alterações
6 EnemyDatabase::GetInstance().Delete( p_enemy );
```

Fonte: Penton (2003) (com adaptações)

4.2.4 Scrum Solo: Artefatos

Pagotto et al. (2016) definem em seu artigo etapas que devem ser seguidas para a realização do Scrum Solo e, como resultado dessas etapas, diversos artefatos de software são gerados, como o próprio *Backlog* do Produto.

Os primeiros artefatos gerado nesse processo foram os quadros dos requisitos funcionais e não funcionais do projeto. Nesses quadros é possível encontrar o identificador do requisito, sua descrição e de qual técnica de elicitação de requisitos ele foi retirado (*Backward From*). O Quadro 1 apresenta os requisitos funcionais, enquanto o Quadro 2 apresenta os requisitos não funcionais.

A partir da modelagem dos requisitos elicitados, foi possível desenvolver o *Backlog* do Produto, dividido em Épico, *Feature* e História de Usuário. Além dessas três colunas principais, o *Backlog* também possui as colunas: Estimativa de tempo (dias); Tempo real (dias); Data de Inserção; Data de Seleção; e Prioridade. As cinco colunas adicionais tiveram origem a partir dos critérios explorados pelo Scrum Solo, para o *Backlog* do Produto

Quadro 1 – Requisitos funcionais

ID	**Descrição**	***Backward From***
RF01	Documentação do projeto	Introspecção
RF02	Documentação do código	Introspecção, *Brainstorming*
RF03	Repositório organizado	Introspecção
RF04	Ambiente de desenvolvimento	Introspecção, *Brainstorming*
RF05	Plano de testes	Introspecção, *Brainstorming*
RF06	Plano de implantação	Introspecção, *Brainstorming*
RF07	Documentação do jogo	Introspecção
RF08	Utilização de um banco de dados	*Brainstorming*
RF09	Ferramentas de socialização no jogo	Introspecção
RF10	Diversidade de items	Introspecção
RF11	Diversidade de habilidades	Introspecção
RF12	Diversidade de monstros	Introspecção

Fonte: Autor.

Quadro 2 – Requisitos não funcionais

ID	**Descrição**	***Backward From***
RNF01	Documentações com linguagem acessível	Introspecção
RNF02	Segurança no armazenamento de dados	Introspecção, *Brainstorming*
RNF03	Segurança na implantação do sistema	*Brainstorming*
RNF04	Interface prática e intuitiva para o jogo	Introspecção
RNF05	Experiência de jogo interessante	Introspecção
RNF06	Progresso de jogo bem ritmado (*well-paced*)	Introspecção

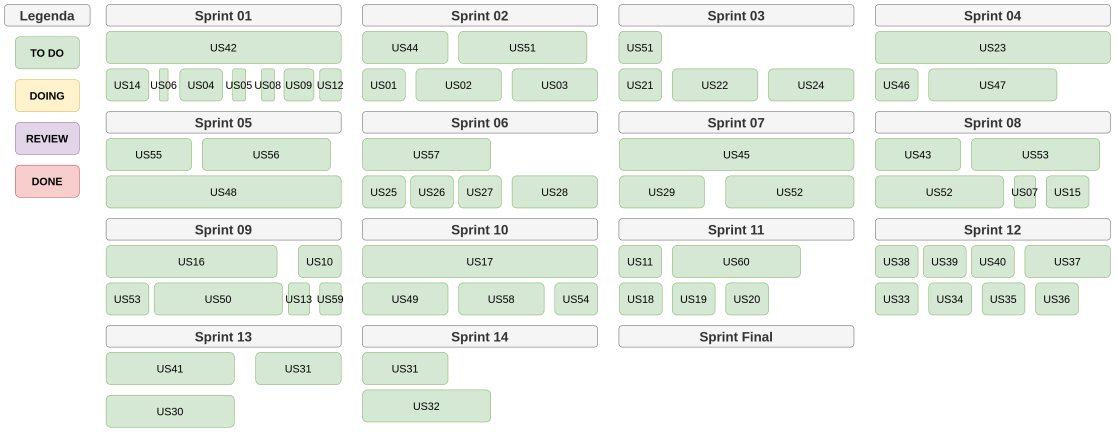
Fonte: Autor.

e o *Backlog* da *Sprint*, e a necessidade de priorizar as atividades a serem realizadas. O Apêndice B apresenta o resultado alcançado na elaboração desse artefato.

Por fim, após a realização da priorização das histórias de usuário com a técnica MoSCoW, foi possível criar um cronograma com base na estimativa de tempo necessário para realização das tarefas principais. Esse cronograma foi representado na forma de um *Roadmap*.

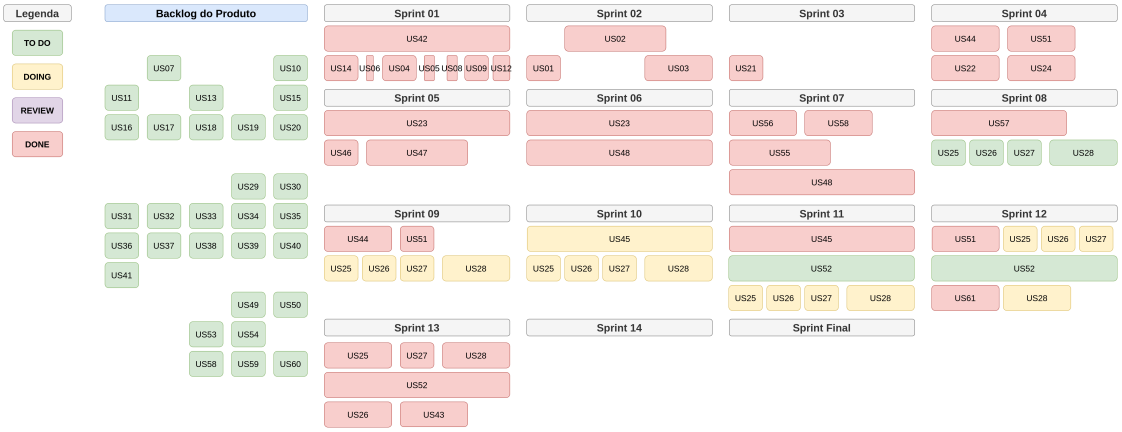
É importante destacar que, embora o *Roadmap* já defina em qual sprint as histórias de usuário serão executadas, esse planejamento passou por revisões a cada sprint e o *Backlog* da *Sprint* foi confirmado após a aprovação do professor orientador, conforme define a metodologia seguida. Portanto, esse artefato teve duas versões, uma inicial baseada nas estimativas (Figura 8) e uma final que representa o trabalho desenvolvido (Figura 9). Nas figuras apresentadas é possível diferenciar histórias de usuários concluídas, em revisão, em andamento e que estão por fazer pelas cores vermelho, roxo, amarelo e verde, respectivamente.

Figura 8 – Roadmap do planejamento estimado



Fonte: Autor

Figura 9 – Roadmap do trabalho desenvolvido



Fonte: Autor

4.2.5 Organização do repositório e aplicação de boas práticas

O primeiro passo para organização do repositório foi criar uma organização do GitHub[4] para que futuramente outras pessoas possam colaborar com o projeto como mantenedores. O nome atribuído a organização e ao repositório principal foi UnMUD, junção das siglas UnB e MUD. A criação da organização permitiu a utilização da aba de projetos do GitHub, onde é mantido um quadro Kanban utilizado para organização do *backlog* da *sprint*.

Após a migração dos arquivos do repositório antigo para o novo, foi realizada a escrita do README e foi criado uma página de apresentação do projeto utilizando o Docsify e o GitHub Pages. O README do repositório pode ser acessado pelo link <https://github.com/UnMUD/UnMUD>, enquanto a página de apresentação do projeto

[4] <https://docs.github.com/pt/organizations/collaborating-with-groups-in-organizations/about-organizations>

pode ser acessada pelo link <https://unmud.github.io/UnMUD/#/>.

A licença escolhida para o projeto foi a AGPL-3.0, que consiste em uma licença que permite a alteração do código fonte, porém com algumas condições. Apesar de permitir uso comercial, realização de modificações, entre outros, a licença requer que o software mantenha a mesma licença e considera o fornecimento do serviço pela rede como distribuição, o que a difere da licença GPL-3.0, que não possui essa condicional. O texto original da licença pode ser lido no Anexo A.

Por fim, o Guia de Contribuição, os modelos de *issue* e o modelo de *pull request* foram elaborados utilizando como referência as informações apresentadas em Brasileo (2022), assim como outros repositórios do GitHub que possuem uma comunidade estabelecida. Os modelos de *issue* e *pull request* também foram adicionados ao GitHub para que colaboradores possam utilizá-los no repositório do projeto.

Todos documentos referentes ao projeto podem ser acessados na página do UnMUD, disponível no GitHub Pages, pelo link citado anteriormente.

4.2.6 Integração

Para realização da integração foram criados dois fluxos de trabalho diferentes: um para *pull requests* e outro para *commits* realizados na *branch* principal. Além das verificações realizadas pelos fluxos de trabalho no GitHub Actions, também foi utilizado o serviço SonarCloud, que verifica a qualidade de código de repositórios.

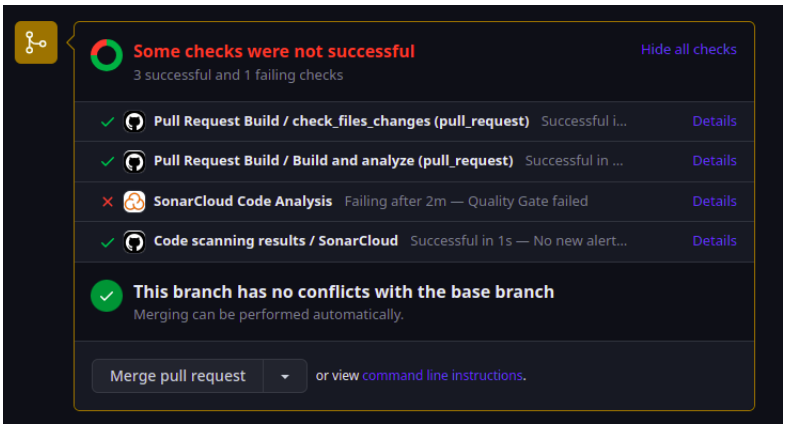
A diferença entre os fluxos de trabalho reside no fato de que o fluxo para a *branch* principal adiciona os arquivos de análise de qualidade de código ao repositório após sua execução. Além disso, ambos os fluxos verificam se houve alguma alteração no código dentro da pasta "MUDGameProgramming". Caso haja modificações, eles realizam as seguintes etapas:

1. instalam as dependências necessárias para a compilação do MUD no servidor disponibilizado;
2. rodam os testes unitários e geram os relatórios de cobertura de testes;
3. executam as ferramentas de análise estática, `Cppcheck` e `Clang-Tidy`;
4. enviam os dados gerados para o SonarQube.

Caso todas etapas sejam realizadas com sucesso, o *pull request* ou *commit* é dado como válido e pode ser integrado na *branch* principal. As Figuras 10 e 11 apresentam um fluxo de trabalho do GitHub Actions bem sucedido e o retorno obtido do serviço SonarQube após a análise do código. O erro apresentado nas figuras citadas se deu pelas

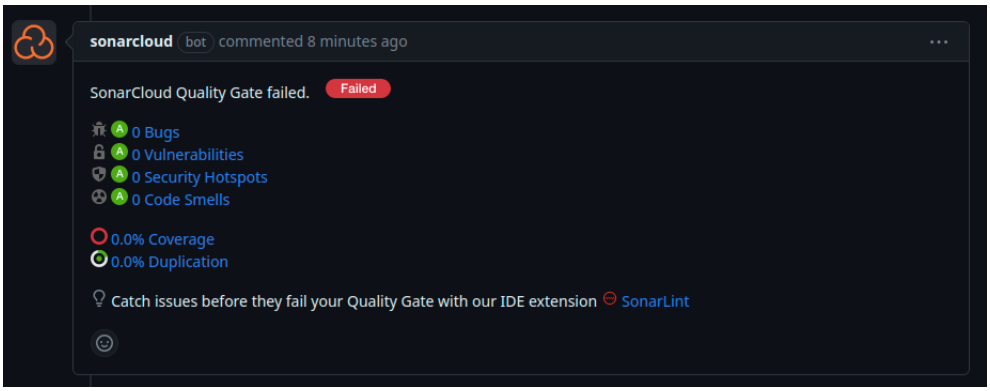
métricas de qualidade utilizadas pelo Sonar para aprovar alterações no código. Como ainda existem melhorias a serem realizadas, as condições estabelecidas pela ferramenta não foram aceitas. No entanto, é possível ver as marcações de sucesso nos fluxos realizados pelo próprio GitHub.

Figura 10 – *Pull Request* com fluxo de trabalho concluído com sucesso



Fonte: Autor

Figura 11 – Retorno da análise de código realizada pelo SonarCloud



Fonte: Autor

4.2.7 Implantação

Para a realização da implantação do sistema, foram realizados testes em uma máquina `t2.micro` instanciada nos servidores virtuais disponibilizados pela empresa Amazon por meio da *Amazon Web Services* (AWS). A máquina `t2.micro` possui um processador Intel Xeon escalável de até 3,3 GHz (Haswell E5-2676 v3 ou Broadwell E5-2686 v4), uma CPU e 1 GiB de memória.

Após a instalação das ferramentas `git` e `docker`, necessárias para executar o MUD, o código foi compilado, executado e testado com 5 usuários jogando simultaneamente. O momento de maior esforço da CPU ocorreu durante a compilação do código, onde ela

alcançou 26,76% de consumo. Durante os momentos de conexão dos jogadores e o tempo em que estiveram no jogo, a CPU não foi afetada significativamente, sendo o menor valor atingido nesse período de 4,86% e o maior valor de 5,54%. Durante esse teste, também foi verificado que o jogo fica disponível para conexão a partir do momento em que é compilado e executado dentro do contêiner `docker`, sem a necessidade de expor manualmente a porta definida para o MUD.

A principal alteração necessária para implantar o código foi a criação de variáveis de ambiente para guardar dados sigilosos, como senhas de jogadores administradores. Para isso, foi criado um arquivo `.env` no formato apresentado no Código 30. Essas variáveis são passadas para o programa por meio do arquivo `docker-compose.yml`. As adaptações realizadas para o uso dessas variáveis podem ser vistas no *pull request* "US28 - Implantar Sistema"[5].

Código 30 – Exemplo de arquivo contendo variáveis de ambiente.

```
1  # Database environment variables
2  PG_HOST=db
3  PG_PORT=5432
4  PG_DATABASE=unmud
5  PG_USER=postgres
6  PG_PASSWORD=postgres
7
8  # SQL environment variables
9  IURI_PASSWORD=123456789
10 MAURICIO_PASSWORD=123456789
11
12 # MUD environment variables
13 MUD_PORT=5100
```

Fonte: Autor

Com base nos testes e nas alterações realizadas, foi possível definir 8 passos necessários para a implantação do código desenvolvido:

1. instalar o Git;
2. clonar o repositório do MUD com o comando `$git clone https://github.com /UnMUD/UnMUD.git`;
3. instalar o Docker e o Docker Compose;
4. entrar na pasta `MUDGameProgramming` com o comando `$cd UnMUD/MUDGameProgramming`;

[5] <https://github.com/UnMUD/UnMUD/pull/50>

5. criar o arquivo `.env` e adicionar as variáveis de ambiente, com base no arquivo `.env_example`;

6. criar e executar os contêineres com o comando `$sudo docker compose up -build -d`;

7. compilar o MUD com o comando `$docker compose exec unmud bash Scripts/compile.sh`;

8. executar o binário gerado com o comando `$docker compose exec unmud bash Scripts/run.sh`.

4.2.8 Planejamento de Testes

O documento de Planejamento de Testes pode ser dividido em quatro partes: Introdução; Materiais de referência para o planejamento e execução dos testes; Ambiente de testes; e Detalhamento dos testes. As duas partes mais importantes são a definição do ambiente de testes e o detalhamento dos testes.

A seção do ambiente de teste descreve como um desenvolvedor pode utilizar o ambiente criado para a realização dos testes unitários usando o Docker e o Docker Compose. A seção de detalhamento apresenta as funções que serão testadas, fornecendo uma breve descrição do que cada função deve realizar e sugestões de casos de teste que podem ser implementados para elas. Para facilitar a visualização, o detalhamento dos testes foi dividido em quatro subtópicos: `BasicLib`, `SocketLib`, `ThreadLib` e `SimpleMUD`. No entanto, o tópico referente ao `SimpleMUD` não foi elaborado devido à complexidade das funções encontradas nessa parte do código e ao tempo necessário para planejar os testes para elas, uma vez que envolvem conexão de rede e dependem, em sua maioria, do relógio interno do sistema ou de entradas do usuário. A Figura 12 apresenta um exemplo de como está documentado o detalhamento de testes.

4.2.9 Planejamento de Melhorias

A primeira etapa do planejamento de melhorias consiste na explicação do sistema de classificação desenvolvido para categorizar os erros e *warnings* identificados no programa. A Tabela 2 apresenta os níveis de dificuldade de resolução de cada erro/*warning*, com base nos valores atribuídos aos fatores classificatórios de conhecimento, tempo e complexidade, sendo o conhecimento e a complexidade baseados no conhecimento empírico adquirido durante o desenvolvimento desse trabalho, enquanto o tempo se refere a estimativa de tempo necessário para resolução do erro ou *warning*. Após a elaboração da tabela de classificação foi possível listar todos problemas encontrados a atribuir seus respectivos níveis. O Apêndice C apresenta uma tabela com essa listagem.

Figura 12 – Detalhamento de testes da função GetFileList da BasicLib

BasicLib/BasicLibFiles.cpp

Função GetFileList

```
filelist GetFileList( const std::string p_directory )
```

Descrição: Recebe uma `string` contendo o nome do diretório e retorna um `set` de `strings` contendo o nome dos arquivos desse diretório

Sugestão de teste:

- Diretório não existente
- Diretório vazio
- Diretório com poucos arquivos dentro
- Diretório com muitos arquivos dentro

Fonte: Autor

Por fim, estabeleceu-se de forma minuciosa o planejamento das melhorias a serem implementadas. As correções foram abordadas com base na classificação e na frequência de cada erro e *warning*, seguindo uma ordem que priorizou as questões mais simples antes das mais complexas e os problemas mais recorrentes antes dos menos frequentes, atribuindo especial atenção aqueles apontados pelo GCC.

4.2.10 Modelagem do Banco de Dados

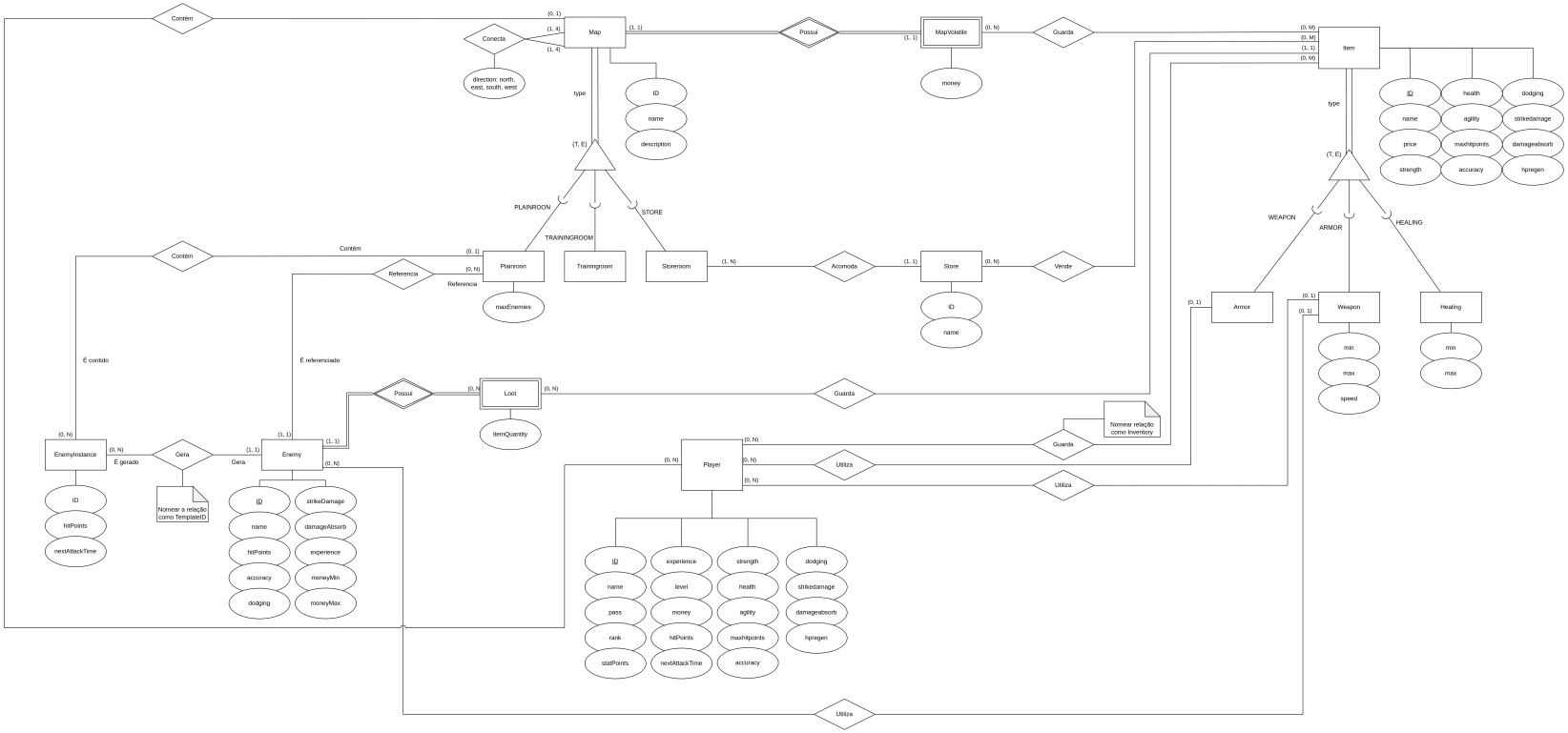
A modelagem do banco de dados foi baseada nas informações sobre a entidades do jogo fornecidas por Penton (2003) e nos arquivos de texto presentes no código disponibilizado. A partir disso foi possível criar o Diagrama Entidade-Relacionamento (DER) apresentado na Figura 13.

Tabela 2 – Sistema de classificação de erros e *warnings*

Nível	**Conhecimento** (1 - 5)	**Tempo** (min)	**Complexidade** (1 - 5)
A	1, 2	180 < x	4, 5
B	1, 2	180 < x	3
C	1, 2	180 < x	1, 2
B	1, 2	90 < x < 180	4, 5
C	1, 2	90 < x < 180	3
D	1, 2	90 < x < 180	1, 2
C	1, 2	30 < x < 90	4, 5
D	1, 2	30 < x < 90	3
E	1, 2	30 < x < 90	1, 2
D	1, 2	x < 30	4, 5
E	1, 2	x < 30	3
F	1, 2	x < 30	1, 2
B	3	180 < x	4, 5
C	3	180 < x	3
D	3	180 < x	1, 2
C	3	90 < x < 180	4, 5
D	3	90 < x < 180	3
E	3	90 < x < 180	1, 2
D	3	30 < x < 90	4, 5
E	3	30 < x < 90	3
F	3	30 < x < 90	1, 2
E	3	x < 30	4, 5
F	3	x < 30	3
G	3	x < 30	1, 2
C	4, 5	180 < x	4, 5
D	4, 5	180 < x	3
E	4, 5	180 < x	1, 2
D	4, 5	90 < x < 180	4, 5
E	4, 5	90 < x < 180	3
F	4, 5	90 < x < 180	1, 2
E	4, 5	30 < x < 90	4, 5
F	4, 5	30 < x < 90	3
G	4, 5	30 < x < 90	1, 2
F	4, 5	x < 30	4, 5
G	4, 5	x < 30	3
H	4, 5	x < 30	1, 2

Fonte: Autor

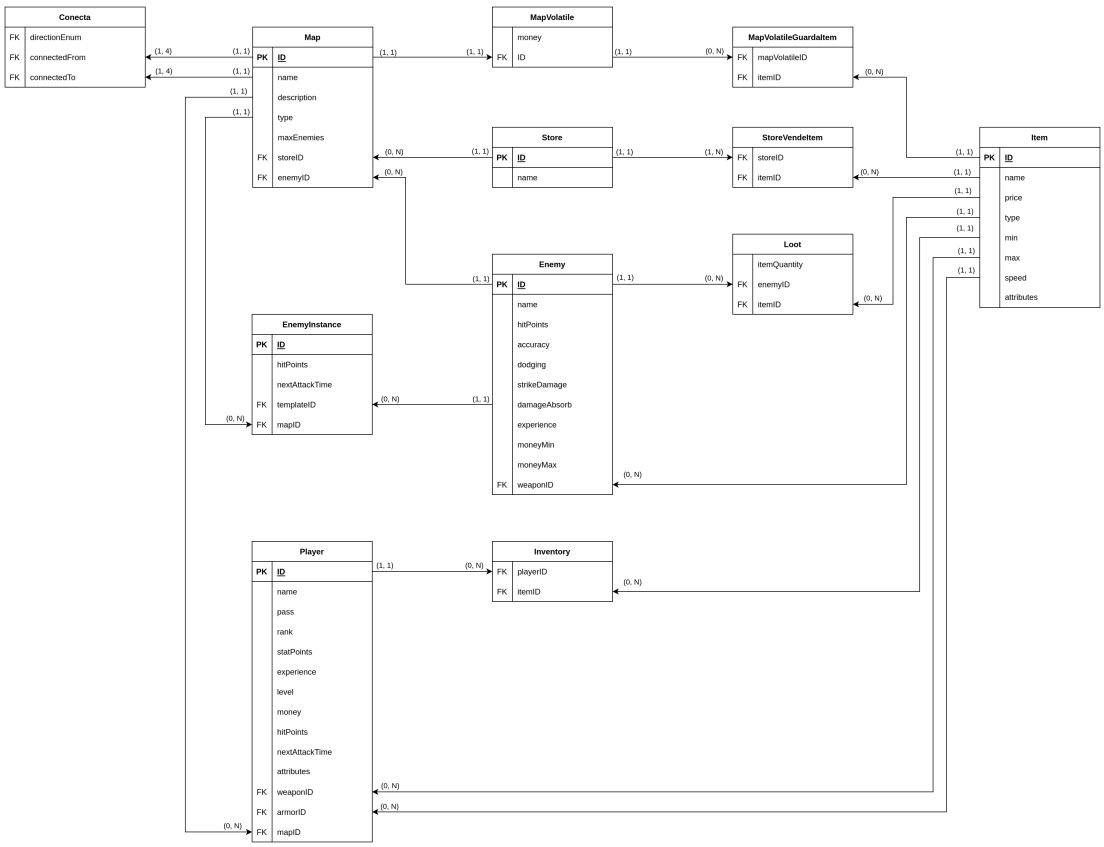
Figura 13 – Diagrama Entidade-Relacionamento do SimpleMUD



Fonte: Autor

O Diagrama Lógico (Fig. 14) foi desenvolvido a partir do DER, baseado na transformação entre modelos de Barcelar (2012) e nas regras de mapeamento de generalizações e especializações de Bagui (2009), em especial regra número 3, que é adequada para especializações exclusivas que possuem um atributo condicional de tipo. Também foi feita uma versão adaptada do diagrama, voltada para o PostgreSQL, com o objetivo de facilitar a visualização da implementação física do banco de dados (Figura 15).

Figura 14 – Diagrama Lógico do SimpleMUD

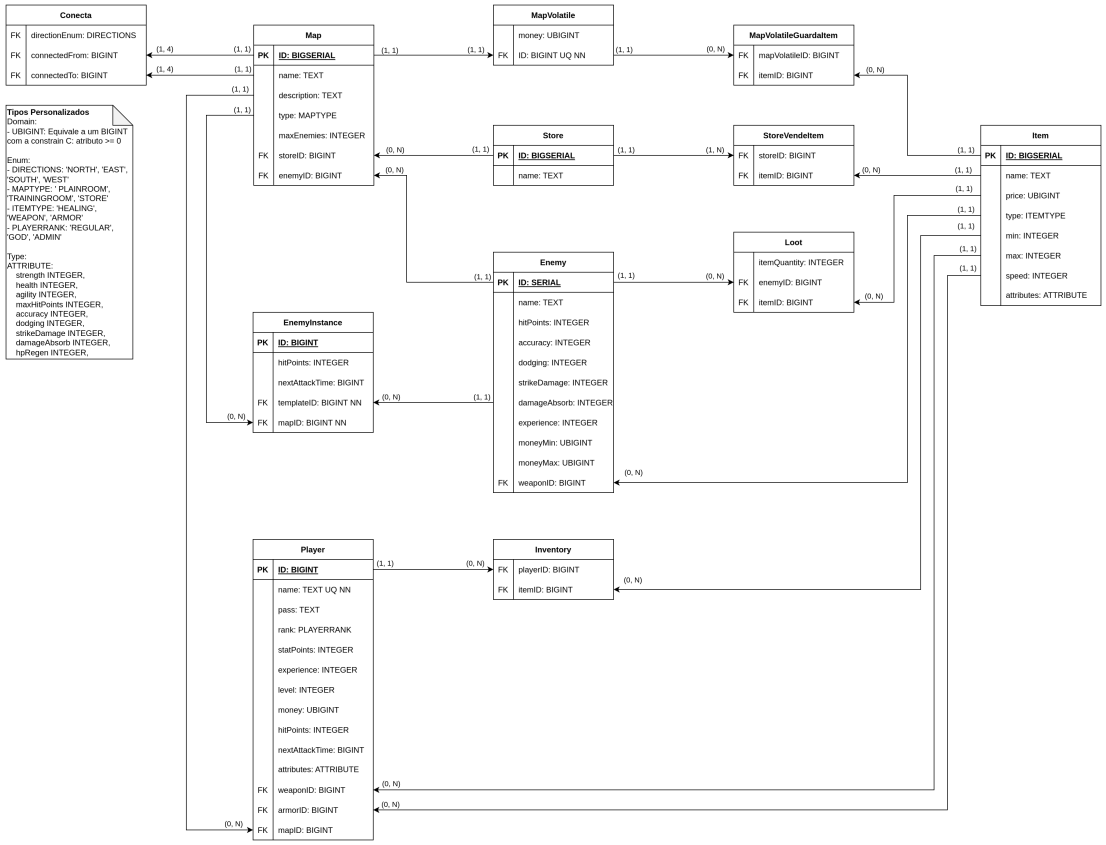


Fonte: Autor

Por fim, a parte referente a implementação do banco de dados foi realizada por meio de três *scripts*, referentes a criação das tabelas, população das tuplas e criação de procedimentos, funções e *triggers*. Com o auxílio da ferramenta `docker`, o banco de dados foi criado e populado durante a geração do contêiner, caso ele não exista.

O Código 31 apresenta um exemplo de código SQL para criação e população de uma tabela, nesse caso, a tabela `Item`. É possível observar no código a enumeração `ITEMTYPE` e o tipo composto `ATTRIBUTE` que foram adaptações necessárias para que os *scripts* se tornassem mais condizentes com os arquivos de texto iniciais do MUD, porém sem perder sua manutenibilidade. A enumeração é utilizada para definir o tipo de item, na especialização, enquanto o tipo composto foi criado para que os atributos comuns entre items e *players* pudessem ser atualizados sem a necessidade de alterar as duas tabelas.

Figura 15 – Diagrama Lógico para PostgreSQL



Fonte: Autor

Código 31 – Exemplo de código SQL para criação e população de tabela.

```sql
CREATE TYPE ITEMTYPE AS ENUM ('HEALING', 'WEAPON', 'ARMOR');

CREATE TYPE ATTRIBUTE AS (
    strength INTEGER, health INTEGER, agility INTEGER, maxHitPoints INTEGER,
    accuracy INTEGER, dodging INTEGER, strikeDamage INTEGER, damageAbsorb
    INTEGER, hpRegen INTEGER
);


CREATE TABLE Item(
    id BIGSERIAL,
    name TEXT,
    type ITEMTYPE,
    min INTEGER,
    max INTEGER,
    speed INTEGER,
    price UBIGINT,
    attributes ATTRIBUTE,
```

```
17
18      CONSTRAINT Item_PK PRIMARY KEY(id)
19  );
20
21  INSERT INTO Item VALUES
22      (35, 'Minor Healing Potion', 'HEALING', 2, 5, 0, 10, ROW(0, 0, 0, 0, 0, 0,
    ↪  0, 0, 0)),
23      (40, 'Rusty Knife', 'WEAPON', 1, 4, 1, 5, ROW(0, 0, 0, 0, 5, 0, 0, 0, 0)),
24      (45, 'Cloth Armor', 'ARMOR', 0, 0, 0, 10, ROW(0, 0, 0, 0, 0, 10, 0, 0, 0))
```

Fonte: Autor

4.2.11 Alterações não planejadas

Durante a refatoração, foram encontrados trechos de código que poderiam ser atualizados, tanto para a utilização de novas funcionalidades adicionadas no C++ quanto para se obter um código mais limpo.

4.2.11.1 EntityDatabase iterator

A classe `EntityDatabase` tem como objetivo servir de modelo para as entidades do banco de dados. Nela ocorre o agrupamento de um dicionário e funções de busca, assim como a abstração dessa estrutura em um novo tipo, chamado `container` e seu iterador, `containeritr`. Por fim, uma classe `iterator` é criada para sobrescrever o iterador padrão da estrutura definida na STL. O Código 32 apresenta partes dessa classe.

Código 32 – Trechos da classe `EntityDatabase`.

```
1  // EntityDatabase.h
2  template< class datatype >
3  class EntityDatabase
4  {
5  public:
6
7      typedef std::map<entityid, datatype> container;
8      typedef typename std::map<entityid, datatype>::iterator containeritr;
9      // --------------------------------------------------------------------
10     //  The inner iterator class, used to iterate through the database.
11     // --------------------------------------------------------------------
12     class iterator : public containeritr{}
13
14 protected:
15     std::map<entityid, datatype> m_map;
16 };  // end class EntityDatabase
```

Fonte: Penton (2003)

Na construção do iterador interno ocorre a sobrescrita de dois operadores, o operador * e o ->, utilizados para deferência e seleção de elemento por ponteiro, respectivamente. No entanto, as novas funcionalidades atribuídas para esses operadores utilizam o operador -> da classe base, porém o acesso a esse operador é perdido, uma vez que ele é sobrescrito. Para contornar essa situação, Peton cria um cópia do ponteiro da classe derivada e a "converte" para classe base, seguido pela chamada do operador desejado, o que resultou em um código não muito claro e de difícil interpretação (Código 33).

Código 33 – Iterador interno da classes `EntityDatabase`.

```
1  // EntityDatabase.h
2  class iterator : public containeritr
3  {
4  public:
5      // ------------------------------------------------------------------
6      // NOTE: the constructors are needed as a result of VC6 sucking.
7      // Have I mentioned that VC6 sucks yet?
8      // ------------------------------------------------------------------
9      iterator() {};  // default constructor
10     iterator( const containeritr& p_itr ) // copy constructor
11     {
12         containeritr& itr = *this; // also needed because VC6 sucks
13         itr = p_itr;
14     }
15     // ------------------------------------------------------------------
16     //  "dereferences" the iterator to return a reference to the
17     //  object that it points to.
18     // ------------------------------------------------------------------
19     inline datatype& operator*()
20     {
21         containeritr& itr = *this; // also needed because VC6 sucks
22         return itr->second;
23     }
24     // ------------------------------------------------------------------
25     //  the "pointer-to-member" operator, which will allow you to use it on
26     //  iterators like this: itr->Function();
27     // ------------------------------------------------------------------
28     inline datatype* operator->()
29     {
30         containeritr& itr = *this; // also needed because VC6 sucks
31         return &(itr->second);
32     }
33 };  // end iterator inner class
```

Fonte: Penton (2003)

Todas as funções apresentadas foram atualizadas. Os construtores passaram a utilizar um método moderno, enquanto os operadores passaram a utilizar a função `static_cast`, que atinge o mesmo resultado de acessar os operadores da classe base, porém de uma forma mais limpa. O Código 34 apresenta a classe `iterator` refatorada.

Código 34 – Refatoração do iterador interno da classe `EntityDatabase`.

```
1  // EntityDatabase.h
2  class iterator : public containeritr
3  {
4  public:
5      iterator() {};  // default constructor
6      iterator( const containeritr& p_itr ) : containeritr( p_itr ){} // copy
   ↪  constructor
7
8      // -------------------------------------------------------------------
9      //  "dereferences" the iterator to return a reference to the
10     //  object that it points to.
11     // -------------------------------------------------------------------
12     inline datatype& operator*()
13     {
14         return static_cast<containeritr>(*this)->second;
15     }
16
17     // -------------------------------------------------------------------
18     //  the "pointer-to-member" operator, which will allow you to use it on
19     //  iterators like this: itr->Function();
20     // -------------------------------------------------------------------
21     inline datatype* operator->()
22     {
23         return &(static_cast<containeritr>(*this)->second);
24     }
25 };  // end iterator inner class
```

Fonte: Penton (2003)

4.2.11.2 Estruturas de Repetição

A partir do C++11 foi adicionada a estrutura de repetição **`for`** baseada em um intervalo definido. O *Range-based for loop* é usado como um equivalente mais legível ao laço for tradicional operando em uma faixa de valores, como todos os elementos em um contêiner (CPPREFERENCE, 2023).

A iteração por todos elementos de um contêiner é facilmente encontrada em diversos segmentos do código do SimpleMUD, como apresentado no Código 35, e podem ser atualizados utilizando o *range-based for loop*, Código 36.

Código 35 – Função `PerformHeal()` da classe `GameLoop`.

```
1  // GameLoop.cpp, linha 151
2  void GameLoop::PerformHeal()
3  {
4      PlayerDatabase::iterator itr = PlayerDatabase::GetInstance().begin();
5      while( itr != PlayerDatabase::GetInstance().end() )
6      {
7          if( itr->Active() )
8          {
9              itr->AddHitpoints( itr->GetAttr( HPREGEN ) );
10             itr->PrintStatbar( true );
11         }
12         ++itr;
13     }
14 }
```

Fonte: Penton (2003)

Código 36 – Função `PerformHeal()` da classe `GameLoop` atualizada.

```
1  // GameLoop.cpp, linha 143
2  void GameLoop::PerformHeal()
3  {
4      for(auto& player : PlayerDatabase::GetInstance()){
5          if( player.Active() )
6          {
7              player.AddHitpoints( player.GetAttr( HPREGEN ) );
8              player.PrintStatbar( true );
9          }
10     }
11 }
```

Fonte: Penton (2003) (com adaptações)

Para atualização das estruturas de repetição foi necessário a alteração de 6 arquivos, totalizando 9 funções refatoradas e a redução de 21 linhas de código.

4.2.12 Alterações planejadas

Conforme estipulado no planejamento de melhorias, as alterações realizadas tiveram início a partir dos *warnings* levantados pelo `GCC`. Durante o período estabelecido, foram corrigidos quatro itens da Lista de Erros e *Warnings* (Apêndice C): "*should be initialized in the member initialization list*", com 298 ocorrências; "*use of old-style cast*", com

227 ocorrências; "*conversion from A to B may change value*", com 78 ocorrências; e "*conversion to B from A may change the sign of the result*", com 36 ocorrências. Além das correções manuais, foram aplicadas as alterações automáticas da ferramenta `clang-format`, utilizando a flag `-i`. Todas alterações realizadas podem ser encontradas no *pull request* "US48 - Aplicar plano de melhorias"[6].

4.2.12.1 *Warning should be initialized in the member initialization list*

A resolução do *warning* "*should be initialized in the member initialization list*" é realizada a partir da adição dos atributos de uma classe ou `struct` em sua lista de inicialização, o que garante maior eficiência na atribuição desses valores. Apesar de ser um aviso de correção simples, foi necessário realizar alterações em 26 arquivos, para que todas suas ocorrências fossem solucionadas. Os Códigos 37 e 38 apresentam parte da correção realizada no arquivo `BasicLibRandom.h` da `BasicLib`. Devido a semelhança entre as alterações realizadas para solução do problema, não serão apresentados outros exemplos de alterações realizadas, no entanto, é possível acessar toda solução implementada por meio dos *commits* C_1[7] e C_2[8].

Código 37 – `Struct random_range`.

```
1  // BasicLibRandom.h, linha 119
2  //
   ↪ ------------------------------------------------------------------------
3  //  returns a random double number from [a..b, inclusivity of b is
4  //  configurable.
5  //
   ↪ ------------------------------------------------------------------------
6  template <bool inclusive, typename generator = random_percent<inclusive>>
7  struct random_range {
8    random_range(double a = 0, double b = 1) { init(a, b); }
9
10   random_range(generator &p_generator, double a = 0, double b = 1)
11       : m_generator(p_generator) {
12     init(a, b);
13   }
14
15   void init(double a, double b) {
16     m_range = b - a;
17     m_offset = a;
18   }
19 };
```

[6] <https://github.com/UnMUD/UnMUD/pull/37>
[7] <https://github.com/UnMUD/UnMUD/commit/8d8ca40f9a097d157b2f7b1f50fe7ff4cbce1e3c>
[8] <https://github.com/UnMUD/UnMUD/commit/7a1d1026f26e7cd75e53dc3a4cb11b69707a2f46>

Fonte: Penton (2003)

Código 38 – *Struct* `random_range` atualizada.

```
1  // BasicLibRandom.h, linha 112
2  // ------------------------------------------------------------------------
3  //  returns a random double number from [a..b, inclusivity of b is
4  //  configurable.
5  // ------------------------------------------------------------------------
6  template <bool inclusive, typename generator = random_percent<inclusive>>
7  struct random_range {
8      random_range(double a = 0, double b = 1)
9          : m_generator(), m_range(b - a), m_offset(a) {}
10
11     random_range(generator &p_generator, double a = 0, double b = 1)
12         : m_generator(p_generator), m_range(b - a), m_offset(a) {}
13 };
```

Fonte: Penton (2003) (com adaptações)

4.2.12.2 *Warning use of old-style cast*

O *warning* "*use of old-style cast*" ocorre quando o código em C++ contém conversões de tipo que seguem o estilo "antigo", ou seja, realizadas de acordo com a definição da linguagem C. Stroustrup, Sutter et al. (2022) estabelecem dois princípios relacionados ao uso de *cast* que, quando seguidos, evitam o problema em questão: ES.48: *Avoid casts*[9]; e ES.49: *If you must use a cast, use a named cast*[10]. Além disso, Stroustrup também define um perfil de programação voltado para a *Type-safety* (segurança de tipos), que se refere à propriedade de garantir que uma variável não seja utilizada de maneira que não esteja em conformidade com as regras estabelecidas para o tipo de sua definição. Esse perfil apresenta padrões não adequados que os programadores devem evitar ou tratar como erros.

O C++ oferece quatro operadores diferentes para realizar conversões de tipo. Segundo Cpluscplus.com (2023), esses operadores e seus propósitos são definidos da seguinte maneira:

- `dynamic_cast`: pode ser usado apenas com ponteiros e referências a objetos. Seu objetivo é garantir que o resultado da conversão de tipo seja um objeto completo válido da classe solicitada. O `dynamic_cast` também pode realizar conversões entre ponteiros nulos, mesmo entre ponteiros de classes não relacionadas, e pode converter ponteiros de qualquer tipo para ponteiros de tipo `void*`.

[9] Evitar conversões de tipo
[10] Se realmente necessário utilizar um *cast*, utilize um *cast* nomeado

- `static_cast`: pode realizar conversões entre ponteiros de classes relacionadas, não apenas da classe derivada para a base, mas também da base para a derivada. Isso garante que pelo menos as classes sejam compatíveis se o objeto adequado for convertido, mas nenhuma verificação de segurança é realizada em tempo de execução para verificar se o objeto sendo convertido é de fato um objeto completo do tipo de destino. O `static_cast` também pode ser usado para realizar qualquer outra conversão não ponteiro que também poderia ser realizada implicitamente, como, por exemplo, a conversão padrão entre tipos fundamentais.
- `reinterpret_cast`: converte qualquer tipo de ponteiro para qualquer outro tipo de ponteiro, mesmo de classes não relacionadas. O resultado da operação é uma simples cópia binária do valor de um ponteiro para o outro. Todas as conversões de ponteiro são permitidas: nem o conteúdo apontado nem o próprio tipo de ponteiro são verificados.
- `const_cast`: manipula a constância de um objeto, seja para definir ou remover. Por exemplo, para passar um argumento constante para uma função que espera um parâmetro não constante.

Para correção dos problemas levantados pelo `GCC` foram utilizados o `static_cast` e o `reinterpret_cast`. Os Códigos 39 a 42 apresentam um exemplo da utilização de cada um deles, respectivamente. No total 15 arquivos foram alterados para que todos *warnings* fossem solucionados. Os *commits* C_3[11] e C_4[12] apresentam a resolução completa.

Código 39 – Exemplo de `static_cast` no estilo "antigo".

```
1 // BasicLibFunctions.h ; Linha 70
2 template <typename type>
3 inline int percent(const type &first, const type &second) {
4   return (int)(100.0 * (double)first / (double)second);
5 }
```

Fonte: Penton (2003)

Código 40 – Exemplo de correção utilizando `static_cast`.

```
1 // BasicLibFunctions.h ; Linha 70
2 template <typename type>
3 inline int percent(const type &first, const type &second) {
4   return static_cast<int>(100.0 * static_cast<double>(first) /
5                           static_cast<double>(second));
```

[11] <https://github.com/UnMUD/UnMUD/commit/f0329a1fd2241bd2b916202324274dd173953336>
[12] <https://github.com/UnMUD/UnMUD/commit/212ad7bc2ce6021c1f866d6c6a437585a47aa778>

```
6 }
```

Fonte: Penton (2003) (com adaptações)

Código 41 – Exemplo de `reinterpret_cast` no estilo "antigo".

```
1 // SocketLibSocket.cpp ; Linha 69
2 // the socket is blocking by default
3 if (p_socket != -1) {
4   socklen_t s = sizeof(m_localinfo);
5   getsockname(p_socket, (sockaddr *)(&m_localinfo), &s);
6 }
```

Fonte: Penton (2003)

Código 42 – Exemplo de correção utilizando `reinterpret_cast`.

```
1 // SocketLibSocket.cpp ; Linha 69
2 // the socket is blocking by default
3 if (p_socket != -1) {
4   socklen_t s = sizeof(m_localinfo);
5   getsockname(p_socket, reinterpret_cast<sockaddr *>(&m_localinfo), &s);
6 }
```

Fonte: Penton (2003) (com adaptações)

4.2.12.3 *Warning conversion from A to B may change value*

Quando a conversão de um tipo de variável para outro tipo pode alterar o valor atribuído a ela, o *warning* "*conversion from A to B may change value*" é emitido. Para corrigir esse problema, existem algumas soluções que variam desde reestruturar o código para eliminar a necessidade de conversão até explicitar a conversão para informar ao compilador que ela é intencional e suprimir o aviso. Ambas abordagens foram utilizadas para solucionar todos problemas informados.

No total 13 arquivos foram alterados para resolver todos *warnings* apontados pelo compilador. Os Códigos 43 e 44 apresentam um exemplo de solução que alterou a estrutura do código, modificando o tipo de variável retornado pela função, enquanto os Códigos 45 e 46 apresentam uma solução onde não era possível alterar o tipo do atributo, então foi necessário realizar uma conversão explícita. Os *commits* C_5[13] e C_6[14] apresentam todas alterações realizadas para correção dos *warnings*.

[13] <https://github.com/UnMUD/UnMUD/commit/1ead6bd03bfdd47c99adb2bdb58e60f7be2fb7ea>
[14] <https://github.com/UnMUD/UnMUD/commit/2a72402ce999f078f69f85e1c51baf8e756fc6ee>

O primeiro exemplo apresenta a função `ASCIIToHex`, que possuía em seu cabeçalho a informação de que seu retorno seria do tipo **`char`**. No entanto, quando ocorre uma operação entre dois caracteres diferentes em C++, o resultado é um valor inteiro. Isso causa o surgimento do *warning* analisado, uma vez que variáveis do tipo **`char`** podem armazenar valores de 0 a 255, enquanto inteiros de quatro *bytes* têm um intervalo de −2147483648 a 2147483647. Para corrigir esse aviso, foi alterado o tipo de retorno especificado no cabeçalho da função para **`int`**. Outra opção para corrigir esse problema seria converter a variável retornada pela função. Contudo, essa solução ainda causaria problemas no código, uma vez que os caracteres retornados não seriam significativos.

Código 43 – Exemplo de conversão onde pode haver perda de informação e é possível alterar a estrutura do código.

```
1  // BasicLibFunctions.h , Linha 76
2  inline char ASCIIToHex(char c) {
3    if (c >= '0' && c <= '9')
4      return c - '0';
5    if (c >= 'A' && c <= 'F')
6      return c - 'A' + 10;
7    if (c >= 'a' && c <= 'a')
8      return c - 'a' + 10;
9    return 0;
10 }
```

Fonte: Penton (2003)

Código 44 – Exemplo de correção para conversão onde poderia haver perda de informação e era possível alterar a estrutura do código.

```
1  // BasicLibFunctions.h , Linha 76
2  inline int ASCIIToHex(char c) {
3    if (c >= '0' && c <= '9')
4      return c - '0';
5    if (c >= 'A' && c <= 'F')
6      return c - 'A' + 10;
7    if (c >= 'a' && c <= 'a')
8      return c - 'a' + 10;
9    return 0;
10 }
```

Fonte: Penton (2003) (com adaptações)

Código 45 – Exemplo de conversão onde pode haver perda de informação e não é possível alterar a estrutura do código.

```
1 // SocketLibSocket.cpp ; Linha 69
2 // the socket is blocking by default
3 if (p_socket != -1) {
4   socklen_t s = sizeof(m_localinfo);
5   getsockname(p_socket, (sockaddr *)(&m_localinfo), &s);
6 }
```

Fonte: Penton (2003)

Código 46 – Exemplo de correção para conversão onde poderia haver perda de informação e não era possível alterar a estrutura do código.

```
1 // SocketLibSocket.cpp ; Linha 69
2 // the socket is blocking by default
3 if (p_socket != -1) {
4     socklen_t s = sizeof(m_localinfo);
5     getsockname(p_socket, reinterpret_cast<sockaddr *>(&m_localinfo), &s);
6 }
```

Fonte: Penton (2003) (com adaptações)

4.2.12.4 *Warning: conversion to B from A may change the sign of the result*

O *warning* "*conversion to B from A may change the sign of the result*" é semelhante ao *warning* "*conversion from A to B may change value*" análisado previamente, porém ele avalia especificamente a variação de sinais em conversões realizadas. As abordagens de correção dos *warnings* também são semelhantes.

Os Códigos 47 e 48 apresentam uma das alterações realizadas para solução dos avisos levantados. Os *commits* C_7[15] e C_8[16] apresentam todas alterações realizadas. Treze arquivos foram alterados no total.

Código 47 – Função `TrimWhitespace` original.

```
1 // BasicLibString.cpp , Linha 45
2 std::string TrimWhitespace(const std::string &p_string) {
3   size_t wsf;
4   size_t wsb;
5
6   // trim the front
```

[15] <https://github.com/UnMUD/UnMUD/commit/0a6354b974e03b2b2a1279fee22ee4f50a60fbc7>
[16] <https://github.com/UnMUD/UnMUD/commit/022fa18580ed35c0cdc11c2d8166f5778c0566ec>

```
7    wsf = p_string.find_first_not_of(WHITESPACE);
8    wsb = p_string.find_last_not_of(WHITESPACE);
9
10   if (wsf == std::string::npos) {
11     wsf = 0;
12     wsb = -1;
13   }
14
15   return p_string.substr(wsf, wsb - wsf + 1);
16 }
```

Fonte: Penton (2003)

Código 48 – Função `TrimWhitespace` corrigida.

```
1  // BasicLibString.cpp , Linha 45
2  std::string TrimWhitespace(const std::string &p_string) {
3    size_t wsf;
4    size_t wsb;
5    // trim the front
6    wsf = p_string.find_first_not_of(WHITESPACE);
7    wsb = p_string.find_last_not_of(WHITESPACE);
8    if (wsf == std::string::npos) {
9      return p_string.substr(0, 0);
10   }
11   return p_string.substr(wsf, wsb - wsf + 1);
12 }
```

Fonte: Penton (2003) (com adaptações)

4.2.13 Testes unitários

Para a implementação dos testes unitários, foi necessário realizar alterações na estrutura de pastas do projeto. Foi adicionada a pasta `Tests`, na qual foram armazenados os arquivos `.cpp` referentes aos testes. A fim de compilar e executar os testes unitários, foi necessário atualizar o `makefile` do projeto (Códigos 49 e 50). Além disso, foi elaborado um *script* em `bash` para executar o `gcovr` e salvar o arquivo de *log* gerado (Código 51).

Código 49 – `makefile` original.

```
1  // makefile, linha 22
2  simplemud: $(wildcard *.cpp)
3  $(CXX) $(CFLAGS) *.cpp -c;
4  $(CXX) $(CFLAGS) $(SIMPLEMUDDIR)/*.cpp -c;
5
```

```
6  main: $(wildcard *.cpp)
7          $(CXX) $(CFLAGS) *.cpp -c;
8
9  link: $(wildcard *.cpp)
10         $(CXX) $(CFLAGS) *.o $(LIBS) -o UnMUD.out
```

Fonte: Penton (2003)

Código 50 – `makefile` atualizado para compilar e executar testes unitários.

```
1  // makefile, linha 22
2  unmud: $(wildcard *.cpp)
3  $(CXX) $(CFLAGS) *.cpp -c --coverage;
4  $(CXX) $(CFLAGS) $(UNMUDDIR)/*.cpp -c --coverage;
5
6  main: $(wildcard *.cpp)
7          $(CXX) $(CFLAGS) *.cpp -c;
8
9  link: $(wildcard *.cpp)
10         $(CXX) $(CFLAGS) \
11         $$(find . -maxdepth 1 -name '*.o' ! -name '$(TESTSMAINFILE).o' \
12         ! -name '$(TESTFILEPREFIX)*' -exec basename {} \;) \
13         $(LIBS) -o $(UNMUDBINFILE) --coverage
14
15 run:
16         rm -f *.gcda
17         ./$(UNMUDBINFILE)
18
19 tests: $(wildcard *.cpp)
20         $(CXX) $(CFLAGS) $(TESTSDIR)/*.cpp -c --coverage;
21         $(CXX) $(CFLAGS) \
22         $$(find . -maxdepth 1 -name '*.o' ! -name '$(UNMUDMAINFILE).o' -exec
           ↪  basename {} \;) \
23         $(TESTSLIBS) $(LIBS) -o $(TESTSBINFILE) --coverage
24
25 run_tests:
26         rm -f *.gcda
27         ./$(TESTSBINFILE)
```

Fonte: Penton (2003) (com adaptações)

Código 51 – Script de execução de testes e análise de cobertura de testes.

```
1  // tests.sh
2  COVERAGEDIR=../AnalysisLogs/GCOVR
3  DATE=$(date +"%Y-%m-%d-%H-%M-%S")
4
5  cd SimpleMUD
6
7  if make tests ; then
8      echo "Tests compiled"
9  else
10     errorNum=$?
11     echo "Error $errorNum compiling Tests..."
12     exit $errorNum
13 fi
14
15 make run_tests
16
17 testsRet=$?
18
19 gcovr --fail-under-line 80 -r ../ -e ../Tests/  \
20     --html $COVERAGEDIR/$DATE-coverage.html --sonarqube
       ↪  $COVERAGEDIR/coverage.xml
21
22 echo "Running tests finished with return $testsRet"
23 exit $errorNum
```

Fonte: Autor

A implementação dos testes foi realizada a partir das bibliotecas desenvolvidas e a cobertura de *branch* alcançada no período disponibilizado para realização da tarefas foi de 3,1%, enquanto a cobertura de linhas foi de 10,0%. O *pull request* "US23 - Aplicar planejamento de testes"[17] apresenta todos testes implementados. Os Apêndices D e E apresentam o relatório de testes antes e depois da implementação realizada.

De maneira geral, é possível afirmar que um software deve apresentar uma cobertura de *branch* de 80% para garantir um alto nível de confiabilidade (MALAIYA et al., 1994). No entanto, durante o desenvolvimento deste trabalho, não foi possível alcançar esse valor, ficando como um objetivo a ser alcançado em trabalhos futuros. É importante ressaltar que a determinação da porcentagem de cobertura de testes necessária e a escolha do tipo de cobertura mais adequado para esse sistema requerem estudos mais aprofundados nessa área.

[17] <https://github.com/UnMUD/UnMUD/pull/32>

4.2.14 Atualização do Banco de Dados

A atualização do banco de dados foi efetuada utilizando a biblioteca `libpqxx`, uma API em C++ para o gerenciamento de banco de dados PostgreSQL. Por meio dessa biblioteca, foi possível estabelecer a conexão com o banco de dados e executar consultas SQL. A fim de evitar a exposição de informações confidenciais, foram empregadas variáveis de ambiente para definir o *host*, o nome do banco de dados, o nome de usuário para autenticação, a senha e a porta utilizada pelo banco. Essas variáveis de ambiente permitiram a criação da conexão com o banco sem a necessidade de especificar esses atributos no código.

Após a definição da configuração da biblioteca, foram realizadas atualizações no código com o propósito de integrar o PostgreSQL ao MUD e remover os arquivos que estavam sendo utilizados para persistir os dados. Para isso, foi necessário atualizar as funções de *Load* e *Save* das classes relacionadas às entidades do software, incluindo inimigos, itens, mapa, jogadores e lojas. Essas entidades eram anteriormente populadas com base nas informações contidas nos arquivos `enemies.instances`, `enemies.templates`, `items.itm`, `default.data`, `default.map`, `players.txt`, arquivos de jogadores correspondentes e `stores.str`. Após as atualizações, todos os arquivos mencionados foram excluídos. O único arquivo que permanece sendo utilizado pelo software é o `game.data`, que é utilizado para armazenar metadados do servidor e não informações relacionadas às entidades.

Os Códigos 52 e 53 apresentam um exemplo de atualização realizadas nos itens, para o consumo do banco de dados. O código referente ao consumo do `postgres` foi apresentado sem as verificações de segurança realizadas, para diminuir o número de linhas listadas.

Foram realizadas atualizações significativas em duas áreas principais do código. A primeira está relacionada ao acesso às informações, que agora não são mais obtidas por meio de um objeto `ifstream`, mas sim através de uma conexão *nontransaction* que realiza consultas SQL ao banco de dados. Essa mudança proporciona uma abordagem mais eficiente e direta para recuperar os dados necessários.

A segunda área de atualização diz respeito à forma como a conversão das informações para a classe `Item` é realizada. Anteriormente, essa conversão era feita por meio da sobrecarga do operador >>. No entanto, após a atualização, foi adotada uma abordagem mais robusta utilizando a função `ParseRow` (Código 54). Essa função é responsável por analisar e extrair os dados da linha retornada pela consulta SQL, realizando a conversão adequada para preencher corretamente os atributos da classe `Item`. Como pode ser visto no Código 54, linha 15, essa abordagem também foi adotada para conversão dos atributos do item.

Código 52 – Função de carregamento do itens salvos no banco de dados (arquivos) para o jogo.

```
1  // ItemDatabase.cpp , linha 23
2  bool ItemDatabase::Load() {
3    std::ifstream file("items/items.itm");
4    entityid id;
5    std::string temp;
6
7    while (file.good()) {
8      file >> temp >> id;
9      m_map[id].ID() = id;
10     file >> m_map[id] >> std::ws;
11     USERLOG.Log("Loaded Item: " + m_map[id].Name());
12   }
13   return true;
14 }
```

Fonte: Penton (2003)

Código 53 – Função de carregamento do itens salvos no banco de dados (`postgres`) para o jogo.

```
1  // ItemDatabase.cpp , linha 26
2  bool ItemDatabase::Load() {
3    entityid id;
4    pqxx::connection dbConnection;
5
6    /* Create SQL statement */
7    std::string sql = "SELECT *, (attributes).* from Item";
8
9    /* Create a non-transactional object. */
10   pqxx::nontransaction nonTransactionConnection(dbConnection);
11
12   /* Execute SQL query */
13   pqxx::result queryResult(nonTransactionConnection.exec(sql));
14   nonTransactionConnection.commit();
15
16   /* List down all the records */
17   for (auto const row : queryResult) {
18     id = row["id"].as<entityid>();
19     m_map[id].ID() = id;
20     ParseRow(row, m_map[id]);
21     USERLOG.Log("Loaded Item: " + m_map[id].Name());
22   }
```

```
23   return true;
24 }
```

Fonte: Penton (2003) (com adaptações)

Código 54 – Função `ParseRow` da classe *Item*.

```
1  // Item.h , linha 91
2  // ---------------------------------------------------------------------
3  //  Extracts an item in pqxx::const_result_iterator::reference form from a
4  //  pqxx::result
5  // ---------------------------------------------------------------------
6  inline void ParseRow(const pqxx::const_result_iterator::reference &row,
7                           Item &i) {
8      row["name"] >> i.m_name;
9      i.m_type = GetItemType(row["type"].as<std::string>());
10     row["min"] >> i.m_min;
11     row["max"] >> i.m_max;
12     row["speed"] >> i.m_speed;
13     row["price"] >> i.m_price;
14
15     ParseRow(row, i.m_attributes);
16 }
```

Fonte: Autor

No total, 39 arquivos foram afetados por essa atualização. Os detalhes das alterações realizadas podem ser vistos no *pull request* "US57 - Implementar conexão entre o banco de dados e o MUD"[18].

[18] <https://github.com/UnMUD/UnMUD/pull/43/files>

5 Considerações Finais

Este capítulo aborda as considerações finais a respeito do trabalho elaborado, fornecendo uma visão geral sobre o objetivo geral e os objetivos específicos alcançados durante o desenvolvimento. Nele também são apresentados comentários a respeito da experiência adquirida nesse período. Por fim, são discutidas as possibilidades planejadas para trabalhos futuros e é abordado, de maneira geral, outras áreas promissoras que podem ser exploradas e desenvolvidas por pesquisadores subsequentes.

5.1 Objetivos Alcançados

Para a elaboração deste trabalho, foram definidos cinco objetivos específicos que, com base nos resultados apresentados no Capítulo 4, pode-se afirmar que foram alcançados com sucesso. A concretização destes objetivos específicos resulta no êxito do objetivo geral de tornar a base de códigos de um MUD implantável, para que outros estudantes possam explorá-lo e evoluí-lo conforme desejarem. Os objetivos específicos são:

- selecionar uma base de códigos de um MUD;
- analisar a base selecionada;
- atualizá-la para que ela funcione utilizando tecnologias atuais;
- criar um ambiente de desenvolvimento onde seja possível realizar um desenvolvimento contínuo da base;
- realizar manutenções necessárias.

Para seleção da base de códigos do MUD foram encontrados algumas dificuldades devido a idade dos códigos dos MUDs, que pararam de ser atualizados a medida que a tecnologia utilizada para jogos evoluiu. Para garantir que o software era funcional e que seria possível realizar o trabalho com base nele foi necessário analisar dois MUDs diferentes, o Dyrt e o SimpleMUD, uma vez que a compilação e execução do Dyrt, base de código escolhida inicialmente, falharam e então foi necessário encontrar outra base de código que funcionasse, que foi o caso do SimpleMUD, desenvolvido por Ron Penton.

Apesar da tentativa de executar o Dyrt ter sido falha, diversas alterações foram realizadas no código disponibilizado. A compilação e alguns dos erros apresentados ao executar o binário também foram corrigidos, porém a conexão com o servidor do MUD falhou. Após essa falha não foram feitas mais investigações dos possíveis problemas.

Após a seleção do código, foram iniciadas as alterações necessárias para que fosse possível conduzir as análises estáticas e de qualidade utilizando as ferramentas Cppcheck, Clang-tidy, Sonarqube e GCC. Essas alterações consistiram na criação do ambiente de desenvolvimento contínuo, com a utilização das ferramentas Docker e Docker Compose para conteinerização do sistema e o GitHub Actions para criação de fluxos de trabalho para integração contínua.

Com o ambiente de desenvolvimento configurado foi possível gerar os dados sobre o código com as ferramentas de análise e, a partir disso, construir os planejamentos de melhoria e de testes de software, para atualizar o código e torná-lo mais compatível com as práticas de desenvolvimento em C++ moderno. Esses planejamentos foram detalhados nas Seções 4.2.9 e 4.2.8. A utilização de ferramentas distintas permitiu a exploração de uma maior variedade de padrões adotados para C++, o que aumentou a experiência adquirida no desenvolvimento do trabalho.

Por fim, foram realizadas as alterações no código com base nos planejamentos elaborados, como foi apresentado na Seção 4.2.12. Além disso, foram feitas modificações pontuais em padrões de projeto, estruturas condicionais, laços e estruturas de dados obsoletas, apresentadas na Seção 4.2.11. Todo o trabalho foi realizado em um repositório público hospedado no GitHub, permitindo o rastreamento de todas as alterações feitas e o acesso aos arquivos de *log* registrados. O repositório foi desenvolvido seguindo os padrões adotados por projetos de código aberto, e a documentação do projeto foi disponibilizada em um site para auxiliar contribuidores externos.

Muito esforço foi necessário para alcançar os resultados obtidos nesse trabalho. Atualizar um código legado exigiu o aprendizado de ferramentas variadas, além da aplicação de diversos conceitos estudados ao longo do curso de Engenharia de Software, como a própria integração contínua desenvolvida. No entanto, esse esforço valeu a pena, uma vez que elas garantem maior facilidade para realização de alterações futuras e para exploração do código por parte de outros desenvolvedores.

5.2 Trabalhos Futuros

Apesar de todo planejamento elaborado, a limitação de tempo para o desenvolvimento deste trabalho não permitiu que todas melhorias e testes fossem implementados, tornando essas pendências possíveis de serem avaliadas em trabalhos futuros.

Em relação ao planejamento de melhorias, cabe ao próximo desenvolvedor definir qual prioridade ele dará para os erros e *warnings* pendentes, no entanto, manter a priorização atual possibilitaria uma curva de aprendizagem antes de se alcançar problemas complexos. Já em relação aos testes unitários, seria indicado finalizar os testes das bibliotecas utilizadas pelo MUD antes de partir para a testagem do jogo em si.

A parte de jogabilidade do MUD também não foi explorada nesse trabalho, porém foi mapeada no *Backlog* do Produto (Quadro 3). Em relação à usabilidade do jogo é necessário revisar a tela de login, a tela de cadastro e a tela inicial para validar se elas estão intuitivas para os usuários. A adição de um mapa para facilitar a locomoção dos jogadores também é uma possibilidade que deve ser analisada.

Outro ponto importante dos mundos virtuais é a socialização e o SimpleMUD não possui praticamente nenhuma funcionalidade voltada para esse tópico. A adição de listas de amigos, formação de grupos e clãs causariam impactos positivos no jogo. O último item listado no *backlog* referente à jogabilidade e que requer uma análise mais aprofundada é o balanceamento do jogo. Estudar os valores definidos para itens, monstros e habilidades proporcionaria dados sólidos para avaliar a qualidade do progresso do jogo e identificar possíveis áreas de melhoria.

Diversos campos de software também podem ser explorados como continuação desse trabalho. Na área de testes é possível explorar as formas de se testar um software baseado em texto e desenvolver testes mais eficientes para o MUD. Em relação a redes, é necessário estudar as bibliotecas de *socket* e conexão TCP e UDP do C++ e atualizar os códigos utilizados, sendo também possível estudar outras formas de realizar a conexão entre jogadores e servidor. Em relação a software livre e o repositório onde esse projeto foi desenvolvido, ainda está pendente a elaboração de um código de conduta e a tradução de todos os documentos escritos para uma linguagem universal.

Além da tecnologia, existem outros ramos de estudo que podem ser explorados com base na análise do código e no trabalho desenvolvido. O desenvolvimento de MUDs, por serem jogos em um mundo virtual, também envolve disciplinas como a psicologia, podendo ser estudado a interação entre os jogadores do MUD ou a motivação dos estudantes para trabalhar em um jogo em funcionamento ao invés de desenvolver um projeto genérico para as matérias; a matemática, com relação ao balanceamento dos monstros, itens, níveis do jogador e como esses valores podem ser calculados para que o jogo se mantenha interessante; a narratologia, uma vez que adicionar uma história ao jogo seria de grande valor, porém criar uma narração em um contexto não linear pode ser um desafio que exigiria um estudo mais aprofundado; entre outras.

Os MUDs oferecem uma infinidade de possibilidades para que outras pessoas possam explorá-las de acordo com suas preferências e interesses e, através da investigação dessas áreas interdisciplinares, novas ideias e avanços podem ser alcançados, enriquecendo o conhecimento e a aplicação desses jogos.

> *There is only one path into the past, and thence comes story; there are infinite paths into the future.*(BARTLE, 2004) [1]

[1] Existe apenas um caminho para o passado e daí vem a história; existem infinitos caminhos para o futuro (tradução livre).

Referências

The Dyrt Project . *Dyrt*. 1995 – 1999. Citado 4 vezes nas páginas 57, 60, 61 e 62.

BAGUI, S. Mapping generalizations and specializations and categories to relational databases. In: *Handbook of Research on Innovations in Database Technologies and Applications: Current and Future Trends*. [S.l.]: IGI Global, 2009. p. 1–11. Citado 2 vezes nas páginas 47 e 85.

BARCELAR, R. R. Banco de dados. 2012. Citado 2 vezes nas páginas 47 e 85.

BARTLE, R. A. *Designing virtual worlds*. [S.l.]: New Riders, 2004. Citado 5 vezes nas páginas 27, 31, 32, 34 e 105.

BARTLE, R. A. Multi-user dungeons (mud s). *The International Encyclopedia of Digital Communication and Society*, Wiley Online Library, p. 1–8, 2015. Citado 2 vezes nas páginas 31 e 32.

BENNETT, K. H.; RAJLICH, V. T. Software maintenance and evolution: a roadmap. In: *Proceedings of the Conference on the Future of Software Engineering*. [S.l.: s.n.], 2000. p. 73–87. Citado 2 vezes nas páginas 34 e 35.

BOEG, J. Kanban em 10 passos. *Tradução de Leonardo Campos, Marcelo Costa, Lúcio Camilo, Rafael Buzon, Paulo Rebelo, Eric Fer, Ivo La Puma, Leonardo Galvão, Thiago Vespa, Manoel Pimentel e Daniel Wildt. C4Media*, p. 27, 2010. Citado na página 45.

BRASILEO, D. J. P. *Modelo Conceitual de Melhoria de Processos: Uma Aplicação para Projetos de Código Aberto em Organizações de Pequeno e Médio Portes*. Dissertação (Trabalho de Conclusão de Curso 1) — Univesidade de Brasília, 2022. Citado 2 vezes nas páginas 45 e 78.

CHACON, S.; STRAUB, B. *Pro git*. [S.l.]: Springer Nature, 2014. Citado na página 49.

CHAPIN, N. et al. Types of software evolution and software maintenance. *Journal of software maintenance and evolution: Research and Practice*, Wiley Online Library, v. 13, n. 1, p. 3–30, 2001. Citado 7 vezes nas páginas 31, 34, 35, 36, 38, 39 e 43.

CPLUSCPLUS.COM. *C++ Tutorial: Type Casting*. 2023. Disponível em: <https://cplusplus.com/doc/oldtutorial/typecasting/>. Acesso em: Junho, 2023. Citado na página 92.

Cpp-Best-Practices. *C++ Best Practices*. 2021. GitHub Repository. Disponível em: <https://github.com/cpp-best-practices/cppbestpractices>. Citado na página 46.

CPPCHECK. *Cppcheck*. 2023. Disponível em: <https://cppcheck.sourceforge.io/>. Acesso em: Abril, 2023. Citado na página 48.

CPPREFERENCE. *Range-based for loop (since C++11)*. 2023. Disponível em: <https://en.cppreference.com/w/cpp/language/range-for>. Acesso em: Janeiro, 2023. Citado na página 89.

DOCKER. *Docker Compose overview*. 2023. Disponível em: <https://docs.docker.com/compose/>. Acesso em: Abril, 2023. Citado na página 48.

DOCKER. *Docker overview*. 2023. Disponível em: <https://docs.docker.com/get-started/overview/>. Acesso em: Abril, 2023. Citado na página 48.

DOCSIFY. *What it is*. 2023. Disponível em: <https://docsify.js.org/#/>. Acesso em: Junho, 2023. Citado na página 49.

EVERETT, G. D.; JR, R. M. Software testing. *Testing Across the Entire*, 2007. Citado na página 46.

GCOVR. *gcovr*. 2023. Disponível em: <https://gcovr.com/en/stable/>. Acesso em: Abril, 2023. Citado na página 49.

GITHUB. *Entendendo o GitHub Actions*. 2023. Disponível em: <https://docs.github.com/pt/actions/learn-github-actions/understanding-github-actions>. Acesso em: Abril, 2023. Citado na página 50.

GITHUB. *GitHub Docs*. 2023. Disponível em: <https://docs.github.com/pt/>. Acesso em: Janeiro, 2023. Citado na página 50.

GNU. *GCC, the GNU Compiler Collection*. 2023. Disponível em: <https://gcc.gnu.org/>. Acesso em: Junho, 2023. Citado na página 49.

GNU. *GDBM*. 2023. Disponível em: <https://www.gnu.org.ua/software/gdbm/>. Acesso em: Janeiro, 2023. Citado na página 54.

GNU. *GNU Make*. 2023. Disponível em: <https://www.gnu.org/software/make/>. Acesso em: Abril, 2023. Citado na página 50.

GOOGLE. *GoogleTest Primer*. 2023. Disponível em: <https://google.github.io/googletest/primer.html>. Acesso em: Abril, 2023. Citado na página 50.

IEEE. *IEEE Standard Glossary of Software Engineering Terminology*. 1990. Citado na página 34.

MALAIYA, Y. et al. The relationship between test coverage and reliability. In: *Proceedings of 1994 IEEE International Symposium on Software Reliability Engineering*. [S.l.: s.n.], 1994. p. 186–195. Citado na página 99.

NESTERUK, D. *Design Patterns in Modern C++: Reusable Approaches for Object-Oriented Software Design*. [S.l.]: Apress, 2018. Citado na página 72.

PAGOTTO, T. et al. Scrum solo: Software process for individual development. In: IEEE. *2016 11th Iberian Conference on Information Systems and Technologies (CISTI)*. [S.l.], 2016. p. 1–6. Citado 2 vezes nas páginas 45 e 75.

PENTON, R. *MUD game programming*. [S.l.]: Premier Press, 2003. Citado 25 vezes nas páginas 27, 28, 32, 53, 68, 69, 70, 72, 73, 74, 75, 82, 87, 88, 89, 90, 92, 93, 94, 95, 96, 97, 98, 101 e 102.

POSTGRESQL. *What is PostgreSQL?* 2023. Disponível em: <https://www.postgresql.org/>. Acesso em: Maio, 2023. Citado na página 51.

SONARQUBE. *SonarQube Documentation*. 2023. Disponível em: <https://docs.sonarqube.org/latest/>. Acesso em: Abril, 2023. Citado na página 51.

STROUSTRUP, B.; SUTTER, H. et al. *C++ core guidelines*. 2022. Disponível em: <https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines>. Acesso em: Janeiro, 2023. Citado 4 vezes nas páginas 43, 44, 72 e 92.

SWANSON, E. B. The dimensions of maintenance. In: *Proceedings of the 2nd international conference on Software engineering*. [S.l.: s.n.], 1976. p. 492–497. Citado na página 34.

TEAM, T. C. *Clang 17.0.0 - ClangFormat*. 2023. Disponível em: <https://clang.llvm.org/docs/ClangFormat.html>. Acesso em: Abril, 2023. Citado na página 47.

TEAM, T. C. *Extra Clang Tools 17.0.0 - Clang-Tidy*. 2023. Disponível em: <https://clang.llvm.org/extra/clang-tidy/>. Acesso em: Abril, 2023. Citado na página 48.

TURNER, J. *Writing C++ the Right Way*. 2022. Disponível em: <https://github.com/lefticus/cpp_weekly/issues/175>. Acesso em: Janeiro, 2023. Citado 2 vezes nas páginas 44 e 47.

VAGRANT. *Vagrant Documentation*. 2023. Disponível em: <https://developer.hashicorp.com/vagrant/docs>. Acesso em: Abril, 2023. Citado na página 51.

Apêndices

APÊNDICE A – Arquivo de configuração do Vagrant

Código 55 – Vagrantfile

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

# All Vagrant configuration is done below. The "2" in Vagrant.configure
# configures the configuration version (we support older styles for
# backwards compatibility). Please don't change it unless you know what
# you're doing.
Vagrant.configure("2") do |config|
    # The most common configuration options are documented and commented below.
    # For a complete reference, please see the online documentation at
    # https://docs.vagrantup.com.

    # Every Vagrant development environment requires a box. You can search for
    # boxes at https://vagrantcloud.com/search.
    config.vm.box = "ubuntu/focal64"

    # Enable provisioning with a shell script. Additional provisioners such as
    # Ansible, Chef, Docker, Puppet and Salt are also available. Please see the
    # documentation for more information about their specific syntax and use.
    # config.vm.provision "shell", inline: <<-SHELL
    #   apt-get update
    #   apt-get install -y apache2
    # SHELL
    config.vm.provision "shell", inline: <<-SHELL
        apt update
        apt install build-essentials gcc-10-doc libstdc++-10-doc
        apt install libmsgsl-dev
    SHELL
  end

```

Fonte: Autor

APÊNDICE B – Backlog do Produto

Quadro 3 – Backlog do Produto

Épico	**Feature**	**ID**	**História de usuário**	**Estimativa de tempo (dias)**	**Tempo real (dias)**	**Data de Inserção**	**Data de Seleção**	**Prioridade**
Desenvolvimento	Ambiente de desenvolvimento	US01	Containerizar o ambiente de desenvolvimento	1	1	Mar 31 2023	Apr 11 2023	Must
		US02	Adicionar verificações estáticas de código	3	3	Mar 31 2023	Apr 11 2023	Must
		US03	Adicionar ferramentas de inspeção da qualidade do código	2	2	Mar 31 2023	Apr 11 2023	Should
		US62	Adicionar runtime analysis	3	-	May 19 2023	-	Must
	Organização do Repositório	US04	Pesquisar boas práticas adotadas por comunidades open source	1	1	Mar 31 2023	Apr 04 2023	Should
		US05	Escrever uma descrição / README significativo	0,3	0,3	Mar 31 2023	Apr 04 2023	Should
		US06	Definir licença do repositório	0,2	0,2	Mar 31 2023	Apr 04 2023	Must
		US07	Criar labels para issues	0,5	-	Mar 31 2023	-	Should

Fonte: Autor.

Quadro 3 – Backlog do Produto (Continuação)

Épico	**Feature**	**ID**	**História de usuário**	**Estimativa de tempo (dias)**	**Tempo real (dias)**	**Data de Inserção**	**Data de Seleção**	**Prioridade**
Desenvolvimento	Documentação do Repositório	US08	Criar templates para criação de Issues e Pull Requests	0,3	0,3	Mar 31 2023	Apr 04 2023	Should
		US09	Criar um guia de contribuição	0,7	0,7	Mar 31 2023	Apr 04 2023	Must
		US59	Criar um código de conduta	0,5	-	Apr 06 2023	-	Should
		US60	Traduzir documentação para o inglês	X	-	Apr 06 2023	-	Could
	Documentação do Projeto	US10	Criar documento de arquitetura	1	-	Mar 31 2023	-	Should
		US11	Criar documento de visão	1	-	Mar 31 2023	-	Could

Fonte: Autor.

Quadro 3 – Backlog do Produto (Continuação)

Épico	**Feature**	**ID**	**História de usuário**	**Estimativa de tempo (dias)**	**Tempo real (dias)**	**Data de Inserção**	**Data de Seleção**	**Prioridade**
Desenvolvimento	Documentação do Projeto	US12	Criar template de atas de reunião e de sprint backlog	0,5	0,5	Mar 31 2023	Apr 04 2023	Could
		US13	Criar EAP	0,5	-	Mar 31 2023	-	Should
		US14	Criar Cronograma	1	1	Mar 31 2023	Apr 04 2023	Must
	Documentação do Código	US15	Pesquisar padrões de documentação adotados para código em C e ferramentas de documentação	1	-	Mar 31 2023	-	Should
		US16	Documentar bibliotecas criadas	4	-	Mar 31 2023	-	Should
		US17	Documentar código base do MUD	5	-	Mar 31 2023	-	Should
	Documentação do Jogo	US18	Criar um guia de "Como Jogar"	1	-	Mar 31 2023	-	Could
		US19	Criar um "Bestiário"	1	-	Mar 31 2023	-	Could
		US20	Criar uma wiki para o jogo	1	-	Mar 31 2023	-	Could

Quadro 3 – Backlog do Produto (Continuação)

Épico	Feature	ID	História de usuário	Estimativa de tempo (dias)	Tempo real (dias)	Data de Inserção	Data de Seleção	Prioridade
Desenvolvimento	Testes	US21	Pesquisar frameworks de testes em C++	2	2	Mar 31 2023	Apr 17 2023	Should
		US22	Criar planejamento de testes	2	3	Mar 31 2023	Apr 17 2023	Must
		US23	Aplicar planejamento de testes	X	14	Mar 31 2023	Apr 28 2023	Must
		US24	Automatizar testes	2	2	Mar 31 2023	Apr 17 2023	Should
		US61	Corrigir erros encontrados com a implementação dos testes	5	2	May 12 2023	Jun 16 2023	Should
	Implantação do sistema	US25	Realizar provisionamento de servidores	1	2	Mar 31 2023	May 19 2023	Should
		US26	Realizar provisionamento de usuários	1	2	Mar 31 2023	May 19 2023	Should
		US27	Realizar provisionamento de redes	1	1	Mar 31 2023	May 19 2023	Should
		US28	Implantar sistema	2	2	Mar 31 2023	May 19 2023	Must
		US29	Automatizar implantação do sistema	2	-	Mar 31 2023	-	Should
Jogabilidade	Gameplay	US63	Corrigir bug de respawn de monstros	X	-	May 19 2023	-	Must
	Socialização	US30	Adicionar lista de amigos	3	-	Mar 31 2023	-	Would
		US31	Adicionar formação de grupos	4	-	Mar 31 2023	-	Would
		US32	Adicionar criação de clãs	3	-	Mar 31 2023	-	Would

Fonte: Autor.

Quadro 3 – Backlog do Produto (Continuação)

Épico	**Feature**	**ID**	**História de usuário**	**Estimativa de tempo (dias)**	**Tempo real (dias)**	**Data de Inserção**	**Data de Seleção**	**Prioridade**
Jogabilidade	Balanceamento	US33	Revisar items implementados	1	-	Mar 31 2023	-	Could
		US34	Revisar habilidades implementadas	1	-	Mar 31 2023	-	Could
		US35	Revisar monstros implementados	1	-	Mar 31 2023	-	Could
		US36	Revisar desenho do mapa	1	-	Mar 31 2023	-	Could
		US37	Revisar progressão do jogo	2	-	Mar 31 2023	-	Would
	Interface	US38	Revisar tela de login	1	-	Mar 31 2023	-	Could
		US39	Revisar tela de cadastro	1	-	Mar 31 2023	-	Could
		US40	Revisar tela inicial	1	-	Mar 31 2023	-	Could
		US41	Adicionar mapa	3	-	Mar 31 2023	-	Could
TCC	Texto	US42	Aplicar revisões sugeridas pela banca de avaliação do TCC1	5	5	Apr 01 2023	Apr 04 2023	Must
		US43	Atualizar fundamentação teórica	2	2	Apr 01 2023	Jun 23 2023	Must
		US44	Atualizar metodologia	2	4	Apr 01 2023	Apr 11 2023	Must
		US45	Documentar tarefas realizadas para o desenvolvimento do projeto	X	7	Apr 01 2023	Jun 02 2023	Must
	Código	US46	Classificar erros encontrados	1	2	Apr 01 2023	Apr 28 2023	Should
		US47	Criar um plano de melhorias	3	2	Apr 01 2023	Apr 28 2023	Should
		US48	Aplicar plano de melhorias	X	14	Apr 01 2023	May 05 2023	Must
		US64	Atualizar funções de geração de valores aleatórios	X	-	May 19 2023	-	Could

Fonte: Autor.

Quadro 3 – Backlog do Produto (Continuação)

Épico	**Feature**	**ID**	**História de usuário**	**Estimativa de tempo (dias)**	**Tempo real (dias)**	**Data de Inserção**	**Data de Seleção**	**Prioridade**
Pibic	Relatório	US49	Escrever resumo	2	-	Apr 01 2023	-	Must
		US50	Escrever introdução	3	-	Apr 01 2023	-	Must
		US51	Escrever metodologia	4	4	Apr 01 2023	Apr 11 2023	Must
		US52	Escrever resultados obtidos	6	-	Apr 01 2023	Jun 09 2023	Must
		US53	Escrever conclusão	4	-	Apr 01 2023	-	Must
		US54	Revisar texto	1	-	Apr 01 2023	-	Should
	Banco de dados	US55	Desenhar banco de dados	2	3	Apr 01 2023	May 12 2023	Must
		US56	Construir banco de dados	3	2	Apr 01 2023	May 12 2023	Must
		US57	Implementar conexão entre o banco de dados e o MUD	3	4	Apr 01 2023	May 19 2023	Must
		US58	Documentar aplicação do banco de dados	2	2	Apr 01 2023	May 12 2023	Should
		US65	Converter maps para acesso direto ao banco	X	-	May 26 2023	-	Should

Fonte: Autor.

APÊNDICE C – Lista de Erros e *Warnings*

Quadro 4 – Lista de Erros e *Warnings*

Identificação	**Fonte**	**Classificação**	**Conhecimento** (1 - 5)	**Tempo** (min)	**Complexidade** (1 - 5)	**Nível**	**Ocorrências**
Undefined error	-	error	1	$90 < x < 180$	4	B	1
A has not been declared	gcc	error	4	$30 < x < 90$	3	F	1
A was not declared in this scope	gcc	error	4	$30 < x < 90$	2	G	19
dependent-name A is parsed as a non-type, but instantiation yields a type	gcc	error	3	$30 < x < 90$	3	E	1
invalid storage class for function	gcc	error	3	$30 < x < 90$	3	E	2
multiple definition of	gcc	error	5	$x < 30$	3	G	35
need 'typename' before A because B is a dependent scope	gcc	error	3	$30 < x < 90$	4	D	5
no matching function for call to A	gcc	error	2	$30 < x < 90$	3	D	2
No such file or directory	bash	error	5	$x < 30$	2	H	3
Segmentation fault (core dumped)	gcc	error	2	$90 < x < 180$	4	B	3

Fonte: Autor.

Quadro 4 – Lista de Erros e *Warnings* (Continuação)

Identificação	**Fonte**	**Classificação**	**Conhecimento** (1 - 5)	**Tempo** (min)	**Complexidade** (1 - 5)	**Nível**	**Ocorrências**
specializing member A requires 'template<>' syntax	gcc	error	2	30 < x < 90	3	D	6
static declaration of A follows non-static declaration	gcc	error	4	x < 30	3	G	1
'sys_errlist' undeclared	gcc	error	5	x < 30	2	H	3
there are no arguments to A that depend on a template parameter, so a declaration of A must be available	gcc	error	5	x < 30	2	H	9
errno: TLS definition in A section .tbss mismatches non-TLS reference in B	gcc	error	3	x < 30	3	F	1

Fonte: Autor.

Quadro 4 – Lista de Erros e *Warnings* (Continuação)

Identificação	**Fonte**	**Classificação**	**Conhecimento** (1 - 5)	**Tempo** (min)	**Complexidade** (1 - 5)	**Nível**	**Ocorrências**
variable or field A declared void	gcc	error	3	30 < x < 90	3	E	1
catching polymorphic type	gcc	warning	3	x < 30	3	F	3
conversion from A to B may change value	gcc	warning	4	x < 30	2	H	78
comparison of integer expressions of different signedness	gcc	warning	4	x < 30	1	H	4
conversion to B from A may change the sign of the result	gcc	warning	4	x < 30	2	H	36
declaration of A shadows a global declaration	gcc	warning	4	x < 30	3	G	1
has pointer data members A but does not override B or C	gcc	warning	3	30 < x < 90	2	F	16
should be initialized in the member initialization list	gcc	warning	5	x < 30	1	H	298
unused parameter	gcc	warning	5	x < 30	3	G	1

Fonte: Autor.

Quadro 4 – Lista de Erros e *Warnings* (Continuação)

Identificação	**Fonte**	**Classificação**	**Conhecimento** (1 - 5)	**Tempo** (min)	**Complexidade** (1 - 5)	**Nível**	**Ocorrências**
use of old-style cast	gcc	warning	4	x < 30	2	H	227
constParameter	cppcheck	warning	5	x < 30	1	H	3
cstyleCast	cppcheck	warning	4	x < 30	3	G	1
no Explicit Constructor	cppcheck	warning	3	x < 30	2	G	46
passedByValue	cppcheck	warning	4	x < 30	2	H	3
selfAssignment	cppcheck	warning	3	x < 30	2	G	1
uninitMemberVar	cppcheck	warning	5	x < 30	1	H	23
unreadVariable	cppcheck	warning	5	x < 30	3	G	1
unusedFunction	cppcheck	warning	5	30 < x < 90	3	F	26
use Initialization List	cppcheck	warning	5	x < 30	1	H	26
android-cloexec-accept	clang-tidy	warning	4	x < 30	1	H	1
array-to-pointer-decay	clang-tidy	warning	3	x < 30	1	G	22
avoid-c-arrays	clang-tidy	warning	5	x < 30	1	H	6
bounds-constant-array-index	clang-tidy	warning	3	x < 30	2	G	15
bounds-pointer-arithmetic	clang-tidy	warning	5	x < 30	3	G	5
braces-around-statements	clang-tidy	warning	5	x < 30	1	H	180
bugprone-macro-parentheses	clang-tidy	warning	4	x < 30	1	H	6
build-using-namespace	clang-tidy	warning	3	x < 30	1	G	17
casting	clang-tidy	warning	4	x < 30	2	H	19
cert-err58-cpp	clang-tidy	warning	3	x < 30	3	F	15
container-size-empty	clang-tidy	warning	5	x < 30	1	H	1
convert-member-functions-to-static	clang-tidy	warning	4	x < 30	3	G	5
default-arguments-calls	clang-tidy	warning	5	x < 30	3	G	131

Fonte: Autor.

Quadro 4 – Lista de Erros e *Warnings* (Continuação)

Identificação	**Fonte**	**Classificação**	**Conhecimento** (1 - 5)	**Tempo** (min)	**Complexidade** (1 - 5)	**Nível**	**Ocorrências**
deprecated-headers	clang-tidy	warning	5	x < 30	1	H	4
else-after-return	clang-tidy	warning	5	x < 30	3	G	4
header-guard	clang-tidy	warning	3	x < 30	2	G	42
implicit-bool-conversion	clang-tidy	warning	5	x < 30	2	H	1
include-order	clang-tidy	warning	5	x < 30	1	H	54
init-variables	clang-tidy	warning	5	x < 30	1	H	23
isolate-declaration	clang-tidy	warning	5	x < 30	1	H	3
loop-convert	clang-tidy	warning	5	x < 30	2	H	5
macro-usage	clang-tidy	warning	3	x < 30	3	F	1
magic-numbers	clang-tidy	warning	5	x < 30	2	H	98
make-member-function-const	clang-tidy	warning	4	x < 30	2	H	3
member-init	clang-tidy	warning	5	x < 30	1	H	12
namespace-comments	clang-tidy	warning	4	x < 30	1	H	2
no-assembler	clang-tidy	warning	3	x < 30	3	F	2
non-const-parameter	clang-tidy	warning	4	x < 30	2	H	1
non-private-member-variables-in-classes	clang-tidy	warning	5	x < 30	3	G	1
overloaded-operator	clang-tidy	warning	5	30 < x < 90	3	F	16
owning-memory	clang-tidy	warning	3	30 < x < 90	3	E	3

Fonte: Autor.

Quadro 4 – Lista de Erros e *Warnings* (Continuação)

Identificação	**Fonte**	**Classificação**	**Conhecimento** (1 - 5)	**Tempo** (min)	**Complexidade** (1 - 5)	**Nível**	**Ocorrências**
performance-unnecessary-value-param	clang-tidy	warning	4	x < 30	3	G	7
return-braced-init-list	clang-tidy	warning	4	x < 30	2	H	1
runtime-references	clang-tidy	warning	5	x < 30	3	G	1
signed-bitwise	clang-tidy	warning	4	x < 30	2	H	6
simplify-boolean-expr	clang-tidy	warning	5	x < 30	3	G	7
statically-constructed-objects	clang-tidy	warning	4	30 < x < 90	3	F	11
type-cstyle-cast	clang-tidy	warning	4	x < 30	3	G	11
unused-using-decls	clang-tidy	warning	4	x < 30	2	H	10
use-auto	clang-tidy	warning	5	x < 30	1	H	20
use-emplace	clang-tidy	warning	5	x < 30	2	H	4
use-nullptr	clang-tidy	warning	5	x < 30	2	H	24
use-trailing-return-type	clang-tidy	warning	4	x < 30	2	H	81
vararg	clang-tidy	warning	3	x < 30	2	G	4

Fonte: Autor.

APÊNDICE D – Relatório de cobertura de testes pré-implementação de testes unitários

GCC Code Coverage Report

			Exec	Total	Coverage
Directory: ./					
Date:	2023-04-24 19:27:20	Lines:	97	2302	4.2 %
Legend:	low: < 75.0 % medium: >= 75.0 % high: >= 90.0 %	Branches:	30	3816	0.8 %

File	Lines		Branches	
Libraries/BasicLib/BasicLibFiles.cpp	**0.0 %**	**0 / 8**	0.0 %	0 / 12
Libraries/BasicLib/BasicLibFunctions.h	**0.0 %**	**0 / 26**	0.0 %	0 / 37
Libraries/BasicLib/BasicLibLogger.h	**95.1 %**	**39 / 41**	46.7 %	28 / 60
Libraries/BasicLib/BasicLibRandom.h	**82.4 %**	**42 / 51**	- %	0 / 0
Libraries/BasicLib/BasicLibString.cpp	**0.0 %**	**0 / 49**	0.0 %	0 / 28
Libraries/BasicLib/BasicLibString.h	**0.0 %**	**0 / 12**	0.0 %	0 / 22
Libraries/BasicLib/BasicLibTime.cpp	**25.5 %**	**14 / 55**	5.9 %	2 / 34
Libraries/BasicLib/BasicLibTime.h	**100.0 %**	**2 / 2**	- %	0 / 0
Libraries/SocketLib/Connection.h	**0.0 %**	**0 / 74**	0.0 %	0 / 30
Libraries/SocketLib/ConnectionHandler.h	**0.0 %**	**0 / 2**	- %	0 / 0
Libraries/SocketLib/ConnectionManager.h	**0.0 %**	**0 / 72**	0.0 %	0 / 82
Libraries/SocketLib/ListeningManager.h	**0.0 %**	**0 / 26**	0.0 %	0 / 38
Libraries/SocketLib/SocketLibErrors.cpp	**0.0 %**	**0 / 158**	0.0 %	0 / 147
Libraries/SocketLib/SocketLibSocket.cpp	**0.0 %**	**0 / 98**	0.0 %	0 / 98
Libraries/SocketLib/SocketLibSocket.h	**0.0 %**	**0 / 4**	- %	0 / 0
Libraries/SocketLib/SocketLibSystem.cpp	**0.0 %**	**0 / 26**	0.0 %	0 / 36
Libraries/SocketLib/SocketSet.cpp	**0.0 %**	**0 / 9**	0.0 %	0 / 4
Libraries/SocketLib/SocketSet.h	**0.0 %**	**0 / 7**	0.0 %	0 / 4
Libraries/SocketLib/Telnet.cpp	**0.0 %**	**0 / 14**	0.0 %	0 / 26
Libraries/SocketLib/Telnet.h	**0.0 %**	**0 / 3**	- %	0 / 0
Libraries/ThreadLib/ThreadLibFunctions.cpp	**0.0 %**	**0 / 5**	0.0 %	0 / 2
Libraries/ThreadLib/ThreadLibFunctions.h	**0.0 %**	**0 / 2**	- %	0 / 0
SimpleMUD/SimpleMUD.cpp	**0.0 %**	**0 / 21**	0.0 %	0 / 38
SimpleMUD/SimpleMUD/Attributes.h	**0.0 %**	**0 / 35**	0.0 %	0 / 28
SimpleMUD/SimpleMUD/DatabasePointer.cpp	**0.0 %**	**0 / 5**	0.0 %	0 / 10
SimpleMUD/SimpleMUD/DatabasePointer.h	**0.0 %**	**0 / 5**	- %	0 / 0
SimpleMUD/SimpleMUD/Enemy.cpp	**0.0 %**	**0 / 59**	0.0 %	0 / 70
SimpleMUD/SimpleMUD/Enemy.h	**0.0 %**	**0 / 3**	- %	0 / 0
SimpleMUD/SimpleMUD/EnemyDatabase.cpp	**0.0 %**	**0 / 40**	0.0 %	0 / 86
SimpleMUD/SimpleMUD/EnemyDatabase.h	**0.0 %**	**0 / 2**	- %	0 / 0
SimpleMUD/SimpleMUD/Entity.h	**0.0 %**	**0 / 32**	0.0 %	0 / 26
SimpleMUD/SimpleMUD/EntityDatabase.h	**0.0 %**	**0 / 46**	0.0 %	0 / 34
SimpleMUD/SimpleMUD/Game.cpp	**0.0 %**	**0 / 629**	0.0 %	0 / 1862
SimpleMUD/SimpleMUD/Game.h	**0.0 %**	**0 / 4**	- %	0 / 0
SimpleMUD/SimpleMUD/GameLoop.cpp	**0.0 %**	**0 / 69**	0.0 %	0 / 130
SimpleMUD/SimpleMUD/GameLoop.h	**0.0 %**	**0 / 2**	- %	0 / 0
SimpleMUD/SimpleMUD/Item.h	**0.0 %**	**0 / 22**	0.0 %	0 / 32

SimpleMUD/SimpleMUD/ItemDatabase.cpp	**0.0 %**	**0 / 12**	0.0 %	0 / 28
SimpleMUD/SimpleMUD/ItemDatabase.h	**0.0 %**	**0 / 1**	- %	0 / 0
SimpleMUD/SimpleMUD/Logon.cpp	**0.0 %**	**0 / 98**	0.0 %	0 / 222
SimpleMUD/SimpleMUD/Logon.h	**0.0 %**	**0 / 17**	0.0 %	0 / 20
SimpleMUD/SimpleMUD/Player.cpp	**0.0 %**	**0 / 180**	0.0 %	0 / 166
SimpleMUD/SimpleMUD/Player.h	**0.0 %**	**0 / 43**	0.0 %	0 / 14
SimpleMUD/SimpleMUD/PlayerDatabase.cpp	**0.0 %**	**0 / 51**	0.0 %	0 / 100
SimpleMUD/SimpleMUD/PlayerDatabase.h	**0.0 %**	**0 / 11**	0.0 %	0 / 20
SimpleMUD/SimpleMUD/Room.cpp	**0.0 %**	**0 / 61**	0.0 %	0 / 84
SimpleMUD/SimpleMUD/Room.h	**0.0 %**	**0 / 10**	- %	0 / 0
SimpleMUD/SimpleMUD/RoomDatabase.cpp	**0.0 %**	**0 / 26**	0.0 %	0 / 50
SimpleMUD/SimpleMUD/RoomDatabase.h	**0.0 %**	**0 / 1**	- %	0 / 0
SimpleMUD/SimpleMUD/Store.h	**0.0 %**	**0 / 18**	0.0 %	0 / 24
SimpleMUD/SimpleMUD/StoreDatabase.cpp	**0.0 %**	**0 / 12**	0.0 %	0 / 28
SimpleMUD/SimpleMUD/StoreDatabase.h	**0.0 %**	**0 / 1**	- %	0 / 0
SimpleMUD/SimpleMUD/Train.cpp	**0.0 %**	**0 / 36**	0.0 %	0 / 84
SimpleMUD/SimpleMUD/Train.h	**0.0 %**	**0 / 6**	- %	0 / 0

Generated by: GCOVR (Version 4.2)

APÊNDICE E – Relatório de cobertura de testes pós-implementação de testes unitários

GCC Code Coverage Report

Directory: ./
Date: 2023-07-01 12:28:45
Legend: low: >= 0% medium: >= 75.0% high: >= 90.0%

	Exec	Total	Coverage
Lines:	270	2689	10.0%
Branches:	149	4812	3.1%

File	Lines		Branches	
Libraries/BasicLib/BasicLibFiles.cpp	100.0%	10 / 10	64.3%	9 / 14
Libraries/BasicLib/BasicLibFunctions.h	100.0%	37 / 37	71.7%	38 / 53
Libraries/BasicLib/BasicLibLogger.h	100.0%	38 / 38	55.0%	33 / 60
Libraries/BasicLib/BasicLibRandom.h	100.0%	33 / 33	-%	0 / 0
Libraries/BasicLib/BasicLibString.cpp	100.0%	55 / 55	77.5%	31 / 40
Libraries/BasicLib/BasicLibString.h	100.0%	20 / 20	57.7%	15 / 26
Libraries/BasicLib/BasicLibTime.cpp	100.0%	45 / 45	50.0%	17 / 34
Libraries/BasicLib/BasicLibTime.h	100.0%	5 / 5	-%	0 / 0
Libraries/SocketLib/Connection.h	0.0%	0 / 68	0.0%	0 / 32
Libraries/SocketLib/ConnectionHandler.h	0.0%	0 / 2	-%	0 / 0
Libraries/SocketLib/ConnectionManager.h	0.0%	0 / 70	0.0%	0 / 82
Libraries/SocketLib/ListeningManager.h	0.0%	0 / 26	0.0%	0 / 40
Libraries/SocketLib/SocketLibErrors.cpp	0.0%	0 / 156	0.0%	0 / 147
Libraries/SocketLib/SocketLibSocket.cpp	0.0%	0 / 96	0.0%	0 / 98
Libraries/SocketLib/SocketLibSocket.h	0.0%	0 / 3	-%	0 / 0
Libraries/SocketLib/SocketLibSystem.cpp	0.0%	0 / 25	0.0%	0 / 36
Libraries/SocketLib/SocketSet.cpp	0.0%	0 / 11	0.0%	0 / 8
Libraries/SocketLib/SocketSet.h	0.0%	0 / 8	0.0%	0 / 4
Libraries/SocketLib/Telnet.cpp	0.0%	0 / 14	0.0%	0 / 26
Libraries/SocketLib/Telnet.h	0.0%	0 / 2	-%	0 / 0
Libraries/ThreadLib/ThreadException.h	100.0%	2 / 2	-%	0 / 0
Libraries/ThreadLib/ThreadLibFunctions.cpp	100.0%	5 / 5	50.0%	1 / 2
Libraries/ThreadLib/ThreadLibFunctions.h	90.9%	20 / 22	50.0%	5 / 10
SimpleMUD/SimpleMUD.cpp	0.0%	0 / 25	0.0%	0 / 52
SimpleMUD/SimpleMUD/Attributes.h	0.0%	0 / 45	0.0%	0 / 46
SimpleMUD/SimpleMUD/DatabasePointer.cpp	0.0%	0 / 5	0.0%	0 / 10
SimpleMUD/SimpleMUD/DatabasePointer.h	0.0%	0 / 5	-%	0 / 0
SimpleMUD/SimpleMUD/Enemy.cpp	0.0%	0 / 77	0.0%	0 / 118
SimpleMUD/SimpleMUD/Enemy.h	0.0%	0 / 3	-%	0 / 0
SimpleMUD/SimpleMUD/EnemyDatabase.cpp	0.0%	0 / 128	0.0%	0 / 302
SimpleMUD/SimpleMUD/EnemyDatabase.h	0.0%	0 / 2	-%	0 / 0
SimpleMUD/SimpleMUD/Entity.h	0.0%	0 / 25	0.0%	0 / 26
SimpleMUD/SimpleMUD/EntityDatabase.h	0.0%	0 / 44	0.0%	0 / 34
SimpleMUD/SimpleMUD/Game.cpp	0.0%	0 / 627	0.0%	0 / 1868
SimpleMUD/SimpleMUD/Game.h	0.0%	0 / 4	-%	0 / 0
SimpleMUD/SimpleMUD/GameLoop.cpp	0.0%	0 / 84	0.0%	0 / 130
SimpleMUD/SimpleMUD/GameLoop.h	0.0%	0 / 3	-%	0 / 0
SimpleMUD/SimpleMUD/Item.h	0.0%	0 / 17	0.0%	0 / 16
SimpleMUD/SimpleMUD/ItemDatabase.cpp	0.0%	0 / 25	0.0%	0 / 64
SimpleMUD/SimpleMUD/ItemDatabase.h	0.0%	0 / 1	-%	0 / 0
SimpleMUD/SimpleMUD/Logon.cpp	0.0%	0 / 105	0.0%	0 / 222
SimpleMUD/SimpleMUD/Logon.h	0.0%	0 / 14	0.0%	0 / 22
SimpleMUD/SimpleMUD/Player.cpp	0.0%	0 / 210	0.0%	0 / 278
SimpleMUD/SimpleMUD/Player.h	0.0%	0 / 39	0.0%	0 / 14
SimpleMUD/SimpleMUD/PlayerDatabase.cpp	0.0%	0 / 146	0.0%	0 / 298
SimpleMUD/SimpleMUD/PlayerDatabase.h	0.0%	0 / 9	0.0%	0 / 18
SimpleMUD/SimpleMUD/Room.cpp	0.0%	0 / 81	0.0%	0 / 152
SimpleMUD/SimpleMUD/Room.h	0.0%	0 / 10	-%	0 / 0
SimpleMUD/SimpleMUD/RoomDatabase.cpp	0.0%	0 / 115	0.0%	0 / 254
SimpleMUD/SimpleMUD/RoomDatabase.h	0.0%	0 / 1	-%	0 / 0
SimpleMUD/SimpleMUD/Store.h	0.0%	0 / 19	0.0%	0 / 28
SimpleMUD/SimpleMUD/StoreDatabase.cpp	0.0%	0 / 25	0.0%	0 / 64
SimpleMUD/SimpleMUD/StoreDatabase.h	0.0%	0 / 1	-%	0 / 0
SimpleMUD/SimpleMUD/Train.cpp	0.0%	0 / 36	0.0%	0 / 84
SimpleMUD/SimpleMUD/Train.h	0.0%	0 / 5	-%	0 / 0

Generated by: GCOVR (Version 5.0)

Anexos

ANEXO A – Licença AGPL-3.0

GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007

Preamble

The GNU Affero General Public License is a free, copyleft license for software and other kinds of works, specifically designed to ensure cooperation with the community in the case of network server software.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, our General Public Licenses are intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

Developers that use our General Public Licenses protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License which gives you legal permission to copy, distribute and/or modify the software.

A secondary benefit of defending all users' freedom is that improvements made in alternate versions of the program, if they receive widespread use, become available for other developers to incorporate. Many developers of free software are heartened and encouraged by the resulting cooperation. However, in the case of software used on network servers, this result may fail to come about. The GNU General Public License permits making a modified version and letting the public access it on a server without ever releasing its source code to the public.

The GNU Affero General Public License is designed specifically to ensure that, in such cases, the modified source code becomes available to the community. It requires the operator of a network server to provide the source code of the modified version running there to the users of that server. Therefore, public use of a modified version, on a publicly accessible server, gives the public access to the source code of the modified version.

An older license, called the Affero General Public License and published by Affero, was designed to accomplish similar goals. This is a different license, not a version of the Affero GPL, but Affero has released a new version of the Affero GPL which permits relicensing under this license.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

"This License"refers to version 3 of the GNU Affero General Public License.

"Copyright"also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program"refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees"and "recipients"may be individuals or organizations.

To "modify"a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version"of the earlier work or a work "based on"the earlier work.

A "covered work"means either the unmodified Program or a work based on the Program.

To "propagate"a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey"a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices"to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code"for a work means the preferred form of the work for making modifications to it. "Object code"means any non-source form of a work.

A "Standard Interface"means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries"of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source"for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your

behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless

of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate"if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain

clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product"is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used"refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information"for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself

materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions"are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of

liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions"within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of

the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction"is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor"is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims"are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control"includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license"is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant"such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying"means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory"if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions

of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Remote Network Interaction; Use with the GNU General Public License.

Notwithstanding any other provision of this License, if you modify the Program, your modified version must prominently offer all users interacting with it remotely through a computer network (if your version supports such interaction) an opportunity to receive the Corresponding Source of your version by providing access to the Corresponding Source from a network server at no charge, through some standard or customary means of facilitating copying of software. This Corresponding Source shall include the Corresponding Source for any work covered by version 3 of the GNU General Public License that is incorporated pursuant to the following paragraph.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the work with which it is combined will remain governed by version 3 of the GNU General Public License.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU Affero General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU Affero General Public License "or any later version"applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU Affero General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU Affero General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of

your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS"WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright"line and a pointer to where the full notice is

found.

<one line to give the program's name and a brief idea of what it does.> Copyright (C) <year> <name of author>

This program is free software: you can redistribute it and/or modify it under the terms of the GNU Affero General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License along with this program. If not, see <https://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

If your software can interact with users remotely through a computer network, you should also make sure that it provides a way for users to get its source. For example, if your program is a web application, its interface could display a "Source"link that leads users to an archive of the code. There are many ways you could offer source, and different solutions will be better for different programs; see section 13 for the specific requirements.

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer"for the program, if necessary. For more information on this, and how to apply and follow the GNU AGPL, see <https://www.gnu.org/licenses/>.