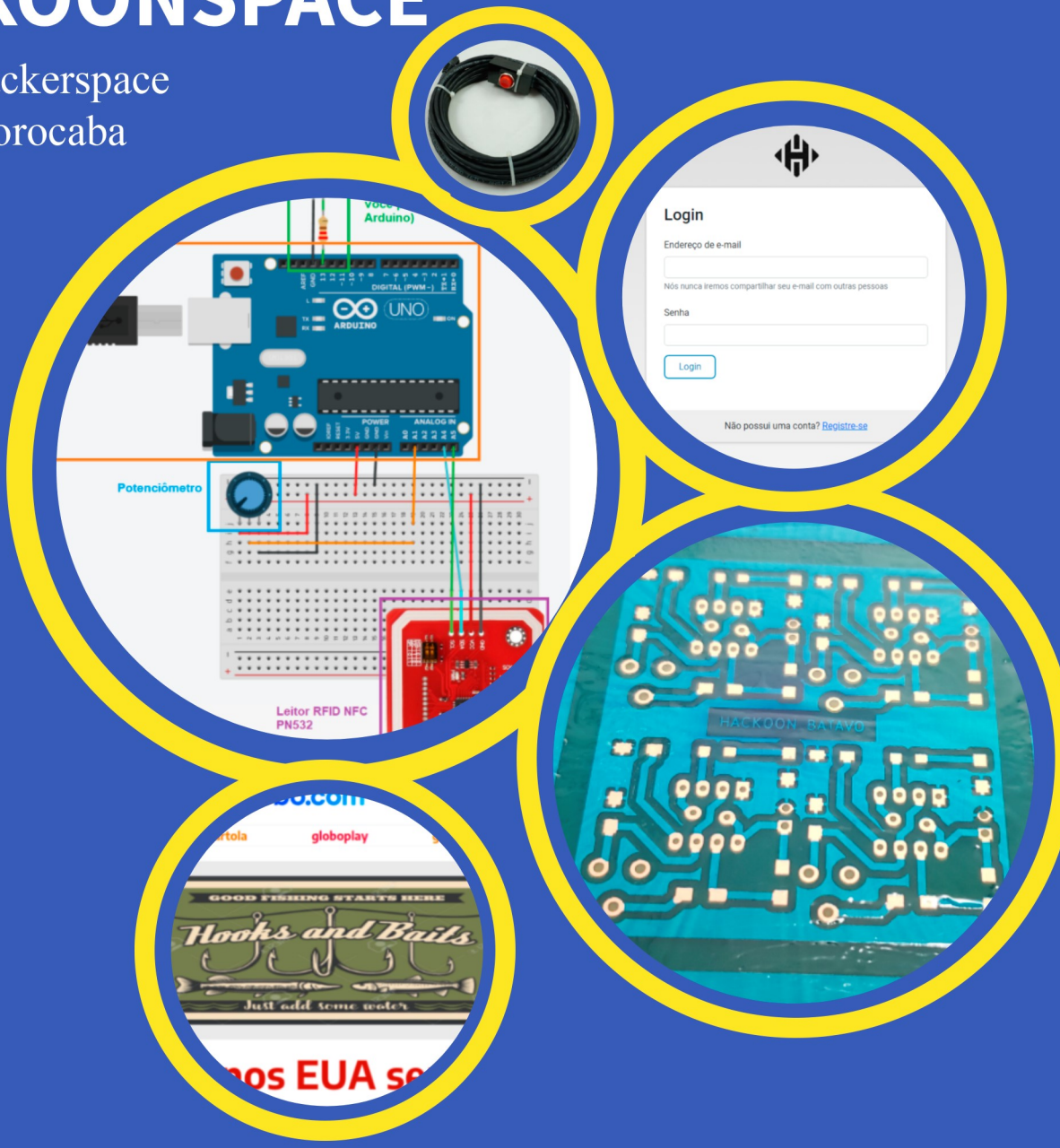
VOL. 4
2022

Revista HACKOONSPACE

Projeto Hackerspace
UFSCar Sorocaba



Apresentação dos artigos e projetos realizados na edição 2022 do HackoonSpace

Universidade Federal de São Carlos
Campus Sorocaba

PROEX
Pró-Reitoria de Extensão

Revista Hackoonspace

Vol. 4

Projeto Hackoonspace
2022

Apresentação

O Projeto Hackoonspace é um projeto de extensão promovido pela UFSCar Campus Sorocaba e realizado na forma de encontros que discutem a contracultura hacker, apresentando personagens, acontecimentos, aspectos socioculturais, artefatos e atividades com a finalidade de que os participantes sejam expostos à contracultura em questão e desenvolvam um projeto ou material acerca da mesma.

A principal intenção do projeto é desmistificar o conceito e figura do hacker, enquanto proporcionando um espaço de exposição, produção e compartilhamento de conteúdo que se configura como dentro da contracultura de forma que participantes de diferentes níveis de conhecimento técnico possam participar.

Esta revista tem como objetivo divulgar os trabalhos desenvolvidos pelos alunos participantes do Projeto Hackerspace no ano de 2022. Esperamos que esta revista possibilite a difusão dos conhecimentos adquiridos na atividade para a comunidade em geral. Gostaríamos de agradecer todos os alunos que contribuíram com artigos e projetos para esta edição da revista.

Organização e edição:
Caio César Brandini
Fernando Favareto Abromovick
Joatan da Silva Marques
Vinícius Carvalho Venturini

Supervisão:
Gustavo M. D. Vieira

Conteúdo

Parte I

Artigos

Capítulo 1

Os estigmas associados à comunidade hacker

Amanda Akemi Perina Kouchi
Daniel Oliveira Rocha
Maria Fernanda Machado
Rayana Saboya Modanez

OBS: Para ver o artigo na íntegra, com imagens maiores, clique **aqui.**

1.1 Resumo

Este trabalho busca apresentar, discutir e desmistificar o estereótipo presente na sociedade a respeito dos *Hackers*, majoritariamente vistos apenas como criminosos virtuais; comparando a visão de pessoas mais ligadas ao ambiente virtual (seja por meio de sua área de atuação profissional ou faixa etária) com a das pessoas que não possuem um contato tão frequente com este tipo de tecnologia. Para isso, foram realizadas duas pesquisas de campo: uma em forma de formulário e outra em forma de entrevista, indagando pessoas de diferentes profissões e faixas etárias em relação às suas percepções quanto aos *Hackers*.

A princípio, tem-se por objetivo comprovar este estereótipo e abordar o conceito de *Hacker*, a diferença entre *Hackers* e Crackers (com base na forma como os termos são definidos dentro da própria comunidade *Hacker*) e a origem desse estereótipo errôneo, que se dá principalmente pela forma como a mídia aborda o assunto.

Palavras-chave: *Hackers*; *Hacking*; estigma; estereótipo; *Crackers*; invasão; mídia; sociedade.

1.2 Introdução

Quando se ouve falar no termo *Hacker*, logo surge na mente a imagem tão divulgada pela mídia, de pessoas invadindo sistemas do governo para obter informações extremamente sigilosas, quebrando contas pessoais em redes sociais por pura diversão ou então para se passar por alguém, de forma a causar grandes transtornos à vida de outras pessoas apenas para benefício próprio.

Mas o que exatamente é um *Hacker*? A mídia realmente passa uma imagem correta a respeito do assunto? As pessoas sabem diferenciar a ficção da realidade e entendem que na vida real as coisas são um pouco diferentes? Este artigo busca falar sobre esta visão estereotipada que a sociedade tem a respeito do assunto, abordando sua origem e o verdadeiro significado da palavra *Hacker*.

1.3 *Hacker* - A Origem do Termo

Atualmente ser um *Hacker* pode ser considerado ser alguém que burla as regras do sistema utilizando inteligência e ideias criativas, ao analisar por este aspecto pode ser considerado que os primeiros *Hackers* surgiram em 1870 onde há relatos de adolescentes que encontraram meios para burlar o sistema telefônico da época, entretanto é claro que o termo "*Hacker*" não existia, é válido

pontuar também que a notoriedade da cultura *hacker* tomou maior proporção cerca de 40 anos para cá.

Diferente da concepção atual, o termo em questão nasceu a fim de rotular indivíduos com conhecimentos profundos na área da computação, a origem do termo data da década de 60 no *"Tech Model Railroad Club"* do Instituto de Tecnologia de Massachusetts (MIT). *Hacker* era a designação concedida aos membros do clube que desenvolviam estratégias e táticas para resolver problemas. Nos dias atuais, entretanto, o termo *"hacker"* é usado popularmente de maneira arbitrária, tanto para descrever nomofóbicos quanto programadores geniais. É comumente notado que entre os *hackers* existe a vontade de destrinchar minuciosamente as atribuições dos sistemas e das redes. Programadores de *Software Open Source* por via de regra se autodenominam *hackers* e atribuem a seus colegas esse mesmo rótulo, sendo este visto de uma maneira honesta e digna.

Nos anos 80, a internet chegou aos computadores, no decorrer disso aconteceram as primeiras detenções em consequência das invasões de computadores, devido a isso foram presos grupos de *Hackers* acusados de invadir mais de 60 computadores, nesse período era comumente visto grupos de *hackers* se unindo em prol de uma causa, os mais populares eram *"Legion of Doom"* nos EUA e os *"Chaos Computer Club"* na Alemanha. Em consequência disso, nessa época os *Hackers* passaram a ser olhados como criminosos e nasceu então o termo *Cracker* para indicar indivíduos inteligentes que usavam seu conhecimento para atividades ilegais.

Com a popularização da internet nos anos 90 muitos jovens passaram a ter acesso à internet, nasceu assim no coração de muitos jovens a paixão pelo assunto, logo a cultura *Hacker* estava sendo difundida em larga escala, em razão disso os anos 90 foram marcados pelo crescimento contínuo dos vírus de computador, dos ataques *Hackers* em larga escala e também pelo crescimento da vexação aos *Hackers*. Neste período, *Hackers* invadiram o sistema de grandes organizações como a base aérea de *Griffith*, a NASA e o instituto de pesquisa atômica coreano. E talvez o caso mais popular da década tenha sido quando Kevin Mitnick foi preso no centro de supercomputadores de San Diego por ter sido caçado por Tsumoru Shimomura.

Entretanto o maior ataque foi contra o site do Yahoo! quando *Hackers* disseminaram uma mensagem de caso Kevin Mitnick não fosse liberado uma bomba lógica acometeria muitos computadores. No fim dos anos 90 foi criado o Centro de Proteção Nacional que visa proteger a comunicação, a tecnologia e os sistemas de transportes dos *Hackers*.

Crackers são ainda um grande problema para usuários da internet, o interessante é que isso gera todo mundo mercado extremamente lucrativo onde inúmeras empresas lucram, com antivírus, *Firewalls* etc. Contratando *Hackers* para desenvolver tais tecnologias.

1.4 Tipos de *Hackers*

Hacker é uma expressão originada no âmbito da informática, muitas vezes utilizada de forma negativa, para nomear alguém capaz de invadir dispositivos e redes com intenção de praticar atos ilegais. No entanto, ser um *hacker* tem um significado muito mais amplo. Um *Hacker* não é necessariamente uma pessoa mal-intencionada, a palavra *Hacker* pode ser usada para designar alguém que usa seus conhecimentos de computação para verificar a segurança de um sistema e modificá-lo, com o objetivo de aperfeiçoá-lo, para que a quantidade de falhas seja menor.

A conotação negativa surgiu na década de 90, quando muitos programadores começaram a usar seus conhecimentos e habilidades com redes e sistemas para roubar informações e aplicar golpes bancários, ignorando a ética estabelecida pela comunidade Hacker. Depois disso, outros termos foram criados para dividir os diferentes tipos de *Hackers*, de acordo com suas intenções e ações.

Crackers/Black Hats

Os *Crackers*, ou *Black Hats* (termo mais atual) são aqueles que usam seus conhecimentos para praticar atos ilegais, danificando sistemas a partir de suas vulnerabilidades, visando obter algum tipo de benefício, como roubar senhas e dados, sabotar sites, implantar vírus e aplicar golpes. Ou seja, são criminosos cibernéticos, cujo único objetivo é quebrar sistemas e conseguir lucros.

White Hats

Conhecidos como *White Hats* ou *Hackers* Éticos, são os usuários que trabalham em prol da segurança. Esse tipo de *hacker* age de acordo com as leis, invadindo sistemas apenas quando são permitidos.

Seguindo um padrão ético, esses *Hackers* verificam a existência de falhas de segurança em sites e empresas, tomando as providências necessárias para minimizar esses erros, com o objetivo de melhorar os sistemas.

Além da área de segurança, os *White Hats* também atuam em setores ligados à engenharia de projetos, desenvolvimento de *softwares*, perícia forense, estão de riscos e pesquisas de vulnerabilidade, direcionando todo o seu conhecimento e habilidades em informática para otimizar *softwares* e redes.

Gray Hats

Chamados de chapéus cinza são os *Hackers* que atuam entre os *White Hats* e os *Black Hats*. Eles não são necessariamente éticos, mas também não atuam com malícia. Diferentemente dos *White Hats*, os *Gray Hats* não são confiáveis, mas os seus objetivos são basicamente os mesmos, por isso que não podem ser chamados de *Crackers*. Os chapéus cinzas podem invadir sistemas ou sites (sem autorização prévia), evidenciando suas falhas, mas, ao invés de usar as informações para aplicar golpes ou agir como criminosos, eles costumam divulgar à empresa a vulnerabilidade, pedindo algum tipo de pagamento em troca.

Outros tipos de *Hackers* menos conhecidos

- *Green Hats*: São chamados assim por serem "verdes", ou seja, são *Hackers* inexperientes e com poucas habilidades. Apesar de não intencional, esses usuários podem causar danos, já que muitas vezes não sabem das consequências de suas ações e nem como consertar os seus estragos.

- *Blue Hats*: Esse tipo de *Hacker* pode ser dividido em 2. A primeira divisão são aqueles que buscam vingança, ou seja, esses não buscam dinheiro ou fama, mas alguma forma de se vingar, seja roubando dados ou inserindo vírus em um sistema, apenas para destruir a imagem e credibilidade de uma empresa, por exemplo. A segunda divisão dos *Blue Hats* engloba os *Hackers* responsáveis por testes de segurança, ou seja, são profissionais contratados por grandes empresas para realizar testes de vulnerabilidade periódicos antes de um produto ser lançado.

- *Red Hats*: Têm os mesmos objetivos dos *White Hats*, mas usam meios extremos e muitas vezes ilegais para alcançar suas metas. Os *Red Hats* frequentemente se infiltram em comunidades *Crackers* para atacar as redes e os sistemas dos criminosos cibernéticos, ou seja, usam métodos ilícitos para fazer coisas boas.

1.5 Análise de Dados

Como dito anteriormente, a imagem que muitas pessoas possuem do termo *Hacker* é a de um criminoso virtual; é muito comum ouvir alguém usando esse termo para se referir a pessoas que invadem ilegalmente contas pessoais (seja em redes sociais, lojas, contas bancárias...), sites de grandes empresas, dentre outros.

A fim de comprovar a existência de tal estereótipo, foi aplicado um questionário para pessoas

de diversas faixas etárias e profissões diferentes, questionando seus conhecimentos a respeito da palavra *Hacker*.

Resultados Esperados

Por conta da grande estereotipação de *Hackers* vinda principalmente das mídias de filmes de ficção, mas também de portais de notícias, é esperado que a maioria das pessoas não soubesse o que significa ser um *Hacker*, e trouxessem visões como as dos filmes de ficção, onde nos são apresentados *hackers* invadindo de forma ilegal grandes empresas, bancos ou até mesmo contas pessoais de pessoas importantes, tudo isso de uma maneira incrivelmente fácil e rápida.

Também espera-se que as pessoas que souberem responder o que é um *Hacker*, e tiverem consciência da importância do trabalho dos *Hackers* para melhorar a segurança de sistemas (ou ainda souberem diferenciá-los dos *Crackers/Black Hats*), serão pessoas que já possuem um maior contato com a tecnologia, sejam essas pessoas que trabalham na área, ou então pessoas mais novas que, além de possuir um grande contato com a tecnologia desde cedo, tendem a ter uma "mente aberta" para esse tipo de assunto.

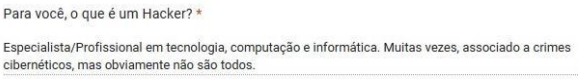
Resultados Obtidos

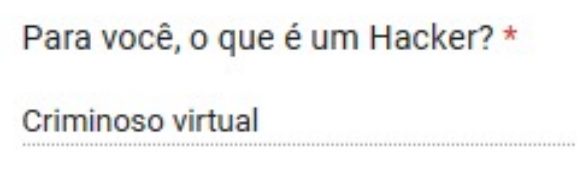
Questionário

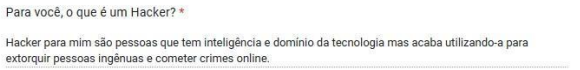
As pessoas que responderam o questionário precisavam indicar sua faixa etária, área de atuação profissional ou acadêmica (caso não tivesse, responder seu nível de escolaridade) e responder à pergunta "Para você, o que é um *Hacker*?". No total, obtivemos 177 respostas ao questionário, das quais três foram desconsideradas por não levarem a pesquisa a sério e serem respondidas de forma irônica ou ofensiva. Sendo assim, esta análise foi feita em cima de 174 respostas ao todo. Para a análise, as respostas obtidas para a última pergunta foram separadas em três categorias:

Categoria 1: Nesta categoria estão presentes as respostas das pessoas que demonstraram saber minimamente ou aprofundadamente o que é um *Hacker*, ou então pessoas que souberam dizer que é possível usar o *hacking* tanto para fins maliciosos quanto para fins benéficos, dependendo da situação. Foram obtidas 42 respostas dessa categoria, sendo um exemplo:

Categoria 2: Nesta categoria estão presentes as respostas das pessoas que associam *Hackers* a criminosos, sejam estes pessoas que invadem ilegalmente contas pessoais, sistemas de grandes empresas, dentre outros. Foram obtidas 65 respostas dessa categoria, sendo alguns exemplos:





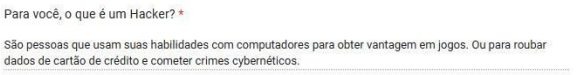


Categoria 3: Nesta categoria, estão presentes as respostas das pessoas que não souberam dizer com precisão o que é um *Hacker*, mas que não associaram o termo necessariamente a algo bom ou ruim. No geral, a maioria das respostas diziam apenas tratar-se de alguém com um amplo conhecimento em computadores e informática. Foram obtidas 67 respostas dessa categoria, sendo alguns exemplos:

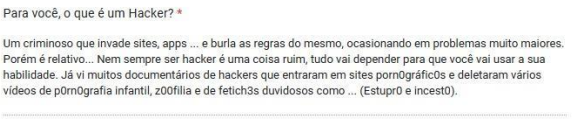
Como já era esperado, a menor parte das respostas se encaixa na categoria 1, tendo em vista que a maior parte associa o termo *Hacker* ou a criminosos virtuais, ou apenas a pessoas com um grande conhecimento de informática, não

sabendo exatamente que tipo de conhecimento é esse e onde ele é aplicado.

Ao longo do questionário, surgiram muitas respostas interessantes, algumas reforçando os estereótipos já ditos anteriormente, e outras trazendo pautas diferentes, como por exemplo o *hacking* no mundo dos jogos digitais:



Algumas pessoas também trouxeram respostas tratando de assuntos de extrema importância, como *Hackers* que invadem sites de procedência duvidosa para remover e denunciar publicações criminosas, como a pauta trazida pela resposta abaixo:



Tais respostas mostram que a visão das pessoas a respeito do assunto vai muito além do que se era esperado, fazendo relações com jogos digitais, onde muitos se utilizam do *hacking* para obter vantagens; além do fato de nos lembrar que *Hackers* não só trabalham encontrando falhas de segurança para ajudar empresas a melhorarem seus sistemas, como também podem auxiliar na investigação de crimes extremamente graves (sejam estes *Hackers* que trabalham para a polícia, ou então apenas que fazem este tipo de tarefa por conta própria para ajudar a sociedade).

Além dessa análise feita de forma geral, também iremos abordar os diferentes padrões de resposta de acordo com as diferentes faixas etárias e, posteriormente, ramos de atuação acadêmica ou profissional (verificando apenas se a pessoa trabalha na área da informática, computação, TI, ou em alguma outra área de tecnologia; e a diferença de respostas das pessoas que estão inserida nesse meio para as que não estão).

As faixas etárias foram divididas da seguinte forma:

- **Menos que 12 anos:** 0 respostas.
- **Entre 12 e 19 anos:** 72 respostas.

Como esperado, as pessoas dentro dessa faixa etária (que foram as mais novas a serem entrevistadas) apresentaram um conhecimento um pouco mais amplo que os demais a respeito do assunto, isso pode ocorrer por essa geração ter um contato maior com a tecnologia desde muito cedo, muitas vezes já conhecendo o termo *Hackers* e até mesmo podendo ter um certo interesse pela profissão, que é cada vez mais falada nos dias de hoje.

No total, foram vinte e três respostas classificadas na categoria 1, dezenove na categoria 2, e trinta na categoria 3; tornando esta a única faixa etária na qual as respostas de categoria 2 foram minoria, ou seja, foi a faixa etária que menos trouxe estereótipos "ruins" a respeito dos *Hackers*.

Quatorze pessoas desse grupo afirmaram atuar na área da tecnologia, sendo que destas, nove respostas foram da categoria 1, cinco foram da categoria 3, e nenhuma foi da categoria 2. Sendo assim, as pessoas que já atuam nessa área foram capazes, em sua maioria, de responder com maior propriedade, e nenhuma afirmou ver a "imagem *hacker*" como algo criminoso e desonesto.

- **Entre 20 e 39 anos:** 51 respostas.

Nesta faixa etária, já houveram grandes diferenças em relação à anterior, tendo as respostas de categoria 1 agora como minoria. Porém, as respostas que mais prevaleceram aqui foram as

de categoria 3, que apesar de não trazerem os estereótipos tratados anteriormente, demonstram uma falta de conhecimento sobre o assunto, ou uma opinião não formada.

No total, foram nove respostas classificadas na categoria 1, dezessete na categoria 2, e vinte e cinco na categoria 3.

Seis pessoas desse grupo afirmaram atuar na área da tecnologia, sendo que todas deram respostas de categoria 1, apresentando um maior entendimento a respeito da profissão. Sendo assim, a maior parte das respostas dessa categoria (aproximadamente 66,6%) vieram de pessoas que já estavam de alguma forma mais inseridas no meio.

- **Entre 40 e 59 anos:** 30 respostas.

A partir desta faixa etária, as respostas de categoria 2 passam a ser maioria, ressaltando inúmeras vezes a ideia de que *Hackers* são criminosos que roubam dados de pessoas e organizações para usarem a seu favor.

No total, foram sete respostas classificadas na categoria 1, dezenove na categoria 2, e quatro na categoria três.

Nenhuma pessoa desse grupo disse atuar na área da tecnologia, o que talvez possa ser o principal motivo da maioria das respostas estarem presentes na categoria 2.

- **60 anos ou mais:** 21 respostas.

Como também já era esperado, assim como na faixa etária anterior, as pessoas mais velhas a responderem o questionário também tiveram a maioria das respostas classificadas na categoria 2, mostrando, novamente, o pensamento estereotipado sobre a profissão aparecendo com bastante frequência.

No total, foram três respostas classificadas na categoria 1, dez na categoria dois, e oito na categoria 3.

Duas pessoas desse grupo afirmaram atuar na área da tecnologia, sendo que destas, uma resposta foi da categoria 2, e uma foi da categoria 3. Sendo assim, até mesmo a resposta de uma pessoa que já está inserida na área acabou por reforçar estereótipos ruins do termo *Hacker* e foi classificada na categoria dois, algo que a princípio não era esperado, porém pode vir a ter alguma relação com a faixa etária analisada, que mesmo atuando na área atualmente, não está desde sempre inserida neste meio.

Entrevista

Além do questionário, também foram feitas entrevistas presenciais, onde nove pessoas foram entrevistadas no total, e as mesmas perguntas foram feitas. Todos os entrevistados responderam seu nível de escolaridade e nenhum especificou atuar na área da tecnologia, portanto, o foco da análise das entrevistas serão as diferentes faixas etárias entrevistadas.

As duas primeiras pessoas tinham 17 anos e foram entrevistadas em conjunto. Ambas não associaram *Hackers* a algo "bom" ou "ruim", disseram depender da maneira e objetivo para o qual você se utiliza das técnicas de *Hacking*, citando até a remoção de vídeos envolvendo conteúdos sensíveis da internet.

As terceira, quarta, quinta e sexta pessoas a serem entrevistadas tinham respectivamente 21, 63, 17 e 33 anos. Todas demonstraram ter uma visão negativa de *Hackers*, relacionando o termo a palavras como "invasão" e "perseguição", ou apenas dizendo ser algo ruim. A sétima pessoa a ser entrevistada também tinha 17 anos e disse existir *Hackers* bons e *Hackers* ruins. O mesmo ocorreu com a oitava pessoa (de 57 anos), que afirmou que alguns *Hackers* ajudam a sociedade enquanto outros não, disse que seu trabalho poderia ser necessário em alguns casos, porém terminou seu discurso dizendo que espera nunca encontrar um em sua vida.

Por fim, a última pessoa a ser entrevistada tinha 69 anos, e diretamente se referiu a *Hackers*

como sendo, em suas palavras, “bandidos”.

Dessa forma, dentre as quatro pessoas com menos de 18 anos, apenas uma demonstrou ter uma visão 100% negativa a respeito dos *Hackers*, enquanto o restante das pessoas mais novas a serem entrevistadas trouxeram uma visão não necessariamente ruim da profissão, reconhecendo e valorizando o trabalho de *Hackers* bons.

O mesmo já não ocorre dentro das outras faixas etárias, onde apenas uma pessoa, de 57 anos, disse que talvez *hackers* fossem necessários para a sociedade, apesar de ainda demonstrar um certo receio quanto a isso. O restante das pessoas mais velhas a serem entrevistadas, como já era esperado, associaram o termo a algo ruim, criminoso e invasivo.

Link para a entrevista: `https://youtu.be/hjUxTxgCr84`

*Todas as pessoas presentes no vídeo autorizaram o uso de sua imagem para fins acadêmicos.

1.6 A Origem e Perpetuação do Estigma na Comunidade *Hacker*

A comunidade *Hacker* surgiu na década de 1960, composta por programadores curiosos e talentosos que exploravam os limites dos sistemas de computadores, descobrindo falhas e criando soluções inovadoras. No entanto, com a popularização da internet e o aumento da conectividade, a comunidade passou a ser associada a atividades ilegais, como invasões de sistemas, roubo de informações e ataques cibernéticos. Essa associação negativa foi atribuída a pessoas mal-intencionadas, conhecidas como "*Crackers*", que cometeram crimes cibernéticos, tendo sido divulgadas pela mídia como gênios do crime ou terroristas cibernéticos. Como resultado, a comunidade *Hacker* passou a ser vista com desconfiança e medo, sendo estigmatizada como uma ameaça à segurança cibernética.

O Papel da Mídia na Perpetuação do Estigma Negativo Associado à Comunidade *Hacker*

A mídia desempenha um papel importante na formação da opinião pública, em relação aos *Hackers* não seria diferente, muitas vezes apresentando uma abordagem equivocada e sensacionalista. Quando um grupo de *Hackers* ataca uma empresa ou instituição governamental, por exemplo, é comum que a mídia trate o caso como um grande escândalo, reforçando a imagem dos *Hackers* como criminosos perigosos. Em muitos filmes e séries de TV, os *Hackers* são retratados como indivíduos anti-sociais, isolados e ameaçadores, contribuindo para a formação de um estereótipo negativo. Um exemplo disso é a série infantil "Cyberchase", onde o personagem *Hacker* é retratado como um vilão malvado e perigoso, sempre tentando sabotar os protagonistas em suas aventuras. Já a série "Mr. Robot", que estreou em 2015 e teve quatro temporadas, apresenta o personagem principal Elliot Alderson, um *Hacker* altamente habilidoso que usa suas habilidades para derrubar grandes corporações e instituições financeiras. Embora Elliot seja retratado como um personagem complexo com motivações que vão além do simples desejo de causar o caos, suas ações são claramente vistas como negativas e ilegais ao longo da série. Ambos os exemplos contribuem para a percepção popular acerca dos *Hackers*, levando muitas pessoas a crer que eles possuem maus e perigosos propósitos, capazes de causar danos significativos com seu conhecimento.

Por outro lado, a mídia pouco destaca trabalhos legítimos realizados por *Hackers*, como a identificação de falhas de segurança em sistemas ou a contribuição para o desenvolvimento de tecnologias. Quando *Hackers* realizam trabalhos éticos e legais, a mídia não dá a mesma cobertura concedida a notícias de ataques cibernéticos bem-sucedidos realizados por *Hackers* mal-intencionados. Essa falta de equilíbrio na cobertura da mídia pode influenciar negativamente a percepção do público em relação aos *Hackers* e à toda comunidade *Hacker*, reforçando estereótipos negativos e alimentando o medo e o preconceito em relação a essa comunidade.

Um exemplo disso é o caso de Kevin Mitnick, *Hacker* e ativista que se tornou conhecido na década de 1990 por invadir sistemas de grandes empresas e agências governamentais. Após ser preso e cumprir pena de cinco anos, ele se dedicou a trabalhar como consultor de segurança cibernética, ajudando empresas a identificar e corrigir falhas em seus sistemas. No entanto, apesar de seu trabalho ético e não ilegal após sua prisão, a mídia pouco se importou em noticiar suas contribuições positivas para a segurança cibernética. Esse caso ilustra como a mídia pode perpetuar o estigma negativo associado à comunidade *Hacker* ao negligenciar trabalhos legítimos realizados por *Hackers* e enfatizar apenas suas atividades ilegais.

1.7 Conclusão

Podemos afirmar que a sociedade *Hacker*, originalmente formada por programadores curiosos e talentosos, acabou sendo estigmatizada e associada a atividades ilegais como *Hacking* de sistemas, extorsão de informações e ataques cibernéticos. Essa coligação negativa foi atribuída a indivíduos mal-intencionados, conhecidos por "*Crackers*", que cometeram crimes cibernéticos e foram retratados pela mídia como mentores da violação ou ciberterroristas.

No entanto, é essencial recordar que a comunidade *Hacker* nunca se limita a atividades ilegais. E há muitos *Hackers* trabalhando de forma ética e lícita para fomentar a cibersegurança e a tecnologia avançada. Lamentavelmente, as mídias muitas vezes perpetuam o estigma maléfico agregado aos *Hackers*, apresentando abordagens equivocadas e sensacionalistas, retratando os *Hackers* como vilões horrendos e reforçando a figura dos *Hackers* como criminosos perigosos.

Consequentemente, é vital inferir que a comunidade *Hacker* é diversa e que existem diversos tipos de *Hackers*, com diferentes motivações e objetivos e assegurar-se de que a mídia forneça uma cobertura equilibrada e equitativa e se concentre não exclusivamente em atividades ilegais realizadas por *Hackers* mal-intencionados, mas igualmente em atividades

legítimas realizadas por *Hackers* éticos e legais, bem como em segurança e tecnologia. Além disso, é essencial que a sociedade como um todo compreenda a relevância da segurança cibernética e da defesa de dados pessoais, incentivando o uso de senhas fortes, atualizações de *Software* e a aceitação de medidas de segurança para precaver ataques cibernéticos. A comunidade *Hacker* pode ser uma grande aliada nesse segmento, ajudando a reconhecer falhas de segurança em sistemas e contribuindo para o desenvolvimento de soluções inovadoras e eficazes para escudar a privacidade e a cibersegurança.

1.8 Bibliografia

[1] Hacker: o que é, origem do nome, cracker x hacker. URL: https://brasilescola.uol.com.br/informatica/o-que-e-hacker.htm.

[2] Hacker types: Black hat, white hat, and gray hat hackers. URL: https://www.avast.com/c-hacker-types.

[3] Hackers do bem: são mocinhos ou bandidos? URL: https://www.kaspersky.com.br/resource-center/definitions/white-hat-hackers.

[4] O que é um hacker. URL: https://www.significados.com.br/hacker/#:~:text=Hacker%20%C3%A9%20uma%20palavra%20da.

[5] O que é um hacker? URL: https://canaltech.com.br/hacker/O-que-e-um-Hacker/.

[6] Qual a diferença entre hacker e cracker? URL: https://codebit.com.br/blog/developer/qual-diferenca-entre-hacker-cracker.

[7] D. BBC. Saiba mais sobre a história dos hackers. URL: https://g1.globo.com/tecnologia/noticia/2011/06/saiba-mais-sobre-a-historia-dos-hackers.html.

[8] A. FROEHLICH. What is a white hat hacker? URL: https://www.techtarget.com/searchsecurity/definition/white-hat.

[9] M. MEHTA. Different types of hackers: the 6 hats explained. URL: https://sectigostore.com/blog/different-types-of-hackers-hats-explained/.

[10] E. T. START. Quem é kevin mitnick e o que ele fez - tech start xyz. URL: https://techstart.xyz/hacking/quem-e-kevin-mitnick/.

[11] R. H. TECH. Hackers, origem e história. URL: https://hackertech.com.br/hackers-origem-e-historia/.

Capítulo 2

Utilizando *Debugger* em C no VScode

Maurício Marques da Silva Junior

OBS: Para ver o artigo na íntegra, com imagens maiores, clique **aqui.**

2.1 Introdução

Um debugger (ou depurador em português) tem como objetivo identificar erros no código fonte, com isso o programa pode ser abortado no meio da depuração, por vários motivos como acessar memória inexistente ou indisponível para ele, uma instrução incorreta para a CPU, erros de digitação, funções mal elaboradas e etc. Por sua vez o debugger oferece funções como: a execução do programa linha por linha, suspensão do programa para analisar seu estado atual- valor de variáveis ou parâmetros, pilha de execução - a partir de um ponto de parada (breakingpoint) predefinido, além de ser possível acessar funções no meio do código e ver seu funcionamento até o seu retorno, para chegar ao ponto de parada. Partindo desse pressuposto o artigo a seguir tem como objetivo ser um tutorial para rodar um debbuger da linguagem C no editor de texto VScode.

2.2 Pré-Requisitos

OBS: Os passos dos pré-requisitos abaixo, são para ser executados em um terminal linux ou usuários de WSL no Windows. Porém para instalação no windows há uma seção chamada "Configurando e instalando *Debugger* da linguagem C no Windows" ou você pode clicar nesse link.

Compilador para C

Primeiro é necessário instalar o compilador **gcc** para C e C++, no linux Debian(e derivados como o Ubuntu, Mint, Kubuntu...etc). Para isso é necessário o pacote **build-essential** que é, na verdade, uma referência (lista) para todos os pacotes que são necessários para compilar programas (pacotes) no Debian. Ele inclui diversos elementos, tais como o compilador gcc (para C), compilador g++ (para C++), a ferramenta make, diversas bibliotecas e utilitários.

Para instalar o pacote build-essential no Debian (e derivados) basta usar o **gerenciador de pacotes apt**– lembrando que é necessário estar logado como root ou possuir **privilégios de administrador (sudo)**

```
$ sudo apt update
$ sudo apt install build-essential
```

Figura 2.1: atualizando o sistema e instalando o pacote build-essential

Depois, verifique a versão do GCC:

```
❯ gcc --version
gcc (Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0
Copyright (C) 2019 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Figura 2.2: Checando a versão do GCC

GDB: GNU *Debugger*

Outro requisito necessário para "debugar" é uma ferramenta para tal obviamente, com isso se tem o **GDB**. Esse programa permite que você veja o que está acontecendo "dentro" de outro programa que está em execução - ou o que um outro programa estava fazendo no momento que falhou na execução.

Para instalar use os seguintes comandos:

```
$ sudo apt-get update
$ sudo apt-get install gdb
```

Figura 2.3: Instalando o GDB

Depois, verifique a versão do GDB:

```
❯ gdb --version
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.1) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

Figura 2.4: Checando a versão do GDB

Instale a extensão "C/C++"

Clique no ícone de extensões:

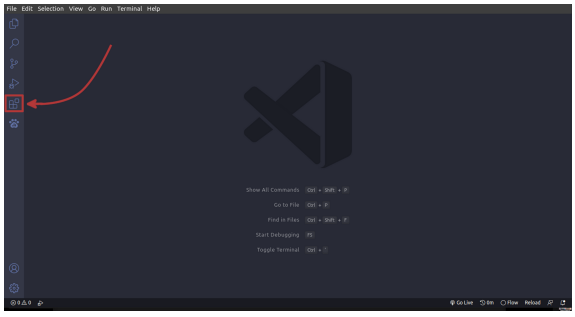


Figura 2.5: Localizando as extensões do VSCode

Pesquise por "C/C++" e clique em *install*, na primeira extensão que aparecer da Microsoft:

Figura 2.6: Baixando a extensão "C/C++"

Aguarde a instalação e pronto!

2.3 Configurando o *Debugger* no VScode

Com os pré-requisitos instalados, chega o momento mais denso do tutorial que se trata da configuração.

Configurando o botão "Run and Debug"

Com o VScode aberto é possível observar uma barra a esquerda com alguns ícones, porém vamos nos concentrar no ícone com um "play and bug", comece clicando no mesmo:

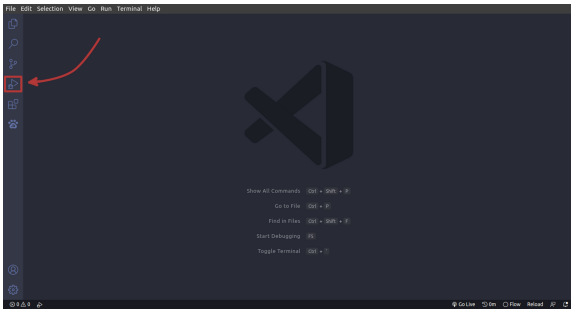


Figura 2.7: botão Run and Debug

Clique em “create a launch.json file”:

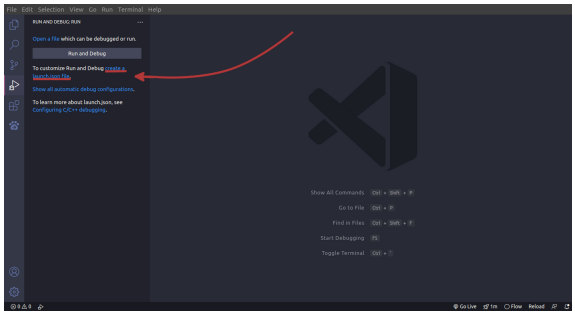


Figura 2.8: Localizando “create a launch.json file”

Clique em “Add Configuration...”:

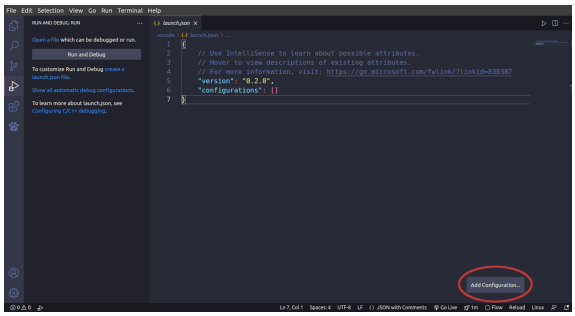


Figura 2.9: Localizando “Add configuration”

Adicione a configuração “(gdb) Iniciar” ou “(gdb) launch”:

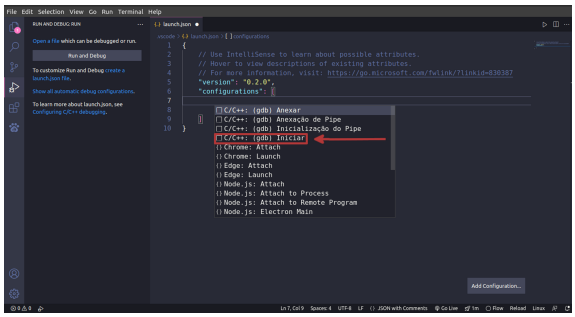


Figura 2.10: Localizando “(gdb) Iniciar”

Mude as seguintes linhas:

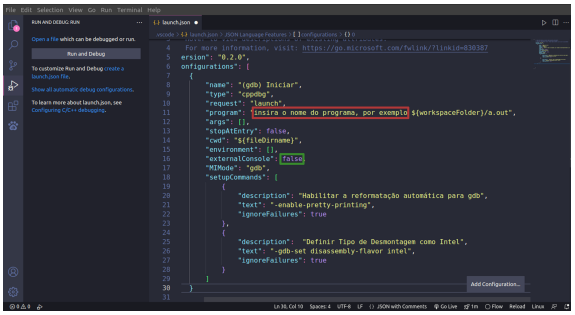


Figura 2.11: Linhas a serem mudadas”

Seleção em vermelho: Apague toda a seleção em vermelho. Tem de ficar da seguinte forma:

- No Linux: “$workspaceFolder/**a.out**”
- No Windows: “$workspaceFolder/**a.exe**”

Seleção em verde: Nessa parte de "externalConsole”, é possível colocar “true” para abrir um terminal externo ou “false” para abrir terminal interno do VScode - que é o recomendável.

Após isso salve (Ctrl+S) e pronto!!

2.4 Como depurar?

Essa parte do tutorial se trata de exemplos para para depurar, assim demonstrando como depurar(”debugar”) o programa.

Passo 1: Clique em “Open Folder”

Escolha a pasta onde esta o código do programa de extensão “.c”:

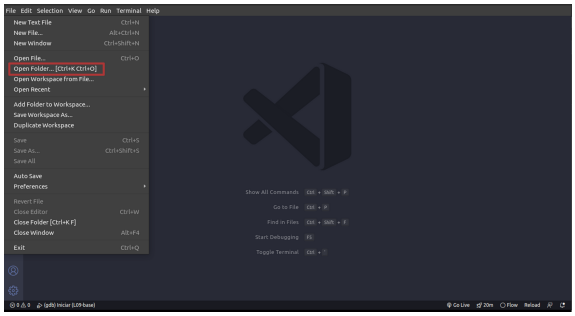


Figura 2.12: Abrindo o arquivo

Clique no arquivo do código:

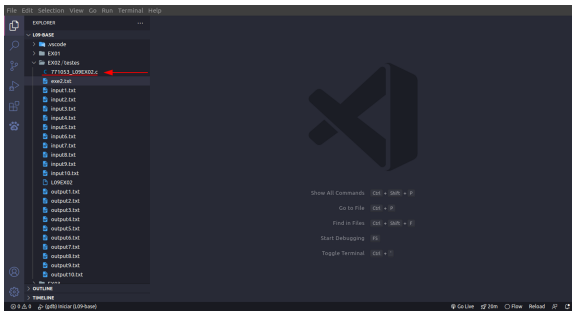


Figura 2.13: Localizando o arquivo

Passo 2: Rodando o gdb

Com a pasta aberta, clique no código do programa. Com o código aberto clique na barra a esquerda onde se encontram os números das linhas de código (parte destacada em roxo), e selecione onde deseja colocar os breakingpoints (pontos de parada destacados em vermelho), que se tratam de onde o programador deseja para o código para verificar variáveis, funções, estado e etc.

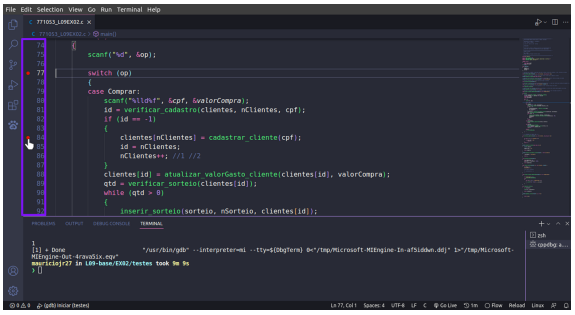


Figura 2.14: Linhas de código e breakingpoints

- OBS: O programador pode colocar quantos breakingpoints forem necessários.

Com os breakingpoints postos, agora será necessário rodar o depurador GDB com o seguinte código no terminal (interno do VScode ou externo, tanto faz desde que esteja na pasta do programa) e depois clicando na tecla “ENTER”:

```
$ gcc 771053_L09EX02.c -o a.out -g
```

Figura 2.15: Rodando o programa no GDB

- O parâmetro -o destacado em vermelho na instrução acima indica para que arquivo vai ser compilado o código, ou seja qual vai ser o executável;

- O parâmetro -g destacado em roxo na instrução acima indica ao compilador para incluir no executável informação extra para o GDB;

A partir desse ponto Clique no ícone do “Bug and play” novamente:

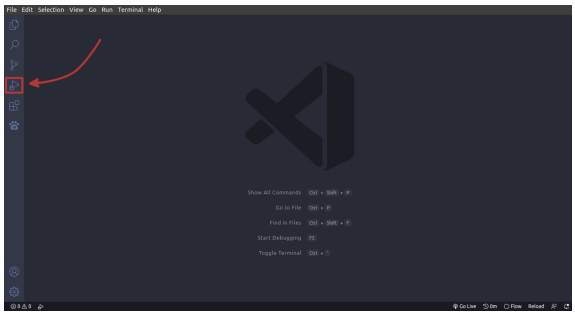


Figura 2.16: botão Run and Debug

Após isso clique no botão play em onde se encontra escrito “(gdb) iniciar” ou “(gdb) launch”:

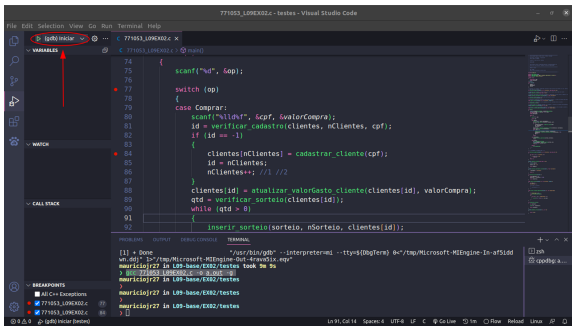


Figura 2.17: Rodando o programa no modo Debug

Após clicar no botão, o terminal vai ficar com o indicador piscando, e surgirão botões no canto superior e quadros (ou espaços) no lado esquerdo:

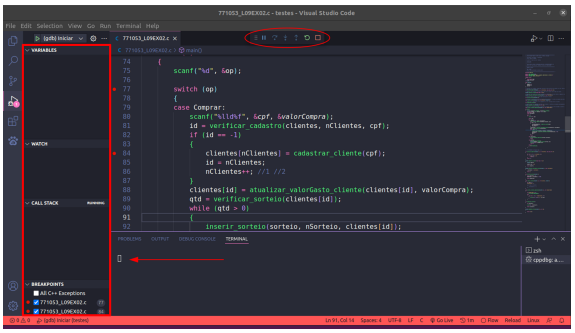


Figura 2.18: Janelas extras do modo Debug

O terminal estará piscando para serem inseridas as entradas do código. Após inserir-las a linha onde está o primeiro breakingpoint ficará destacada:

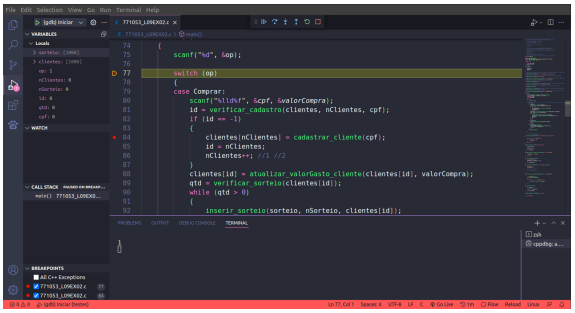


Figura 2.19: Programa parado na linha 77

E assim se inicia a depuração!

Passo 3: Funcionamento do layout do *debugger*

Com os conceitos da depuração tratados anteriormente, se faz necessário explicar o layout do *debugger* e seu funcionamento, a partir dos botões:

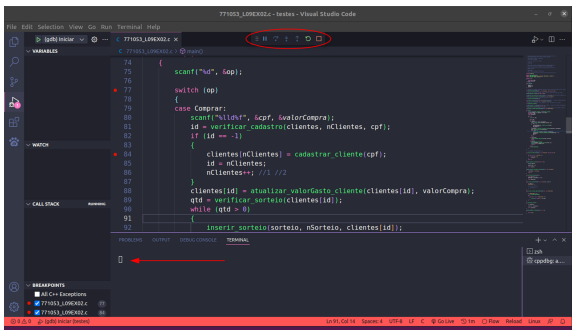


Figura 2.20: Layout de Debug

Figura 2.21: Novos botões de Debug

- **Continue ou Pause (botão com seleção laranja):** esse botão de *play* serve para rodar o código direto sem ver linha por linha, e assim o programador pode ver algum resultado rápido de um *loop* ou uma recursão por exemplo, e parar só para olhar um resultado específico;

- **Step over (botão com seleção amarela):** esse botão tem a funcionalidade de ir para a linha seguinte cada vez que é pressionado, assim proporcionando ao programador observar o código atentamente linha por linha, assim vendo o conteúdo das variáveis, ponteiros e etc;

- **Step into e Step out (botão com seleção roxa:** o botão da esquerda com seta para baixo serve para entrar em uma função chamada no meio do pseudo-código (o botão da seleção amarela o "step over", não adentra a função passa direto), assim o programador verifica o funcionamento dentro da função. Já a seta para cima a direita sai da função;

- **Restart (botão com seleção verde):** basicamente recomeça do zero a depuração;

- **Stop (botão com seleção vermelha):** responsável por parar a depuração;

Após a explicações dos botões existe a necessidade de se explicar a lateral onde se observa a Breakpoints (pontos de parada), Call Stack (pilha de chamadas) e Variables (variáveis):

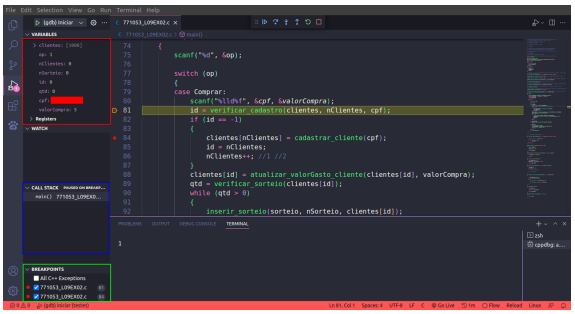


Figura 2.22: painel de parada do programa

- **Variables (seleção em vermelho):** nessa parte é possível o programador observar o valor de uma variável ou ponteiro, a cada linha do código;

- **Call Stack (seleção em azul):** nessa parte é possível observar a pilha de chamadas de funções que o programador realiza em seu código - ótimo pra observar e aprender o funcionamento de recursões;

- **Breakpoints (seleção em verde):** aqui é possível ver os pontos de parada e a linha onde estão, além de editar as condições de parada e remove-los;

2.5 Como depurar com arquivos de cabeçalho separados?

Às vezes há códigos que tem cabeçalhos em arquivos separados com extensão ".h", são arquivos que normalmente tem estruturas de dados

e protótipos de funções a serem seguidas ou consumidas e mais de um arquivo com extensão ".c". Porém para rodar o *debugger* o processo é diferente, existe fuma ordem e suas instruções no terminal.

```
$ gcc -o Atividade04 main.c queue.c -g
```

Figura 2.23: comando para rodar o *debugger* no terminal

Nesse exemplo "Atividade04" é o **nome do executável** , "main.c" e "queue.c" são os **arquivos** que vão ser compilados. O "-g" é a **flag do gdb** .

A partir disso é só "debugar" normalmente como já foi ensinado!

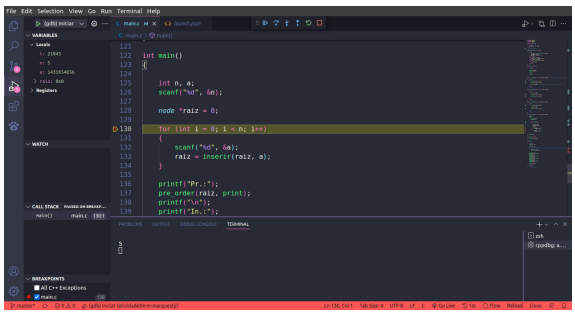


Figura 2.24: *debugger* rodando a partir do terminal

2.6 Compilando com o MakeFile

Para ínicio de conversa o **GCC** se trata de um componente principal do **GNU *toolchain***, que é uma "cadeia de ferramentas" para desenvolvimento e depuração. Basicamente os arquivos de código fonte são transformados arquivo-objeto e são ligados e transformados em binários.

Com isso em mente existe a ferramenta **make** do **GNU *toolchain*** que usa como alvo um arquivo chamado *makefile*, que detém um conjunto de diretivas de instalação, remoção e desinstalação de arquivos, definidas pelo usuário para compilação dentro do próprio arquivo. Assim o *makefile* pode conter variáveis, regras (ou "*targets*") e comentários, para executar a compilação de forma rápida, sem a necessidade digitar todo o comando no terminal.

Porém para utilizar o **make** é necessário instalá-lo no Sistema Operacional, usando os seguintes comandos:

```
$ sudo apt-get update
$ sudo apt-get install make
```

Figura 2.25: atualizando o sistema e instalando o make

A ideia é mostrar uma forma simples de utilização do makefile. Primeiro se faz necessário criar um arquivo makefile na pasta onde se encontra o projeto utilizando o seguinte comando no terminal(ou pode-se criar pela interface gráfica também):

```
$ touch makefile
```

Figura 2.26: criando o arquivo do makefile

A partir disso é necessário inserir os comandos para criação de arquivos no makefile como esses abaixo, como código fonte e cabeçalho vão ser utilizados os arquivos **main.c, queue.c e queue.h**:

```
all: nomedoexecutavel #pré-requisito

nomedoexecutavel: main.o queue.o #pré-requisito
  gcc -o nomedoexecutavel main.o queue.o -g

arquivo-objeto1: main.c queue.h #pré-requisito
  gcc -c -g main.c -o main.o

arquivo-objeto2: queue.c queue.h #pré-requisito
  gcc -c -g queue.c -o queue.o

clean:
  rm *.o Atividade*
```

Figura 2.27: estrutura do arquivo makefile

- **all** : é a primeira target que o comando make procura no arquivo makefile, e detém o nome do executável final

- **nomedoexecutavel** : executável a ser criado

- **arquivo-objeto** : é o arquivo que vai ser criado na pasta onde esta o código fonte, após a compilação

- **clean** : se trata de uma *target* para apagar arquivos de ligação (ou *linker*) da pasta, executando um comando *rm* no terminal e apagando arquivos com a extensão ".o" e que começam com a palavra "Atividade."

- **pré-requisito** : as palavras em laranja são as pré-requisito para a criação do determinado arquivo após a compilação

Com as explicações das *targets* anteriores é necessário explicar o funcionamento após a execução do comando *make* no terminal. Bom a primeira *target* a ser executada é a **all** , como não detém a receita para gerar o executável vai para **nomedoexecutavel** que é a próxima *target*, mas ela tem dois pré-requisitos **main.o** e o **queue.o** , então vai para a *target* **arquivo-objeto1** , executa o "gcc -c -g main.c -o main.o", e produz o main.o, após isso ainda há a queue.o como pré-requisito da *target* **nomedoexecutavel** , logo é executado o comando "gcc -c -g queue.c -o queue.o", da *target* **arquivo-objeto2** com os arquivos-objeto prontos, o make volta para a *target* **nomedoexecutavel** e executa sua receita que é o comando "gcc -o Atividade04 main.o queue.o -g".

A partir disso o make invoca o *linker* do gcc, e utilizando a *flag* "-o nomedoexecutavel" define o arquivo binário de saída chamado "nomedoexecutavel" e indica para o *linker* os dois arquivos-objeto "main.o" e "queue.o" então adiciona a flag -g do gdb para depuração. E para finalizar com o binário "nomedoexecutavel" pronto, é cumprido o pré-requisito da *target* **all** .

Para executar todos os comandos basta digitar **make** no terminal, que já vai acionar a *target* **all** . Porém você também pode acionar uma *target* específica como o *clean* por exemplo, da seguinte forma:

```
$ make clean
```

Figura 2.28: comando para target específico, nesse caso com o clean

Após isso basta compilar o executável normalmente:

```
$ ./nomedoexecutavel
```

Figura 2.29: rodando o executável

E assim pode ser executado o passo a passo da depuração na parte "Como depurar?" do artigo.

2.7 OBS: Configurando e instalando Debugger da linguagem C no Windows

Para quem utiliza Windows precisará instalar o compilador GCC e junto o GDB, segue o link do minGW que é um pacote com essas ferramentas do GNU portadas para Windows.

OBS: Se você já possuir o gcc, execute o comando gdb no terminal e verifique se ele também está instalado, se já possuir pule a instalação.

Link de download do Mingw: https://sourceforge.net/projects/mingw/

Instalação:

Irá ser instalado um gerenciador de pacotes, abra-o e marque essas opções:

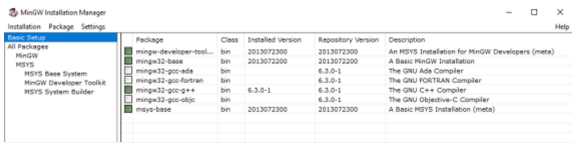


Figura 2.30: opções do Mingw

Após, clique no menu e em *installation* e selecione "*Apply Changes*":

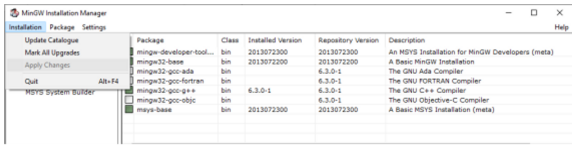


Figura 2.31: Botão Apply Changes

Agora será necessário adicionar os comandos nas variáveis de ambiente para poderem ser executados no terminal. Abra o painel de controle e procure por sistema ou clique com botão direito em "meu computador" e entre em propriedades. Já aberto selecione "configurações do sistema", irá abrir uma nova aba chamada "propriedades do sistema", nela clique em variáveis de ambiente:

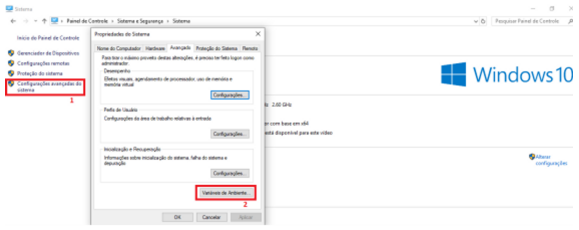


Figura 2.32: Localizando as variáveis de ambiente

Na primeira tabela selecione a linha com nome ***Path*** e depois clique em editar:

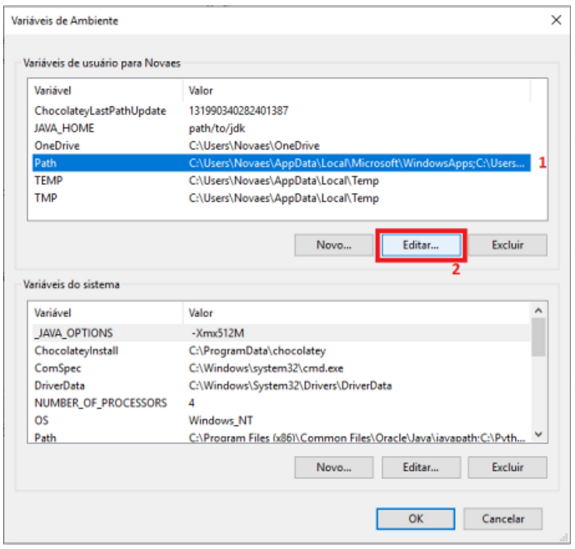


Figura 2.33: Editando as variáveis de ambiente

Uma nova aba irá aparecer, nela clique em Novo e adicione o caminho de instalação do MinGW com a pasta bin. (verifique primeiro onde você instalou). Após confirme clicando OK em todas abas que foram abertas.

Para testar se está tudo ok, é só executar os comandos no terminal. Caso os comandos não sejam encontrados, reinicie o computador se persistir confira se os passos foram feitos corretamente.

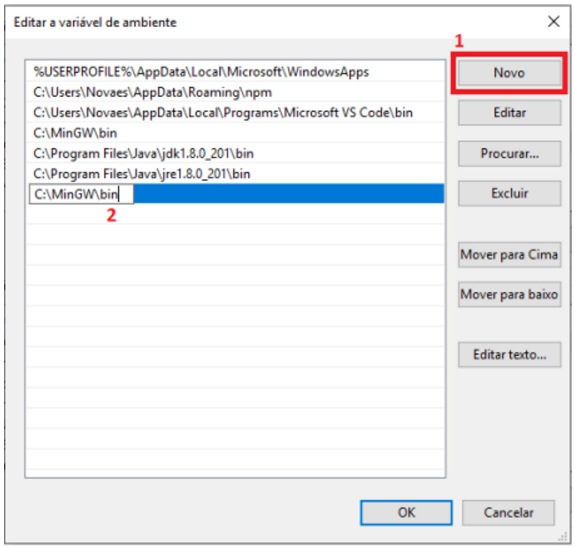


Figura 2.34: Criando novas variáveis de ambiente

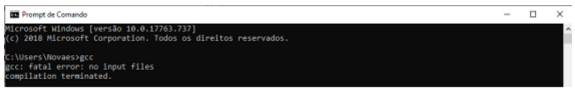


Figura 2.35: Conferindo acesso ao gcc pelo terminal windows

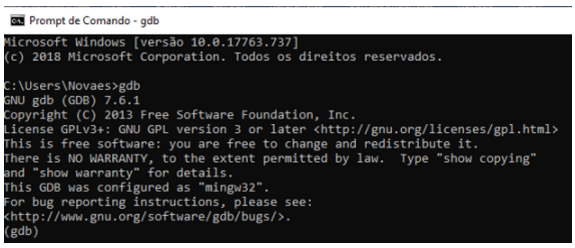


Figura 2.36: Conferindo acesso ao gdb pelo terminal windows

Após a instalação no Windows, é só voltar para o começo do tutorial e seguir a partir do tópico “Pré-requisitos” na parte “Instale a extensão “C/C++”” ou vá direto clicando aqui!!

2.8 Bibliografia

[1] Ubuntu. DEVELOPERS. Como instalar gdb no ubuntu. como instalar, c2022. URL: https://howtoinstall.co/pt/gdb/.

[2] Wellington. DOMICIANO. Instalando o gcc no windows com mingw. URL: https://wldomiciano.com/instalando-gcc-no-windows-com-mingw/.

[3] Fabio. DOS REIS. Como instalar gcc e pacotes de desenvolvimento no linux debian 10. bóson treinamentos em ciência e tecnologia, 2020. URL: http://www.bosontreinamentos.com.br/linux/como-instalar-gcc-e-pacotes-de-desenvolvimento-no-linux-debian-10/.

[4] Marcos. OLIVEIRA. Como criar um makefile. embarcados, 2019. URL: https://terminalroot.com.br/2019/12/como-criar-um-makefile.html.

[5] PROGBASE. How to debug c code with gdb in vscode (linux). youtube, 28 de nov. de 2017. URL: https://youtu.be/aWIs6Kv1MvE.

[6] Lincoln. UEHARA. Introdução ao makefile. embarcados, 2019. URL: https://embarcados.com.br/introducao-ao-makefile/.

[7] Pedro. VASCONCELOS. Introdução ao gdb. departamento de ciÊncia de computadores, 2019. URL: https://www.dcc.fc.up.pt/~pbv/aulas/progimp/teoricas/teorica12.html/.

Capítulo 3

Netcode em Jogos de Luta: Introdução e Exploits

Leonardo Valerio Morales
Vitor Kenzo Fukuhara Pellegatti

OBS: Para ver o artigo na íntegra, com imagens maiores, clique **aqui.**

Resumo

Este artigo pretende ser uma espécie de introdução ao conceito de netcode, em específico, os dois estilos mais utilizados especialmente para jogos de luta, Delay-based e Rollback netcode. Além de fazer uma breve análise histórica da situação dos jogos de luta que nos levaram a esses dois estilos de netcode, adentramos um pouco nas minúcias de ambos estilos de netcode, no funcionamento de jogos de luta em geral.

Palavras-chave: *Rollback Netcode*; Jogos de Luta; FGC; *Delay-Based Netcode*; *Netcode Exploits*.

3.1 Introdução

O artigo tem como principal objetivo a discussão de diferentes métodos para a implementação do *netcode* de jogos em geral, mais centralizada em jogos de luta. Nas primeiras seções, faremos uma breve análise histórica do surgimento do gênero de jogos de luta até como ele se apresenta hoje em dia. Seguindo isso, explicaremos um pouco de como funciona a lógica por trás desses jogos e os possíveis problemas que podem ser gerados. Logo em seguida, iremos tentar introduzir alguns dos aspectos importantes para discussão quando se trata da rede. Não só isso como traremos o enfoque ao *Netcode* como um aspecto de extrema importância para o possível sucesso de jogos de luta no futuro. Depois disso, explicaremos o funcionamento essencial dos dois principais sistemas de *Netcode* em jogos de luta, o sistema *Delay-based* e o *Rollback*. Posteriormente vamos ainda explicar como esses dois sistemas podem ser misturados para se auxiliarem. Por fim, falaremos um pouco sobre o **GGPO**, ferramenta que pode ser usada para incluir o *Rollback* em jogos assim como vamos falar um pouco sobre os possíveis *exploits* que podem ser feitos nos *Netcodes*.

3.2 Um pouco de história

Antes de qualquer coisa, vamos tentar entender como esse estilo de jogos e a comunidade construída em volta deles chegou no ponto que está nos dias de hoje. Vamos tentar começar com o que é a essência de jogos de luta, dar um breve histórico deles até os dias de hoje e então passar pelas características principais que transformam esse estilo de jogo interessante e divertido tanto de assistir quanto de jogar e o que ferir essas características pode acarretar.

O que são jogos de Luta

Desde o seu nascimento nos anos 90 até os mais variados estilos que se tem disponível hoje no mercado, jogos de luta em sua essência colocam duas pessoas e seus conhecimentos e habilidades relacionadas a dado jogo uma contra a outra, para definir quem é o vitorioso. Cada jogador é representado por um ou mais personagens de sua escolha, que possuem um arcabouço vasto e interessante de movimentos e táticas que podem ser utilizadas por aquele que está no controle para conseguir ganhar de seu oponente.

Em sua mais pura essência, jogos de luta testam as capacidades mecânica de um jogador, de execução de movimentos, e muitas vezes uma sequência desses movimentos(também conhecidos como combos), assim como a capacidade desse jogador de ler o que está passando na cabeça de seu oponente para então ter uma vantagem sobre ele.

James Chen, narrador e uma figura influente na comunidade de jogos de luta(também conhecida como FGC) menciona que jogos de luta são como "um jogo de xadrez em alta velocidade", o que é válido para retratar as tomadas de decisões em frações de segundo necessárias no estilo, mas temos que lembrar que precisam ser somadas as dificuldades mecânicas que esse jogos possuíam e ainda podem possuir.

O gênero teve seu nascimento no início dos anos 90 e até hoje passou por diversas turbulências que definiram o estilo como o conhecemos hoje.

Breve histórico dos Jogos de luta

Jogos de luta se tornaram populares com o lançamento do **Street Fighter II** (SFII) de 1991. Esse jogo se tornou o *template* para jogos de luta como conhecemos hoje em dia: a possibilidade de escolha de personagens, diferentes habilidades e golpes para cada um dos seus 8 personagens, um total de rounds que precisam ser ganhos para sair vitorioso da partida, um tempo limite para cada *round*, a capacidade de conectar um movimento de seu personagem em outro(o que hoje chamamos de combos), até mesmo a possibilidade de jogar contra outra pessoa só foi concebida com esse jogo.

Esse jogo se tornou um fenômeno e então diversos outros jogos vieram logo em seguida. Várias franquias que ainda lançam jogos até hoje nasceram da popularidade do SFII como **Samurai Showdown** da SNK e o tão conhecido **Mortal Kombat**, mas nenhum chegou a ter o sucesso que o grande bisavô dos jogos de luta teve.

Claro que hoje em dia cada sub-gênero pegou um desses aspectos e modificou um pouco, mas a princípio, essas características se tornaram fundamentais para quase todos os jogos de luta que vemos hoje em dia sendo oferecidos. Essa grande febre por jogos de luta veio ao seu declínio no final da década de 90 e começo dos anos 2000 e durou até aproximadamente 2008. Esse período está diretamente relacionado ao período em que a *Capcom*, produtora da série **Street Fighter**, ficou sem lançar um jogo principal da franquia(a última entrada, **Street Fighter III: 3rd Strike**, havia sido lançada em 1999 e a próxima, **Street Fighter IV** só seria lançada em 2008). Durante o período, esse grande conglomerado de diferentes tipos e estilos de jogos de luta começou a ver um declínio no grande pico de popularidade até então adquirido, não necessariamente por um declínio de qualidade dos jogos que estavam lançando na época, mas por dois fatores principais: a morte dos *arcades* e a especialização dos jogos de luta. O último citado acabava afastando muitos jogadores novos devido a grande quantidade de informações que eram necessárias apenas para começar a entender o básico desses jogos, e, somados ao declínio dos *arcades* e o aumento de consoles caseiros com uma grande variedade de jogos de outros estilos disponíveis, muitas pessoas apenas abandonaram o gênero por algo mais cômodo.

O estilo sobreviveu ainda sim a essa era das trevas, inclusive, um dos momentos mais marcantes da história dos jogos de luta competitiva(até mesmo de jogos competitivos em geral) nasceu durante esse período. O tão conhecido *EVO Moment 37*, também conhecido como o **Daigo**

Parry, uma sequência incrivelmente precisa do jogador **Daigo Umehara** aconteceu em um dos grandes torneios de jogos de luta em 2004 (EVO 2004).

O período de declínio acaba com a quarta entrada da série **Street Fighter**, **Street Fighter IV**, lançada em 2008 que reacendeu a popularidade que o gênero um dia possuiu. Um jogo que remetia bastante o seu ancestral SFII, foi o jogo que se tornou porta de entrada e volta de muitos jogadores que haviam abandonado o gênero durante a época sombria. Não só isso, os grandes torneios se tornaram cada vez maiores devido a grande paixão que a comunidade possuía, assim como a ajuda dos próprios desenvolvedores. Além disso, o advento da internet e *streaming* ajudava a propagar a comunidade e esses jogos para fora do seu nicho.

Atualmente, os jogos de luta vem crescendo em quantidade e qualidade. Devido a grande base já existente de jogadores dos mais variados subgêneros de jogos de luta, é difícil que uma pessoa que acaba entrando no mundo dos jogos não se depare com um jogo do estilo.

O que restou para os jogos de luta

Aqui precisamos primeiro relembrar que é intrínseco a esse estilo de jogo a competição entre um indivíduo contra outro, e esse mesmo fato torna esses jogos únicos nesse quesito. Jogos populares dos dias de hoje ou até mesmo antigamente, colocam um jogador sozinho contra certos desafios, como por exemplo **Hollow Knight**, **Outer Wilds**, **Celeste**, etc. Ou até mesmo um time de pessoas contra outro time de pessoas, como jogos *multiplayers* online, por exemplo **League of Legends**, **Counter-Strike Global Offensive** ou até mesmo **Rocket League**.

Essa essência competitiva única do estilo acabou sendo o estopim para que várias pessoas em suas cenas locais se juntassem e começassem a tentar melhorar suas próprias habilidades, seja por reconhecimento e fama na cena, ou até mesmo porque você ficaria mais tempo jogando caso fosse bom nos tempos dos *arcades*. E essas comunidades acabaram apenas se estabelecendo e crescendo conforme o tempo passou.

Manter uma integridade competitiva para esses jogos é nada mais do que essencial para que eles tenham o seu potencial máximo de serem interessantes e divertidos, que propulsiona a criação de uma comunidade em volta desses jogos. Essa integridade competitiva não precisa ser muito questionada quando consideramos esses jogos localmente, porém, temos que tomar cuidado quando estamos conectando jogadores pela internet.

É fato que, enquanto outros jogos competitivos da época conseguiram se adaptar bem ao advento da internet, não foi bem o caso para o gênero de jogos de luta. **Tony Cannon** menciona que haviam duas maneiras de jogar jogos de luta com outras pessoas durante a Idade das trevas. Uma delas era *"couch play"*, que seria juntar pessoas localmente para jogar, que era cada vez mais difícil de ocorrer dado o declínio dos *arcades* da época. A outra maneira seria jogar online, porém os jogos possuíam muita latência, diretamente proporcional ao tempo que demorava para se transmitir os pacotes pela rede.

Essa latência mencionada era um fator extremamente negativo que quebrava completamente a integridade competitiva do jogo, retirando aquilo que tornava esse jogos interessantes. Essa solução de rede, que introduziu latência ao conectar dois jogadores online, especificamente para jogos de luta, fazia com que os jogos perdessem o seu valor competitivo, que é o principal aspecto do estilo.

3.3 Como funcionam os jogos de luta?

Jogos de luta em sua essência são jogos na sua grande maioria determinísticos, ou seja, dado um estado e um conjunto de entradas, o próximo estado será único. Isso faz com que dada uma situação e um conjunto de comandos dos jogadores, o resultado será sempre o mesmo.

O que é determinismo e porque é utilizado?

Dentro do mundo da computação, nos deparamos com diversos algoritmos determinísticos, na verdade, qualquer algoritmo é determinístico, visto que até elementos que fazem uso de RNG (*Random Number Generation*) para introduzir o não-determinismo em algoritmos se baseiam em geradores de números pseudo-aleatórios que podem ser até previstos. Mesmo com a inexistência de não-determinismo puro em qualquer algoritmo e por extensão qualquer jogo, o uso de RNG é considerado não-determinismo pois em um cenário real, nenhum ser humano é capaz de prever o resultado de uma ação governada por RNG.

Visto isso, é imperativo para Jogos de Luta, que tem como pilar o confronto direto de dois jogadores sem a interferência de nenhum outro elemento e que visam a simetria no conflito, o uso de uma lógica determinística.

Por meio da lógica determinística, os jogadores podem desenvolver memória muscular e *"timing"* ou seja, passam a aplicar o conhecimento prévio adquirido para melhorar e se tornarem mais consistentes. De fato, o que atrai jogadores a este gênero é justamente este aspecto! É possível melhorar com a prática desenvolvendo memória muscular e o chamado *"game sense"* ou senso de jogo. O senso de jogo é o conhecimento de que dado um estado e um comando do jogador é garantido um outro estado único. Exemplificando, eu sei que se meu oponente estiver no ar e se eu aplicar uma combinação de golpes com ritmo correto, o oponente irá sofrer uma quantidade específica de dano e será jogado para longe.

Os dois conceitos de memória muscular e senso de jogo transformam um jogo de luta em uma batalha de reflexo, previsão e memória muscular que é extremamente prazerosa de jogar.

Como aplicar a lógica determinística a um Jogo?

Em um algoritmo comum, a aplicação do determinismo é trivial, na verdade, caso o algoritmo dependa apenas de uma entrada fornecida, a saída será sempre a mesma naturalmente, sem necessidade de nenhuma outra modificação.

Mas quando se trata de um jogo as entradas não são fornecidas de maneira comum, e portanto, a execução desse algoritmo deve ser cuidadosamente implementada para garantir o determinismo aparente para ambos os jogadores.

Uso do *Game Loop*

O *Game Loop* é uma estrutura de execução que visa padronizar as ações do Jogo em diversas etapas. O *Game Loop* não muda durante a execução do jogo, sendo assim, nenhuma ação deixa de ser executada na ordem. O *Game Loop* comum de um jogo de luta consiste nas seguintes etapas:

1. Pegar Amostra dos controles
    a) Ambos os jogadores constantemente geram amostras de comandos em seus controles, até a posição neutra (Nenhum botão pressionado) é considerado um comando a ser amostrado
2. Computar o próximo estado utilizando as amostras dos controles e a lógica do Jogo
    a) A lógica do jogo dita quais ações serão tomadas por cada personagem dada a situação do mesmo e o comando de seu jogador. Por exemplo, caso o personagem esteja em um estado neutro e o comando de pular seja amostrado do seu jogador, o personagem pode entrar em estado de pulo
3. Renderizar um *frame* do Jogo
    a) Com a informação do novo estado do jogo, um novo *frame* (ou quadro) pode ser renderizado e enviado para a saída de vídeo.
4. Espera

a) Pode haver uma espera entre o fim e o início do próximo *Loop*, isso acontece para padronizar o ritmo de exibição do jogo caso desejado. Lembrando que o ritmo de execução e resposta do jogo é algo imprescindível para a experiência dos jogadores. Ou seja, esse passo pode ocorrer ou não mas é sempre desejado que seja executado no mesmo ritmo definido.

Dada a aplicação correta do *Game Loop*, a execução do jogo se torna completamente determinística. Onde a saída determinística em si é o quadro a ser renderizado que representa o estado do jogo no momento.

A transição para um sistema distribuído e como isso causa problemas

Quando a execução do jogo deixa de ser isolada e passa a ser distribuída, ou seja, passa a ocorrer em mais de um sistema, são introduzidos alguns problemas para manter o determinismo.

O determinismo agora passa a ser desejado de modo que em ambos os sistemas, o jogo estará apresentando o mesmo estado e mais importante ainda, ao mesmo tempo.

Problemas lógicos introduzidos por Jogos de Luta Online

Lembrando do *game loop*, o jogo primeiro colhe amostras dos controles, executa uma computação e então exibe o próximo estado. Quando movidos para um ambiente distribuído, como ocorre a coleta das amostras dos controles? Quando isolado isso era trivial, visto que as amostras sempre eram coletadas ao mesmo tempo, dada a mesma distância dos controles às entradas e ao mesmo tempo de *clock* dos aparelhos. A mudança de ambiente traz uma série de problemas lógicos.

Primeiramente, não é mais possível coletar os mesmos comandos ao mesmo tempo. Enquanto o Jogador 1 pressionava um botão e o Jogador 2 outro, ambas as instâncias do Jogo não sabem das amostras do outro jogador e portanto é impossível computar o próximo estado imediatamente.

Além disso, como o jogo será executado em mais de um sistema, todas as diferenças de hardware introduzem diferença no ritmo de execução e portanto no resultado de cada *loop*.

Finalmente, no modo isolado não havia confusão sobre a presença de dois jogadores, mas agora com a distribuição, como um dos processos irá saber caso o outro pare de funcionar?

Todos esses problemas na verdade fazem parte de uma área da ciência da computação chamada Algoritmos Distribuídos. E na verdade o maior e único problema de estudo desta área é: Como fazemos dois processos que executam o mesmo algoritmo decidir a mesma coisa dada as circunstâncias inconsistentes da conexão entre os mesmos?

Na sessão seguinte essa área e as soluções encontradas na mesma serão discutidos de maneira mais profunda.

3.4 A rede e suas inconsistências

Por mais que tenhamos aprendido na disciplina da área de Redes que por natureza a comunicação é inconsistente e é preciso criar medidas preventivas para se obter confiabilidade, a essência deste problema existe a muito mais tempo que as redes em si.

Inicialmente na história da computação, computadores eram grandes e quase nunca eram conectados uns com os outros, contudo, após certo progresso se tornou mais viável conectar dois computadores, mas não para fins de comunicação e sim para meios de acelerar computações.

Surgiu então um estudo de como projetar algoritmos para serem executados paralelamente em vários computadores. Algoritmos Distribuídos estuda justamente isso, e na verdade, levava em conta as inconsistências de conexões e execuções

antes da rede se tornar global.

O problema geral de algoritmos distribuídos

Para recapitular, Jogos de Luta utilizam um meio de conexão P2P, ou seja, a conexão entre dois jogadores é realizada estritamente entre os meios, sem utilização de um servidor mediador. Nestas conexões *P2P* são trocadas informações para permitir que ambas instâncias do jogo que estão rodando em cada computador produzam o mesmo resultado em cada *frame* renderizado.

Este problema se encaixa no problema geral dos algoritmos distribuídos: Como fazer todas as instâncias de um algoritmo sempre decidirem a mesma coisa? Pode parecer simples, mas essa pergunta é a essência dessa área de estudo e não é simples de responder.

Para responder essa pergunta, o ambiente foi abstraído em camadas cada vez mais próximas da realidade:

- Inicialmente a conexão entre todos os computadores é perfeita, nenhuma informação é perdida e nenhum processo falha.
- Processos começam a falhar, mas avisam que falham para todos os outros.
- A conexão passa a não ser perfeita, o que significa que todas as informações são transmitidas, porém com um certo *delay*, assim não podemos mais confiar nos avisos de falha dos outros computadores.
- A conexão agora é falha, informações podem se perder no meio do caminho sem nenhum dos computadores saber, os processos que falham não avisam mais os outros.

A última instância de abstração é onde nos encontramos com Jogos de Luta, por utilizar UDP, os pacotes que transmitem os comandos podem simplesmente se perder no meio do enlace, sem tempo máximo de transmissão pois pode variar.

O formato geral das soluções para o problema geral

Para resolver o problema geral de algoritmos distribuídos, é escolhido um conjunto de computadores ou um único computador que age como o líder do grupo, este líder possui a autoridade de decidir para o resto e o resto apenas decide quando o líder decide. Contudo, essa arquitetura de solução não é uma boa opção para Jogos de Luta, ou melhor, uma maioria de gêneros. Como jogos de luta são construídos na base de semelhança e simetria, ter apenas um dos jogadores como líder iria introduzir uma falta de simetria. Imagine que se os comandos do Jogador não cheguem ao Jogador Líder, o Jogador Líder pode criar uma tremenda vantagem efetuando combos e no final, ainda tem a autoridade de falar ao outro jogador que tudo aquilo aconteceu mesmo.

Para Jogos de Luta *P2P*, é obrigatório que ambas as instâncias sejam uma instância Líder, o que nos traz a dificuldade de diferença de decisão novamente. Quando discutirmos as soluções de *netcode* utilizadas nos jogos, iremos desenhar um paralelo entre as soluções e os conceitos de Algoritmos Distribuídos.

Redes e suas inconsistências no mundo não-abstrato

Já foram discutidos os problemas de redes em um nível abstrato, mas como esses problemas se traduzem para o mundo real?

A transmissão de pacotes na redes está sujeita a diversos bloqueios, seja quando um pacote encontra uma fila congestionada em um aparelho de rede ou caso um meio físico sofra interferência do ambiente, pacotes são perdidos e a latência sobe.

Para remediar o problema da perda de pacotes, foi criado o protocolo *TCP*, este protocolo visa garantir o envio de um pacote introduzindo pacotes pares mandados pelo recipiente. Contudo, essa nova característica introduz um tamanho maior de pacote e um congestionamento maior de

rede com os pacotes pares, por isso ainda existe o protocolo padrão *UDP*. O *UDP* não possui confiabilidade, logo não é garantido o envio de um pacote, mas por outro lado, por ser mais leve e congestionar menos a rede, possui uma latência menor.

Quando se trata de jogos P2P em sua grande maioria, a perda de pacotes é esperada e remedia pelo próprio código de rede daquele jogo, portanto, é possível utilizar o *UDP* para fornecer aos jogadores uma menor latência geral. Novamente, por utilizar *UDP*, os jogos de luta se submetem a perda de pacotes, o que é desastroso para o jogo, mas menos desastroso que utilizar *TCP* e introduzir latência que limitaria o ritmo do *game loop*.

Além da perda de pacotes, pode ocorrer também o atraso de pacotes, e picos de latência. Caso uma série de pacotes tomem uma rota específica que em algum ponto fica congestionada, o tempo de viagem desses pacotes será anormalmente maior do que os outros, introduzindo um pico de latência que depois se normaliza.

Todos estes problemas são resolvidos utilizando algumas técnicas que abrangem conceitos puros de redes e também algoritmos distribuídos, e serão discutidas especificamente para cada abordagem.

3.5 *Netcode* como aspecto essencial

Agora que já sabemos como a rede possui diversas inconsistências e temos uma ideia geral de como jogos de luta funcionam em seus pormenores, podemos partir para a ideia principal do artigo que é discutir o *netcode* e como na atualidade é uma solução fundamental para a sobrevivência desse estilo de jogo.

Uma mudança em como o jogo é jogado

Mesmo agora com certa estabilidade, o período de pandemia somadas as dores de cabeça que eram jogar jogos de luta mesmo depois do período das trevas fizeram a comunidade clamar por jogos que pudessem ser jogados online como se estivessem sendo jogados localmente. **Gerald** do canal **A-core Gaming** menciona em um de seus vídeos "Nem todo mundo possui uma excelente conexão com a internet, assim como nem todo mundo mora na mesma região. Se todos conseguíssemos jogar localmente, a latência não seria um problema, mas vivemos num mundo onde *arcades* morrem e vírus vivem, então jogar online importa bastante." O comentário está apontando o quão problemático a latência gerada pela rede e a implementação que era mais comum antigamente de *netcode*, *delay-based*, era e continua sendo.

Para demonstrar o quanto essa latência atrapalha, Gerald no mesmo vídeo compara a precisão necessária para fazer comandos em jogos de luta com a precisão necessária para se tocar uma música no violão, por exemplo. Imagine o quão frustrante seria aprender uma música, se o som de uma corda tocada ou tecla pressionada demorasse segundos inteiros para sair de seu instrumento. Voltando da analogia, podemos entender o problema gerado por essa latência quando dois jogadores eram conectados pela internet, acabando com a responsividade do jogo e o tornando praticamente não jogável.

Netcode: definição e objetivos

No glossário dos jogos de luta, o termo *netcode* é definido como: "Descreve exatamente o método que um jogo de luta usa para implementar o jogo online. Existem dois métodos principais usados para jogos de luta, delay-based *netcode* e *rollback netcode*. Resumidamente, *delay-based* é um pouco mais fácil de implementar, mas não funciona bem na prática, enquanto o *rollback* é mais trabalhoso para os desenvolvedores, mas permite um jogo online que se assemelha muito ao jogo local."

Podemos dizer então, com o que foi dito anteriormente pelo Gerald, que o principal objetivo do *netcode* de jogos de luta é mascarar com perfeição a latência que existe quando se joga através da rede contra outro jogador.

Vamos entrar nas próximas seções mais a fundo tanto no *delay-based* quanto no *rollback netcode*, mas o fato é que com essa definição em mãos podemos perceber que existe uma versão que era mais utilizada como padrão devido sua simplicidade e contexto, uma vez que a maioria das pessoas na época de ouro dos jogos de luta não se interessava tanto em jogar esse jogos online até porque não era uma necessidade.

Os tempos mudaram, e agora pode-se argumentar que, até mesmo para que o gênero sobreviva, é de extrema importância que esses jogos e novos lançamentos de franquias icônicas de jogos luta suportem a possibilidade de dois jogadores se conectarem através da rede e jogarem o jogo sem qualquer perda de responsividade. Existe a necessidade de trocar o foco para o jogo online, porque se tornou a maneira mais comum das pessoas do mundo inteiro jogarem esse jogos. Tony Cannon em uma palestra sua dada em 2017 na EVO daquele ano mostra uma postagem feita por ele mesmo em um fórum de discussões de 2006 onde ele discute a possibilidade de esconder a latência da internet, não negá-la: "Mesmo com o melhor *netcode*, ainda você ainda teria que lidar com latências reais da internet. Não podemos negar o fato de que pode levar 50 ms ou mais para um pacote se mover de um NIC (*Network Interface Controller*, as placas que possuem endereços MAC) físico para outro pela internet, não importa o quão bom são *netcode* é.(...) Aceite que você não consegue transmitir um pacote em menos de 1 *frame* e foque ao invés disso em criar a ilusão de um jogo com zero *frames* de latência usando truques de mágica para esconder o tempo de transmissão de pacotes. Se alguém resolver esse problema, será com bom *netcode* para minimizar a latência e um molho especial para esconder a latência que é inerente na transmissão de pacotes pela internet."

Adam Heart, hoje lead-designer do *Rumbleverse*, em um tweet seu de 2018 menciona que "a medida para um bom *netcode* não é em como ele lida com boas conexões, mas como esconde efeitos de conexões ruins." É crucial entender que qualquer um dos dois métodos, dado condições ideais de conexão não devem apresentar problemas. Precisamos verificar como o *netcode* lida com situações que fogem do ideal para verificar se ele funciona bem ou não.

Adam Heart expande na tolerância que pode ser observada ao se jogar uma partida online assim como um pouco da diferença entre usar *delay-based* e *rollback netcode*: "(...)A coisa que a maioria das pessoas não pensa sobre quando se tem um bom *netcode* é o sistema de *matchmaking*. Não é só um fator de, eu consigo achar uma partida que tem um ping baixo o suficiente ou perto do meu *rank*, certo? Quando você tem realmente um bom *netcode* que faz *rollback*, você está expandindo a busca. Provavelmente o *ping* máximo tolerável para a maioria das pessoas no delay-based *netcode* está entre 60 e 80. Qualquer coisa acima disso é lixo in-jogável. Caso haja qualquer perda de pacotes, é lixo in-jogável. Com *rollback*, você expande isso, agora você pode jogar com uma pessoa à 150 de *ping*. Parece offline. Você pode jogar pessoas à 200 de *ping*, com alguns artefatos visuais que são estranhos, mas parece offline. De repente minha esfera de matchmaking não são os dois estados próximos do meu, é na realidade grande parte do globo.(...) É muito difícil encontrar qualquer coisa que vá além de 150(*ping*). Você encontra coisas(conexões) que vão além de 200, se você realmente estiver do outro lado do mundo, mas por exemplo, Orlando até Londres provavelmente não está acima de 150."

O *rollback* é um sistema que apresenta uma robustez necessária para que esses jogos tragam a diversão que eles podem trazer em um ambiente online, algo que o *delay-based netcode* sozinho não consegue fazer. Nas próximas seções vamos primeiro expandir e entender como o *delay-based netcode* funciona para que depois possamos explicar e discutir o rollback, assim como a utilização dos dois métodos em conjunto, que parece ser uma implementação ideal para o *netcode* de um jogo de luta.

Por fim vamos lembrar que, ambos são métodos de *netcode* que podem e são implementados de maneiras diferentes pelos diversos desenvolvedores de jogos de luta. Porém pode-se dizer que o método de *rollback* é mais robusto e permite que pessoas que possuem latências maiores, inerentes a distância de conexão, possam ser pareados em

partidas onlines sem sofrer as consequências da latência.

Agora vamos entender esse dois métodos começando pelo *delay-based netcode* e depois seguindo para o *rollback* e discutir as vantagens e desvantagens de ambos os métodos.

3.6 *Delay Based Netcode*

Uma das soluções para as inconsistências de redes que os jogos de luta sofrem, é o código de net baseado em atrasos, ou, *Delay Based Netcode*. Foi usado por muito tempo, com início justamente na época de transição de *arcades* presenciais para jogos de luta online. No geral, é uma solução preguiçosa, em que problemas de conexão são tratados de maneira bruta primeiramente e depois uma métrica gulosa é utilizada para evitar mais problemas.

A resposta lógica ao problema geral de Algoritmos distribuídos

Lembrando que o problema geral que se apresenta é um problema de autoridade de decisão. Como ambas as instâncias do jogo têm que ter autoridade para decidir o próximo estado do jogo, é imprescindível o recebimento dos inputs de ambos os jogadores ao mesmo tempo.

Em *Delay Based Netcode*, inicialmente, todo *input* de um jogador em cada instância do jogo sofre um delay, ou atraso, esse atraso serve para que o pacote que carrega o *input* tenha tempo de chegar no computador do outro jogador, quando ambos os computadores recebem o *input*, ambos decidem o próximo estado, assim chegando na mesma conclusão.

Surge um grande problema com essa abordagem durante um pico de latência, ou seja, uma perda de pacote. Se um pacote se perder, ele não será recebido durante a janela do atraso do *input* do jogador, e portanto o jogo não terá a informação necessária para decidir o próximo *frame*. O *Delay Based Netcoding* nesse caso opta por congelar o jogo para os dois jogadores, esperar um reestabelecimento da rede e pede novamente o *input* perdido, só então quando o recebe decide o próximo *frame*.

O fato do jogo ser congelado é extremamente indesejável para jogos de luta, onde a fluidez e rapidez de resposta do jogo é um fato decisivo na performance dos jogadores. Por isso, para evitar outras perdas, o *Delay Based Netcode* decide aumentar o atraso de *input* em ambas as instâncias do jogo, aumentando assim a janela de atraso para a chegada de um pacote.

Vantagens e desvantagens

Como demonstrado, a lógica por trás da solução para a inconsistência de rede é fácil de implementar e resolve o problema de decisão. Esse aspecto se torna a maior vantagem desse tipo de *Netcode* e por isso foi utilizado por vários jogos de luta online, porém suas desvantagens foram responsáveis por uma queda na popularidade desse gênero.

Existem três desvantagens com essa solução:

1. O congelamento do *game loop* é nocivo para a jogabilidade. A última coisa que um jogador espera é um congelamento inesperado sem nenhum aviso, isso fere o aspecto de precisão do gênero.

2. A variabilidade no atraso de *input* introduz elementos aleatórios ao jogo. Um atraso constante não é problema, já que o jogador pode se acostumar com o mesmo, porém um atraso que muda durante uma partida é muito menos adaptável. O ritmo de um combo agora é outro e em um tempo de segundos pode mudar para outro ritmo, pois o atraso sofreu uma nova mudança.

3. A intolerância a melhora da rede. O *Delay Based Netcode* sempre assume o pior caso, isto é, se a rede em algum momento chegou a um estado ruim, assume que este estado será permanente, e só se adapta caso a rede fique pior ainda. Isso causa com que um pico de latência atômico se propague durante toda a

partida, o atraso de *input* raramente diminui, portanto uma partida inteira pode sofrer por um evento atômico.

Em geral, o aspecto vantajoso dessa solução é a sua facilidade de implementação e performance, mas deixa em muito a desejar no aspecto da jogabilidade. O uso de *Delay Based Netcode* é fruto de frustração em jogadores do gênero até hoje, e pode fazer a diferença na preferência entre um jogo e outro que utiliza um *Netcode* mais amigável a jogabilidade.

3.7 *Rollback Netcode*

Nesta seção vamos discutir o funcionamento do *rollback netcode*. Intrinsecamente não é uma ideia muito difícil de entender, e por mais que existem certas dificuldades ao implementar o método, pode-se dizer que esse estilo de *netcode* é mais sólido que seu antecessor, o *delay-based*, assim como mais consistente. É uma maneira de fazer jogos de luta no geral funcionarem em um ambiente online sem nenhuma latência de *input*.

Relembrando como os jogos funcionam

Para podermos entender como o *rollback* é implementado precisamos primeiro entender como os jogos que esse sistema trabalha em cima funcionam. Explicamos como isso funciona em sessões anteriores mas sempre vale a pena relembrar.

Basicamente, 60 vezes a cada segundo o jogo irá ler o controle de seus jogadores e então capturar o que quer que o jogador esteja fazendo. Depois disso o jogo simula um estado de jogo dado o *input* recebido e então gera esse estado que geralmente é chamado de *frame*. Depois que ele é gerado ele é renderizado na tela e então o *loop* volta ao começo.

Claro que existem várias checagens que são feitas na parte de atualização do estado do jogo. Pode-se por exemplo, caso o jogo esteja sendo jogado pela internet, fazer constatações em relação ao estado da conexão, ou caso o jogo esteja sendo jogado contra um *CPU*, pode-se rodar a IA

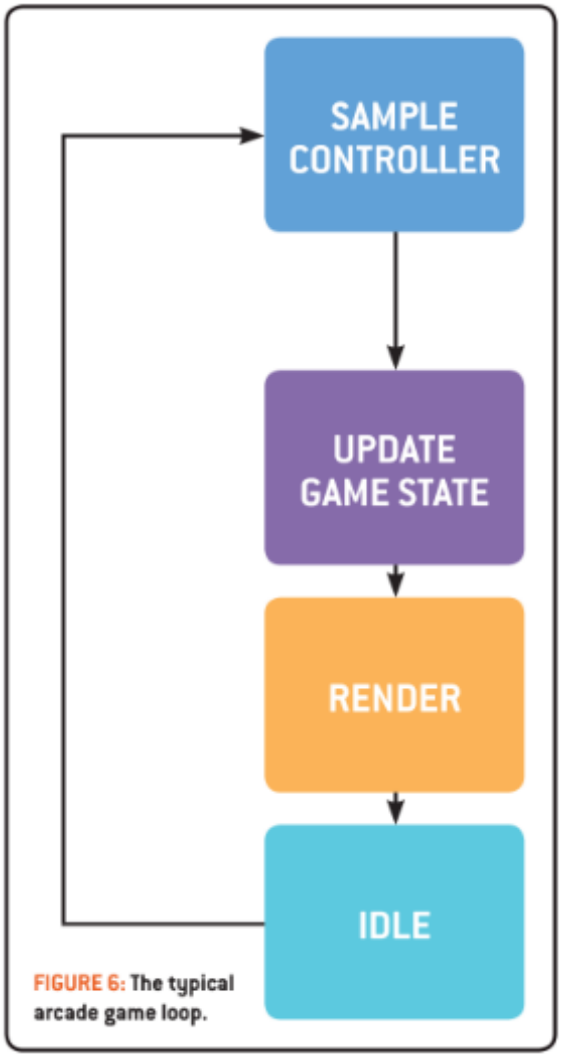


Figura 3.1: *Game loop* de jogos de luta

que controla o oponente. Além disso, a animação dos movimentos feitos pelos personagens e a checagem se algum dos jogadores está sendo acertado acontecem nesse estado que é feito em cada *loop*. Como é feito a cada *loop* e traz informações necessárias para que o jogo simula os próximos estados corretamente, tudo aqui precisa ser consistente e preciso.

A verdadeira dificuldade se encontra quando ambos os jogadores não estão no mesmo sofá apertando botões em seus controles.

O funcionamento do *loop* na rede

Agora que sabemos do *loop* básico, precisamos entender que, quando dois jogadores se conectam através da internet para jogar um contra o outro a situação muda. Nossa situação atual faz com que na hora de obter os comandos dos jogadores, um pacote precisa ser enviado e recebido do outro lado para que a simulação aconteça e isso sem sombra de dúvidas leva mais tempo do que a situação original.

Como podemos perceber pela imagem acima, o jogo não irá simular o próximo estado ou *frame* até que os inputs cheguem, ou seja, isso gera um *delay* do *input* que é igual ao tempo necessário para enviar o pacote de uma máquina a outra. O jogo portanto perde sua responsividade por causa disso, tornando os controles e a memória musculares dos jogadores não confiáveis. Imagine se o pacote simplesmente se perde na rede. O jogo literalmente não consegue simular o próximo estado então o jogo congela e tenta simular o próximo estado. Isso gera um desconforto extremo e pode ser o motivo da quebra de vários controles por aí.

Ainda podemos adicionar o fato de que caso o jogo congele, os inputs do jogador local serão perdidos também durante aqueles *frames*, em consequência inputs de movimento necessários para usar golpes especiais como o quarto de meia lua por exemplo podem ser "comidos" pelo jogo.

Claro que essa situação é o que acontece se o jogo possui o método de *delay-based netcode* implementado, que no geral funciona razoavelmente bem para conexões de 60-80 ms, como já foi mencionado, porém todos sabemos que existe uma instabilidade na conexão de qualquer pessoa na internet e até mesmo essas flutuações podem acabar atrapalhando a experiência dos jogadores não importa o quão bom a latência seja.

Então como pode-se solucionar esse problema sem mudar o *loop* intrínseco ao funcionamento desse estilo de jogo, mas melhorando a responsividade deles quando jogados através das redes? A resposta está em voltar ao futuro.

A ideia principal: Indo de volta ao futuro

O grande problema é a espera. Ao ter que esperar os inputs do usuário que vem dos confins da rede se perde completamente a consistência e precisão necessária para deixar os jogos divertidos. Portanto, a solução aqui é nunca esperar! Caso tenhamos os inputs do jogador conectado pela rede ótimo, caso contrário, vamos simular esses inputs tentando predizer o que o jogador do outro lado da rede vai fazer.

Com essa predição, o jogo pode simular imediatamente o próximo estado sem problemas. Porém é claro que essa predição pode estar errada (incrivelmente na maioria dos casos ela não está!), e para isso guarda-se inputs do jogo e estados enquanto a simulação ocorre para que seja possível corrigir a predição incorreta, caso realmente haja um erro na simulação.

Mas apenas com essa predição feita já se elimina completamente o *delay* que era necessário para obter o *input* do jogador pela rede não é necessário. O jogador vai se sentir como se estivesse jogando offline, seus comandos serão precisos e responsivos como se estivesse fora de rede.

Agora a grande pergunta do leitor deve ser: Certo, mas e quando a predição estiver errada? Que tipo de inteligência artificial mirabolante conseguiria prever os inputs de jogadores reais? O que se faz quando a predição erra? A realidade é que não é necessário muito para predizer o que vai acontecer no próximo estado dado o anterior nesse estilo de jogo. Na grande maioria dos casos em jogos de luta, o jogador está em uma ação comprometida. Não importa o que ele aperte, se o seu personagem está realizando um de seus golpes, ele o continuará fazendo indiferente dos inputs do jogador. Adam Heart, o entrevistado que fala sobre a implementação do *rollback* no jogo **Killer Instinct** menciona que "Na maioria dos casos, eu diria 80% do tempo em um jogo de luta você está em uma ação comprometida (...) não há nada que você aperte que possa mudar

isso".

Não só isso, como o próprio criador do *rollback*, Tony Cannon menciona que em média, jogadores não pressionam tantos botões assim: "Quão rápido as pessoas mexem no controle? Quantos *inputs* você tem? É provavelmente não mais que 5 por segundo, e isso é muito.(...) Então isso significa que que se temos 60 *frames* por segundo, isso significa que você esta certo mais que 90% do tempo"

Portanto a tática é simples. Caso os inputs do oponente que está conectado pela internet não estejam presentes o jogo simplesmente continua a simular o jogo normalmente fazendo uma aposta de quais seriam tais inputs e guardando os estados resultantes dessa escolha. Uma vez que o *input* do *frame* em questão chegue e, caso ele seja divergente do *input* feito, o jogo que já havia ido uma quantidade de *frames* para o futuro volta a este, agora com a entrada correta, e então simula novamente com essas entrada até o estado atual que deveria ser renderizado. Com condições ideais, os efeitos visuais dessas mudanças ficariam completamente imperceptíveis, sempre caindo nos *frames* de começo dos movimentos dos personagens.

E sim, todos esses cálculos são feitos em apenas um *frame*, portanto, o jogador não perceberia qualquer coisa a não ser o próximo *frame*. Isso dá a confiança e certeza para os jogadores de que os botões que serão apertados vão sair consistentemente no jogo, não importando se estão online ou não.

Mas nem tudo são flores. Se acredita que seja possível colocar *rollback* em qualquer jogo já existente. Mas existem certos efeitos visuais que realmente podem fazer a experiência mesmo com *rollback* serem bem estranhas. Por isso existem certos fatores que são necessários e outros que são desejáveis para que o jogo em questão suporte o *rollback*.

Quero *rollback* também, como faz?

Pode-se inferir que o sistema descrito acima traz sem sombra de dúvidas uma melhor experiência online para jogadores de jogos que possuem o *rollback*. Mas existe uma certa prioridade que precisa ser dada quando se trata da estrutura interna do jogo, uma estrutura tal que, sem ela não é nem possível implementar a técnica descrita acima.

Ou seja, uma série de cuidados precisam ser tomados para que o jogo sendo desenvolvido suporte com eficiência esse sistema de *netcode*. Começando pelo aspecto mais importante, o determinismo. Isso em si não é um problema, porque a maioria dos jogos de luta que estamos falando aqui são determinísticos, ou seja, dado uma entrada específica temos a certeza do resultado esperado e ele sempre será igual.

A próxima estrutura essencial para que o jogo consiga voltar para trás é a serialização. Um conceito não muito distante dos cientistas da computação, basicamente o ato de transformar uma dada estrutura de dados em um formato que possa ser guardado e recuperado posteriormente. Não é difícil entender o porquê ele é necessário aqui. Para saber do passado precisamos de um mecanismo eficiente para guardá-lo e recuperá-lo.

Ser determinístico e possuir um sistema de serialização não basta. De nada importa se não pudermos mudar a lógica por debaixo dos panos sem renderizar nada na tela para simular o passado e calcular o presente correto. Então é essencial ao *rollback* também que haja uma separação entre a lógica do jogo e a sua apresentação na tela. A lógica precisa ser independente de tudo dentro do loop de um frame.

Com isso, pode-se começar a pensar em introduzir o *rollback* no jogo, porém a eficiência aqui é chave, porque novamente, tudo precisa ser calculado dentro de 1 *frame*! Otimizações no geral são bem vindas e provavelmente necessárias, desde o método de serialização até as partículas utilizadas nos personagens, tudo que possa ser otimizado para deixar o cálculo dos diversos estados que terão de ser feitos ao corrigir o

presente é bem vindo.

Outros aspectos que precisam ser levados em conta são por exemplo, simulações de particulares, tempo de vida de objetos, efeitos sonoros, sistema de animação, a interface de usuário e até mesmo o sistema de detecção de dessincronização. Tony Cannon especificamente menciona ate: "não faça VFXs grandes e disruptivos ou texto na tela nos primeiros 2 ou 3 *frames*, o quão longo você espera que o seu *rollback* seja(...) Não projete movimentos que tenham deslocamentos muito grandes nos primeiros três *frames*...".

Conclui-se que, priorizar o *rollback* é um fardo que é colocado nos desenvolvedores que aumenta drasticamente a experiência online de seus jogos. Existem uma série de preocupações a mais que precisam ser levadas em conta, mas, se a satisfação dos jogadores for o foco, é um esforço a mais que vale a pena.

Na próxima seção vamos mostrar que na realidade esses dois métodos não são excludentes e a mistura deles pode estabilizar ainda mais a situação desses jogos quando se trata da experiência online de seus jogadores.

3.8 A solução mista

Anteriormente, ambos principais *Netcodes* em uso atualmente foram discutidos, juntamente com suas vantagens e desvantagens na perspectiva de implementação e de aprovação pelos jogadores. Agora iremos apresentar a solução que mistura aspectos de ambos para entregar um *Netcode* ainda mais desejável pelo usuário e não mais difícil de implementar do que o *rollback*.

O pequeno problema do *Rollback*

Em geral, o *Rollback* é extremamente robusto, dando a ambos jogadores alta velocidade de *input*, na verdade podemos dizer que é velocidade até demais. Como nenhum *input* dos jogadores sofre absolutamente nenhum atraso, não existe nenhuma janela para transmissão desse *input* pela rede. Ou seja, todo *input* acaba desencadeando um *rollback* já que a transmissão em um *frame* de um *input* é muito difícil.

Pela natureza do atraso do pacote de *input*, o problema não é o retorno visual que ocorre no *rollback*, mas sim o impacto em performance, visto que o processo de carregar um estado serializado, simular várias iterações do game *loop* e então serializar o novo estado é um procedimento computacionalmente intensivo. Se esse procedimento ocorrer em todos os *frames*, o impacto será perceptível.

Delay-Rollback Netcode

Para evitar o desencadeamento de um *Rollback* em todo *frame*, a grande maioria das implementações desse *Netcode* pegam emprestado um conceito chave do *Delay-Based Netcode*, o atraso de *input*. Adicionando um atraso de *input* razoável, como 3 *frames*, já é o suficiente para limitar o desencadeamento de *Rollback* apenas para eventos de perda ou piora de rede.

Para manter a consistência, esse atraso pode ser escolhido pelo usuário e não muda durante a partida, ou decidido pelos desenvolvedores e não deve mudar durante a partida. Assim a precisão e ritmo dos inputs não sofre alteração e portanto os jogadores sentem fluidez.

3.9 GGPO, a ferramenta que mudou o jogo

Nesta seção vamos falar um pouco do GGPO que é uma ferramenta cujo a origem se confunde com a do próprio conceito de *rollback*. Vamos passar um pouco pela história de como ela foi feita, até no momento atual onde ela é uma ferramenta que é disponibilizada para qualquer desenvolvedor que queira utilizá-la em seus jogos.

Uma breve introdução ao GGPO

GGPO, o acrônimo para *Good Game Peace Out* (Bom jogo, falou), é uma SDK(*Software Develop-*

ment Kit), ou seja, é um conjunto de ferramentas ou até mesmo uma biblioteca para o desenvolvimento de jogos online com zero latência de *input*.

A ferramenta é pioneira no uso de *rollback* para jogos *peer-to-peer* online e já foi utilizada em diversos jogos que estão no mercado nos dias de hoje. Títulos atuais como por exemplo **The King of Fighters XV** (2022) e **Melty Blood: Type Lumina** (2021) foram implementados utilizando o *middleware* criado por Tony Cannon em 2009. Existem muitos outros exemplos de jogos de luta que também adaptaram seus jogos para utilizar a ferramenta ou tiveram seu desenvolvimento desde o início utilizando o GGPO, por exemplo, **Skullgirls** (2012) , **Pocket Rumble** (2017), **Punch Planet** (2017), **Them's Fightin' Herds** (2020), **Fantasy Strike** (2019), **Street Fighter III: 3rd Strike Online Edition** (2011), **Samurai Showdown** (Jogo de 2019 que receberá uma atualização em 2023 adicionando rollback através do GGPO), etc.

O *rollback* utilizado é aquele que foi explicado em sessões anteriores e a ferramenta tenta facilitar a integração do algoritmo de *rollback* e a lógica de baixo nível de rede ao *loop* de um jogo qualquer já existente: "Se você simplesmente implementar a funcionalidade de salvar o estado do jogo, carrega-lo novamente e executar um *frame* do jogo sem renderizar o resultado, GGPO consegue tomar conta do resto." O excerto anterior está no site oficial da ferramenta.

Atualmente, a SDK está disponível para *Windows* com *Visual Studio Code 2019* e *CMake*. Ela está disponível em um repositório no *github* e está sob a licença MIT que dá a permissão para qualquer um reutilizar o código para qualquer propósito, até mesmo que seja parte de software proprietário, desde de que aquele que use inclua uma cópia do licença MIT original em sua distribuição. Além disso, o usuário pode fazer mudanças ou modificações ao código para que atenda suas necessidades.

Vamos agora pincelar um pouco de como foi o desenvolvimento da ferramenta nos baseando principalmente em uma palestra dada por Tony em 2017

Um pouco mais de história

A ideia principal que o Tony teve desde o início era mascarar as inconsistências das redes através de alguma "magica". Essa mágica seria simular sempre, mesmo que o jogo não tenha o *input* dos dois jogadores, arranjar um jeito de simular o próximo *loop* de qualquer jeito e lidar com as consequências depois. Mas mesmo assim, sua ideia de ficar indo e voltando para o presente parecia uma ideia meio boba. Será que não iria gerar efeitos visuais esquisitos, os personagens sairiam teleportando pela tela, etc.

Cannon percebeu que na realidade poderia muito bem funcionar para jogos de luta, uma vez que não fosse necessário dar o *rollback* em muitos *frames*. No caso em que a *rollback* ficasse dentro do limite de *start-up* da maioria dos movimentos, os efeitos visuais causados seriam quase imperceptíveis até mesmo para os jogadores mais observadores.

Mesmo assim, isso tudo era uma grande teoria do criador do GGPO, então ele começou a cogitar como ele testaria essa teoria. Primeiramente ele precisaria de um jogo porém, ele não era desenvolvedor e mesmo se fosse e criasse um jogo, não haveriam tantas pessoas assim querendo jogar seu jogo. Mas ele pensou que existiam vários jogos clássicos de luta que as pessoas adorariam jogar e rodar em emuladores.

Então um emulador foi escolhido, o **Final Burn Alpha** (hoje em dia infelizmente o projeto está paralizado), porque ele era *open-source*, então o código poderia ser modificado contanto que nenhuma monetização fosse feita, possuía uma ótima licença não comercial e o mais importante de tudo, tinha suporte a serialização, que é um dos aspectos principais para conseguir salvar estados passados e poder compará-los sempre que necessário, um dos principais aspectos do *rollback* como já mencionado. A plataforma possui suas desvantagens, ela era não determinística e *crashou* diversas vezes, porém o Tony levou esse projeto em seu tempo livre até que ele conseguiu finalizá-lo em um período de 8-12 meses.

Então ele começou a realizar testes em cima do

código que ele havia implementado. O primeiro foi ele contra ele mesmo em seu próprio computador e funcionou sem problemas. Depois ele testou dentro do seu próprio estado da Califórnia e funcionou também sem problemas e então ele conseguiu um teste entre a costa Leste e Oeste dos EUA, Califórnia até Virginia. O jogador do outro lado da América disse "eu não sei o que você está fazendo mas está incrivel". E finalmente, eles conseguiram fazer um teste entre os EUA e a Austrália e o *feedback* não poderia ter sido mais positivo: "O jogo parece feito. Incrível parecer offline com uma conexão internacional".

Sendo que isso não era o suficiente, então Tony se juntou com seu irmão Tom para criar uma ferramenta própria para testar o código em grande escala e com diferentes condições de rede diversas pelo mundo, foi então que o site GGPO.net surgiu. O site era na verdade uma versão do emulador base que eles utilizavam com algumas funcionalidades a mais: a possibilidade de um chat, salões para jogos específicos, *replays*, assistir ao vivo partida de outros jogadores, e um sistema de desafiar/aceitar partidas contra outros jogadores do mundo.

Foi então que o Tony percebeu que ele não era o único que gostaria de jogar aqueles jogos de luta clássicos online contra outras pessoas online e finalmente eles tinham a oportunidade de fazer isso, mesmo que de maneira tecnicamente "ilegal". Jogar estes jogos sem possuir os ROMs desses jogos.

O cliente do GGPO possui um mapa com o histórico de jogadores e era extremamente promissor ver jogadores de todos os cantos dos EUA jogando um contra os outros e até mesmo jogos entre LA e Tokyo.

Tony menciona que esse período foi extremamente importante em termos de aprendizado tanto em relação a internet quanto o GGPO e torná-la a melhor ferramenta possível. Jogar entre os dois hemisférios (EUA e Japão por exemplo) do mundo usando GGPO parecia funcionar bem, muito provavelmente devido a boa infraestrutura de rede, com fibra óptica pelo pacifico e infraestruturas de rede de primeiro mundo no Japão e Coreia por exemplo.

É mencionado que a América do Sul era horrível devido a infraestrutura precária da rede aqui em baixo. De fato era tão ruim que os jogadores que tentavam usar a plataforma tentavam sair de suas redes locais o mais rápido possível para ter bons jogos. A experiência com a América do Sul foi muito frutífera para a plataforma porque a comunidade desses países queriam muito que a plataforma funcionasse e trouxesse uma robustez para o GGPO. Os irmãos Cannon aprenderam que conseguir diminuir não que sejam bytes de um pacote poderia realmente tornar a experiência daqueles que jogavam em condições um pouco piores consideravelmente melhor.

Inclusive Tony menciona que não acreditava muito no *feedback*, além de ter a barreira linguística, não fazia sentido para o engenheiro que retirar alguns bytes pudesse realmente fazer tanta diferença. Ele inclusive chegou a testar fazendo atualizações no cliente que na realidade não mudavam nada e o *feedback* da comunidade condizia com a falta de mudança e quando a atualização de verdade chegava realmente o *feedback* positivo vinha.

Então os irmãos deixaram a ferramenta evoluir organicamente durante 18 a 24 meses para tornar o código melhor, otimizar sistemas, fazer o código se tornar robusto depois de tantos testes.

Um dos aspectos mais importantes que eles perceberam foi o balanceamento do que eles chamam de *"rift"*(fenda). O *rift* nada mais é do que a distância de *frames* renderizados entre os dois jogadores que gera uma percepção assimétrica da qualidade do jogo. Em outras palavras, o jogo pode estar liso sem nenhum efeito visual anormal de um lado e estar completamente caótico do outro devido a essa propriedade.

Por exemplo. Imagina que temos dois jogadores, o número 1 e o número 2 e que o primeiro se encontra no *frame* 15 enquanto o segundo no *frame* 18. Considere também que leva uma média de 4 *frames* para transmitir um *frame* através da rede. O jogador dois portanto enviará o *input* do *frame* 18 através da rede, que chegaria no *frame* 19

e geraria possivelmente um *rollback* de apenas um único *frame* que é ótimo. Porém do outro lado, os inputs do *frame* 15 que o jogador envia seriam recebidos do *frame* 22 do jogador 2, gerando um possível *rollback* de 7 *frames*, que já pode gerar efeitos visuais bem negativos.

Uma vez detectado a solução do *rift* não é muito difícil. O que se pode fazer é pedir para o jogador que está *frames* a frente para esperar por alguns *loops* de jogo e não renderizar nada, para parear um pouco mais a situação. Imagina que no nosso exemplo o jogador 2 que está 3 *frames* a frente espere esses 3 *frames* e então o *input* recebido nos dois lados vindo dos respectivos outros jogadores geram um *rollback* de 4 *frames* que considerando a situação está ótimo. Tony percebeu que deixar o *rift* ficar maior do que 8 *frames* gera os mesmos efeitos de imprecisão e frustração que o *delay*, portanto não valeria nem a pena e a sua ferramenta, GGPO, garante que o *rift* nunca ultrapasse esses 8 *frames*.

Até mesmo em 2017, o ano em que Tony deu a palestra sobre o GGPO o site criado GGPO.net já não era mais acessível. A ferramenta estava obtendo tanto sucesso que estava começando a captar alguns usuários de negócios chineses e coreanos que estavam começando a entrar na falência. Esses negócios ofereciam um modelo onde essencialmente você poderia jogar os jogos de luta online que estavam disponíveis no GGPO online, só que o GGPO era de graça é melhor.

Então o que começou a acontecer foram ataques DDoS no servidor que o Tony possuía. Ele até tentou melhorar os *firewalls*, fechar portas, comprar servidores maiores, porém os ataques continuavam e chegou a um ponto em que a própria ISP não o queria mais como consumidor. Ataques de DDoS afetam não só o servidor que está sendo atacado mas também tudo que está ao seu redor. Então as pessoas que atacavam o GGPO estavam gerando DDoS na ISP inteira.

Em 2009, Tony subiu novamente o site GGPO.net que agora fala sobre a sua SDK para tornar mais fácil o desenvolvimento de jogos *peer-to-peer* online.

Good Game Peace Out

Podemos concluir com essa sessão que o *rollback* não é apenas um conceito que funciona mas um método que já está sendo colocado em prática por alguns desenvolvedores e que é muito benéfico para o cenário de jogos em geral. Agora o que nos resta é apertar *play* e se divertir.

3.10 Exploits em Netcodes

Apesar de boas soluções para um tremendo problema lógico de algoritmos distribuídos, os *Netcodes* de qualquer jogo apresentam uma porta de entrada para *exploits*. Nesta seção iremos discutir um pouco como esses *exploits* acontecem.

Delay Based e Rollback

Como esse artigo trata de Netcodes de jogos de luta, iremos iniciar com os Netcodes apresentados, como funcionam os exploits existentes e como evitar os mesmos.

Lag Switching

Esse *exploit* é extremamente simples e elegante, e por isso é difícil de ser evitado. Discutimos na sessão do *Delay Based Netcode* que caso um dos jogadores não receba o *input* do adversário, ambos os jogos pausam até a conexão ser restabelecida. Isso faz com que a decisão de ambos fique sincronizada.

Um exploit extremamente simples nesse caso é simplesmente interromper o tráfego de rede manualmente, causando uma pausa no meio de uma sequência sensível de inputs. O adversário irá perder o ritmo e quando a rede voltar, provavelmente irá falhar a sequência.

Lag Switching pode ser feito no meio físico, por meio de um interruptor com botão, ou no meio digital por meio de softwares especializados. Em software, a maneira mais simples de construir um *lag switch* é adicionando e removendo regras

no *firewall* do sistema operacional.

Figura 3.2: Lag Switch físico, compatível com todos os aparelhos que utilizam cabo de rede

Evitar o *Lag Switch* é uma tarefa difícil, já que o comportamento é extremamente similar a perda normal de pacotes da rede. Quando se trata de *Lag Switches* em software, um *Anticheat* pode detectar mudanças no *firewall* que envolvem o jogo em tempo real e assim detectar o uso. Mas no quesito físico é impossível diferenciar a perda normal da perda forçada. Para tratar isso, partidas com conexões cheias de perda são anuladas, em geral essa solução é boa pois no pior dos casos uma partida ruim com jogadores honestos foi anulada, e no melhor uma partida ruim com um jogador desonesto foi anulada.

Pode-se ver exemplos desse comportamento na bibliografia. O primeiro exemplo consiste em uma partida de Tekken 7 onde o jogador controlando o personagem Kazuya possui um lag switch automático. Cada ataque do oponente, que controla o personagem Armor King, gera uma instância de congelamento do Delay Based Netcode. Isso retira o ritmo do jogador que controla o Armor King e dá tempo de reação para o jogador controlando o Kazuya, que agora pode pensar no melhor "punish" para o ataque.

Forced Rift

Quando se trata do *Rollback*, percebemos que o *Netcode* é mais tolerante a perda de pacotes, não congelando o jogo quando um pacote é perdido. Contudo, uma desvantagem é o *Rift*, que ocorre quando o ritmo de envio de pacotes diminui para um jogador, o que acaba machucando a experiência de seu adversário. *Rift* é um problema corrigível e evitável, pelo menos as consequências não simétricas dele, caso uma instância do jogo perceba que a outra está sofrendo *rift*, passa a enviar pacotes em um menor ritmo também, pelo menos igualando a situação.

Contudo nem todos os jogos possuem esse sistema de correção de *Rift*. Lembrando que o *Rollback* já quebra uma barreira lógica de sempre decidir corretamente, assumindo que eventualmente vai decidir corretamente. Caso o *Rift* não seja corrigido, o *Rollback* deixa de ter a garantia de que eventualmente vai decidir corretamente, ou seja, o estado do jogo será dessincronizado para os dois jogadores, resultando em uma desconexão ou uma vantagem para o jogador que manda menos pacotes.

O Forced *Rift* consiste em um simples software que pode fazer inúmeras coisas para diminuir o ritmo de pacotes enviados. Inundar a interface de rede por dentro com pacotes sem rumo. Modificar variáveis do programa em execução etc... É menos simples de se implementar do que o Lag Switch mas é efetivo.

Outros *Netcodes*

Neste artigo não foram discutidos outros *Netcodes* para outras arquiteturas de jogos online, isso pois quando se trata de qualquer arquitetura P2P, as duas dominantes são *Delay Based* e *Rollback*. Contudo, existem arquiteturas diferentes para outros gêneros, onde existe um servidor central que possui um nível mais alto de autoridade de decisão.

Autoridade em Arquiteturas de Rede para Jogos

Quando falamos de autoridade, estamos descrevendo a capacidade de decisão de uma instância sob um aspecto do algoritmo ou programa. Para jogos online temos vários exemplos e níveis de autoridade.

- *MMOs* tendem a dar autoridade da movimentação do personagem do jogador a instância cliente, enquanto todo resto é autoridade do servidor, como dano do jogador dano dos inimigos etc.
- *FPSs* casuais tendem a dar autoridade de movimentação e dano a instância cliente, ou seja, todos os tiros e movimentos do jogador são autoridade própria e por isso não sofrem modificações ou correções.
- *MOBAs* tendem a dar toda autoridade de movimentação, habilidades, etc... ao servidor e apenas autoridade sob a câmera para o cliente. Tornando a movimentação do personagem jogável dependente do ping do cliente para servidor.

Existem mais exemplos, mas estes nos mostram algumas falhas possíveis e *exploits*. Uma falha possível é a perda de pacote com informações de movimento. Como a autoridade é do cliente o mesmo tem poder sob seu movimento mas o servidor toma autoridade quando há uma perda por não receber o movimento para repassar aos outros jogadores, o que resulta no chamado *rubber banding* (O jogador parece voltar para sua posição passada como um elástico).

Por outro lado, se o servidor não toma autoridade na perda de pacotes, abre uma janela para exploits. Por exemplo, um jogador pode usar uma Lag Switch para forçar a perda, tornando-se parado e invulnerável para seus inimigos. Andar até um inimigo (Que também está parado e invulnerável), dar um tiro e reconectar sua rede. Quando isso ocorre, o servidor recebe as informações de movimento e de dano do jogador e com isso mata o inimigo do mesmo, que recebeu um tiro sem nenhum aviso.

Outro simples *exploit* é modificar um parâmetro de autoridade do cliente, por exemplo a câmera em MOBAs. *Exploits* para o jogo **League of Legends** tendem a simplesmente alterar a câmera do jogo para o modo espectador, o que permite se ver muito mais informações do que o normalmente permitido. E o servidor nunca irá saber pois informações da câmera do jogo não são enviadas ao servidor.

Em geral a balança de autoridade entre cliente e servidor nessas arquiteturas tem que ser extremamente bem pensada, pois *exploits* nestes casos são mais devastadores. Como a conexão não é *P2P* e a autoridade não é dividida igualmente para todos, *exploits* podem não atrapalhar um jogador enquanto atrapalham muito outro, similar a um *Rift* em *Rollback* mas muito pior.

Evitar esses *exploits* é difícil pois envolve, fiscalizar parâmetros que não são da autoridade do servidor ou diminuir a autoridade do cliente em si, o que diminui o sentimento de controle dos jogadores.

3.11 Considerações finais

Esperamos que o artigo tenha conseguido introduzir o problema de *Netcode*, assim como as duas formas principais de resolução para este problema que foram desenvolvidas com o passar do tempo.

Por meio deste artigo, buscamos mostrar como a solução para um problema geral de redes foi descoberta graças a necessidade vinda de um espaço de entretenimento, os jogos de luta. Não só isso, mas como a natureza dos jogos denota a natureza das soluções encontradas.

As soluções encontradas fazem parte da caixa de ferramentas de criadores de algoritmos distribuídos. Como lidam com o problema lógico geral e também se adaptam às situações reais de redes, são soluções altamente desejáveis e continuam sendo aplicadas até o presente momento.

Por fim, demonstramos como a mera existência da necessidade de um *netcode* introduz aspectos de insegurança dos jogos. Apresentamos *exploits* relativamente simples e complexos para cada nível de segurança dos *Netcodes* e também como a existência de *exploits* pode se estender para cada arquitetura de rede, seja baseada em servidores ou P2P.

3.12 Bibliografia

[1] 8 frames in 16ms: Rollback networking in mortal kombat and injustice 2. URL: https://www.youtube.com/watch?v=7jb0FOcImdg&ab_channel=GDC.

[2] Analysis: Why rollback netcode is better. URL: https://www.youtube.com/watch?v=0NLe4IpdS1w&t=426s&ab_channel=Core-AGamin.

[3] Escape from tarkov - when you quickly need to use lag switch. URL: https://www.youtube.com/watch?v=fqMklZbCt_8.

[4] Evo 2017: Ggpo panel. URL: https://www.youtube.com/watch?v=k9JTIn1SVQ4&t=2112s&ab_channel=evo2kvids.

[5] Explaining how fighting games use delay-based and rollback netcode. URL: https://arstechnica.com/gaming/2019/10/explaining-how-fighting-games-use-delay-bas.

[6] Fight the lag! the trick behind ggpo's low latency netcode. URL: https://drive.google.com/file/d/1nRa3cRBQmKj0-SEyrT_1VNOkPOJWNhVI/view.

[7] The fighting game glossary. URL: https://glossary.infil.net/?

[8] freeze your enemys lag switch escape from tarkov 2022. URL: https://www.youtube.com/watch?v=W024dom0I9I.

[9] Ggpo | rollback networking sdk for peer-to-peer games. URL: https://www.ggpo.net/.

[10] The ggpo repository. URL: https://github.com/pond3r/ggpo.

[11] Netcode [p1]: Fightin' words. URL: https://ki.infil.net/w02-netcode.html.

[12] Online fighting games during covid: How rollback helps us connect. URL: https://arstechnica.com/gaming/2021/02/how-guilty-gear-saved-its-online-play-in-a-pos.

[13] Talking netcode with adam "keits" heart. URL: https://www.youtube.com/watch?v=1RI5scXYhK0&ab_channel=HoldBacktoBlock.

[14] Worst type of cheating in fighting games. URL: https://www.youtube.com/watch?v=ahiTENh8Pm0&t=84s.

Capítulo 4

Deep Web: além da superfície

Ryan Guerra Sakurai

OBS: Para ver o artigo na íntegra, com imagens maiores, clique **aqui.**

4.1 Resumo

Muito popular entre canais de curiosidade e teorias da conspiração no *Youtube*, *Deep Web* é um conceito cercado de mitos, desinformação e sensacionalismo. Visto isso, o objetivo deste artigo é desmistificar e explicar o que de fato é a *Deep Web* e o que a diferencia da "*web* normal"

4.2 Introdução

Apesar dos termos internet e web serem frequentemente utilizados intercambiavelmente, eles não são sinônimos. Por isso, antes de mais nada, é importante diferenciá-los.

A internet é uma rede que conecta computadores, servidores e outros aparelhos em escala global, além da infraestrutura por trás dessa rede. Sua criação data em meados da década de 1960 e ela possibilita o transporte de informações no mundo contemporâneo e tecnologias como a web.

Por sua vez, a web - também conhecida como *World Wide Web* (WWW) - é um sistema de compartilhamento de informações através da internet. Ela é um conjunto de arquivos que podem ser visualizados através de um navegador, como o *Google Chrome*, o *Mozilla Firefox*, o defunto *Internet Explorer* e seu substituto *Microsoft Edge*.

Figura 4.1: Os dez navegadores mais usados no mundo

O compartilhamento de informações na web é feito através de *HTML*, uma linguagem de marcação de hipertexto que, por sua vez, funciona sob os princípios de hipertexto e hipermídia. Hipertexto define o conceito de texto não linear, ou seja, que possui link para outros textos, e hipermídia consiste na coexistência de texto, vídeos, imagens e outras formas de conteúdo na mesma página.

Tendo isso em vista, a *web* possui 3 principais camadas: a *Surface Web*, a *Deep Web* e a *Dark Web*, que vão ser apresentadas a seguir.

4.3 Pré-Requisitos

OBS: Os passos dos pré-requisitos abaixo, são para ser executados em um terminal linux ou usuários de WSL no Windows. Porém para instalação no windows há uma seção chamada "Con-

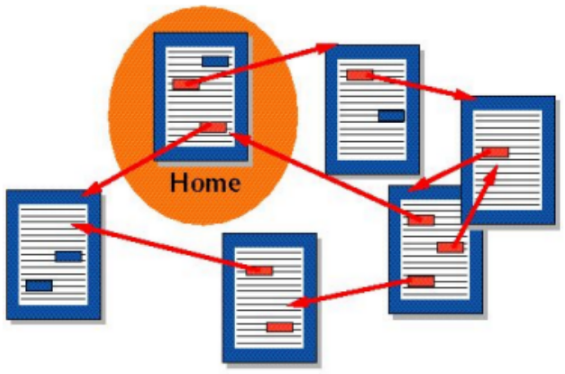


Figura 4.2: Representação visual do conceito de hipertexto

figurando e instalando *Debugger* da linguagem C no Windows" ou você pode clicar nesse link.

Surface Web

A primeira camada da web, a Surface Web, é a parte da web formada por páginas que podem ser encontradas e acessadas através de mecanismos de busca como *Google* e *Bing*. O que torna essas páginas disponíveis é a ação de algoritmos chamados *web crawlers*.

Web crawlers são os algoritmos dos mecanismos de busca responsáveis por fazer a indexação das páginas da *web*. Esses *bots* inspecionam seus códigos *HTML* procurando por dados sobre a página, as indexam e as classificam com base nesses dados. Dessa maneira, quando um usuário faz uma pesquisa no *Google*, ele encontra os resultados que o mecanismo julga serem mais relevantes.

4.4 Deep Web

A segunda camada da web, a *Deep Web*, em oposição à *Surface Web*, é a parte da web composta por páginas que não podem ser acessadas através de mecanismos de busca, ou seja, que não foram indexadas. A maior parte dessas páginas utilizam as mesmas tecnologias da *Surface Web* e, apesar de dificuldades no cálculo de seu tamanho, a *Deep Web* compõe a esmagadora maioria da web.

Figura 4.3: Ilustração da indexação feita por mecanismos de busca

Os motivos para a não indexação das páginas da Deep Web podem ser diversos: necessidade de autenticação ou pagamento, uso de outros protocolos que não sejam HTTP/HTTPS, conteúdo gerado dinamicamente, dificuldade em indexar mídia ou até falta de link entre uma página e outra. A indexação pode, inclusive, ser impedida de maneira voluntária por parte do criador do site, através do arquivo *robots.txt*, que diz aos *web crawlers* quais páginas não indexar.

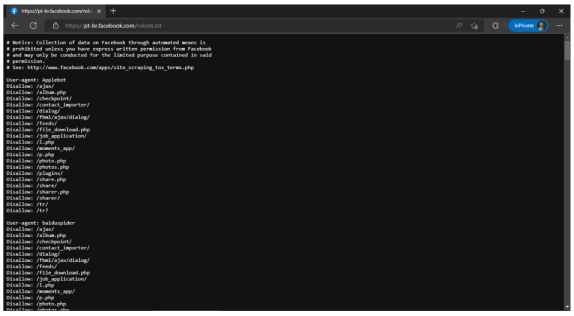


Figura 4.4: Screenshot do arquivo robots.txt do Facebook

Exemplos de páginas que fazem parte da *Deep Web* são: conversas no *WhatsApp Web*, *feed* do *Facebook*, caixa de *e-mails*, salas virtuais do *Google Meet*, bancos de dados, jornais científicos, etc. Portanto, é equivocada a crença popular de que a *Deep Web* seja um lugar hostil e perigoso, formado majoritariamente por sites criminosos e visitado por *hackers*.

4.5 Dark Web

Por último, a camada mais profunda da web é a *Dark Web*. Ela também é formada por páginas que não foram indexadas pelos sites de busca, ou seja, tecnicamente também faz parte da *Deep Web*. Porém, o fator que a diferencia do resto da *Deep Web* é que ela funciona usando como suporte as *darknets*, ao invés da internet padrão.

Darknets são redes construídas em cima da internet que só podem ser acessadas com tecnologias ou configurações específicas. Apesar de existirem várias, a que está mais comumente associada à *Dark Web* é a rede **Tor** - *The Onion Router*.

Tor é uma rede descentralizada mantida por vários voluntários que usa uma técnica chamada *onion routing* para a transmissão de informações entre usuário e servidor, que consiste em passar o pacote por várias máquinas intermediárias, o envolvendo em várias camadas de criptografia. Pode ser feita uma analogia entre essas camadas com as de uma cebola, que é de onde se origina o termo *onion routing*.

No envio de pacote do cliente ao servidor, primeiramente, o dado é encriptado três vezes e encaminhado à primeira máquina, chamada de nó guarda. Essa máquina irá tirar uma camada de criptografia e encaminhar o pacote à máquina intermediária, que, por sua vez, removerá mais uma camada de criptografia e o enviará ao chamado nó de saída. Após a terceira camada de criptografia ser removida, o dado finalmente será encaminhado pelo nó de saída e chegará ao seu destino.

A volta do pacote (sentido servidor-cliente) é feita da maneira inversa. Primeiro o dado é enviado sem nenhuma das camadas de criptografia do Tor (pode ser que o dado já tenha sido encriptado antes de passar pela rede) ao nó de saída, que irá adicionar a primeira. Após isso, será adicionada uma camada de criptografia por máquina, até que o dado seja enviado do nó guarda até o cliente, que removerá as três camadas de uma vez.

A anonimidade nesse sistema se deve ao fato de que um nó conhece apenas o IP do nó anterior e o do nó posterior. Por isso, com exceção do guarda, nenhum dos nós ou o servidor conhecem o IP do cliente, e vice-versa.

Figura 4.5: Ilustração do processo de onion routing

Os sites hospedados na rede *Tor* costumam ter a extensão *.onion* e podem ser acessados facilmente através do navegador *Tor*, que também pode ser usado para acessar sites da *Surface Web*, apesar de vários deles impedirem acesso através da rede *onion*.

Por causa da anonimidade provida pela *Dark Web*, ela muitas vezes é palco de atividades ilegais, como: pornografia ilegal, mercado ilegal, contrato de serviço de hackers, etc. Dito isso, seu acesso não é seguro e não compensa ser feito, se não houver um motivo específico para isso. Apesar disso, a *Dark Web* não é completamente negativa, pois também permite combater a censura em regimes totalitários e facilitar a ação de *whistleblowers*, por exemplo.

4.6 Conclusão

Através deste artigo, conclui-se que apesar do uso intercambiado entre *web* e *internet*, os dois conceitos são diferentes, sendo a web dividida em camadas de acesso. Além disso, é errado dizer que a *Deep Web* é uma rede usada majoritariamente por criminosos, pois esse tipo de atividade se concentra em parte de uma porção da *Deep Web*: a *Dark Web*. Por conta de sua privacidade e anonimato, ela é uma ferramenta que pode muitas vezes ser usada de maneira benigna, porém ainda é insegura e carregada de ilegalidade e perigos.Portanto, ela não deve ser acessada por pessoas leigas e que não tem um bom motivo para acessá-la.

4.7 Bibliografia

[1] 10 navegadores mais usados no mundo. URL: https://www.oficinadanet.com.br/internet/30735-10-navegadores-mais-usados-no-mundo.

[2] A brief history of the internet. URL: https://www.usg.edu/galileo/skills/unit07/internet07_02.phtml.

[3] deep web. URL: https://www.techtarget.com/whatis/definition/deep-Web.

[4] Going dark: The internet behind the internet. URL: https://www.npr.org/sections/alltechconsidered/2014/05/25/315821415/going-dark-the-internet-behind-the-internet.

[5] Hypertext. URL: https://en.wikipedia.org/wiki/Hypertext.

[6] Internet vs. web: What's the difference? URL: https://www.lifewire.com/difference-between-the-internet-and-the-web-2483335.

[7] Tor's most visited hidden sites host child abuse images. URL: https://www.bbc.com/news/technology-30637010.

[8] Web. URL: https://www.techopedia.com/definition/5613/web.

[9] Web crawler: Entenda o que é, quando usar e como funciona. URL: https://neilpatel.com/br/blog/web-crawler/.

[10] Web-indexing. URL: https://computersciencewiki.org/index.php/Web-indexing.

[11] What is a web crawler. URL: https://www.cloudflare.com/pt-br/learning/bots/what-is-a-web-crawler/.

[12] What is hypertext. URL: https://www.w3.org/WhatIs.html.

[13] What is the deep and dark web? URL: https://www.kaspersky.com/resource-center/threats/deep-web.

[14] What is tor and how does it work? URL: https://www.techradar.com/features/what-is-tor-and-how-does-it-work.

Parte II

Projetos

Capítulo 5

Helpfac

Luis Felipi Cruz de Souza
Ryan de Melo Andrade

5.1 Conceito do projeto

Helpfac é uma aplicação WEB onde é possivel realizar perguntas e responder perguntas de outros usuários. Ele foi criado visando um ambiente de estudos onde temos cursos e matérias associadas a estes cursos, de forma que ambos podem sem cadastradas a qualquer momento pelo administrador. Sua principal finalidade é sanar possíveis duvidas de alunos, podendo esses questionamentos serem respondidos por qualquer usuário da aplicação independentemente de curso.

5.2 Pré-requisitos e recursos utilizados

No front-end do projeto foi utilizado HTML, CSS e Javascript. No back-end do projeto foi utilizado Javascript com Node, e, na parte do banco de dados foi utilizado o MySQL, com o construtor de consultas Knex.

5.3 Passo a Passo

- Criação da funcionalidade de visualização dos cursos
- Criação da funcionalidade de visualização das matérias por curso
- Criação da funcionalidade de visualização de perguntas do curso
- Criação da funcionalidade de criação de perguntas
- Criação da funcionalidade visualizar uma pergunta, junto as suas respostas
- Criação da funcionalidade responder uma pergunta
- Criação da funcionalidade login/registro
- Criação da funcionalidade de cadastro, edição e remoção de cursos (ADMIN)
- Criação da funcionalidade de cadastro, edição e remoção de matérias (ADMIN)
- Criação da funcionalidade de exclusão, edição e visualização de perguntas (ADMIN tem acesso a todas, usuário tem acesso apenas as que são suas)
- Criação da funcionalidade de edição e exclusão de resposta (ADMIN pode gerenciar todas, usuário pode gerenciar apenas as que são suas)

5.4 Instalação

1. Clone o repositório

```
git clone https://github.com/
    LuisFSouza/helpfacProject.
    git
```

2. Certifique de que voce tem o npm e o nodejs instalado.

3. Instale as dependências

```
npm install
```

4. Crie um banco de dados em MySQL

```
CREATE DATABASE nomedobanco
```

5. Crie um arquivo .env na raiz do projeto. Modelo:

```
DB_HOST = Digite aqui o hostname
DB_PORT = Digite aqui a porta do
    banco de dados
DB_USER = Digite aqui o usuario
    do banco de dados
DB_PASS = Digite aqui a senha do
    banco de dados
DB_DATABASE = Digite aqui o nome
    do banco de dados criado no
    passo anterior
SECRETKEY = Digite aqui uma
    chave para verificar
    assinatura (sistema de login
    )
PASS_ADMIN = '
    $2b$10$umH10iFUgtO2y8cmh
    tLg8OSg2iA7vT63zPxBu2s4HTRq1rY
     AqHQgC'
```

6. Rode as migrations na seguinte ordem

```
npx knex migrate:up
    20230202112034
    _create_table_users.js
```

```
npx knex migrate:up
    20230131144246
    _create_table_course.js
```

```
npx knex migrate:up
    20230131144251
    _create_table_matter.js
```

```
npx knex migrate:up
    20230131144256
    _create_table_question.js
```

```
npx knex migrate:up
    20230131144303
    _create_table_answer.js
```

```
npx knex migrate:up
    20230131144307
    _create_table_image.js
```

7.Rode as seeds com o seguinte comando

```
npx knex seed:run
```

Com isso, você terá cadastrado no sistema o usuário administrador, cujo email será admin@admin.com e a senha será admin

5.5 Pré-requisitos e recursos utilizados

No front-end do projeto foi utilizado HTML, CSS e Javascript. No back-end do projeto foi utilizado Javascript com Node, e, na parte do banco de dados foi utilizado o MySQL, com o construtor de consultas Knex.

5.6 Execução

1. Certifique que o banco de dados está rodando.
2. Execute o seguinte comando para a aplicação começar a funcionar

```
node server.js
```

3. Acesse o seguinte link:

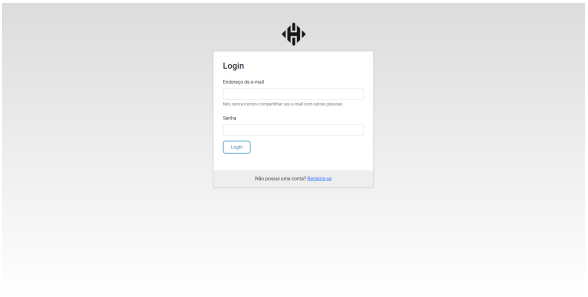
```
http://localhost:3333/login
```

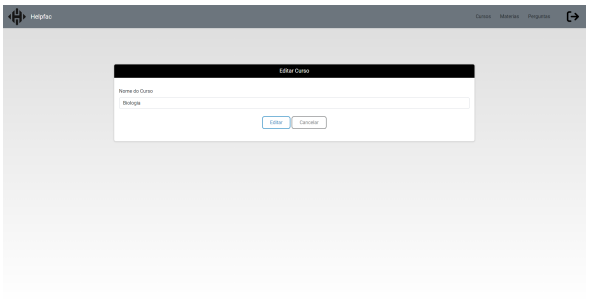
5.7 Repositório

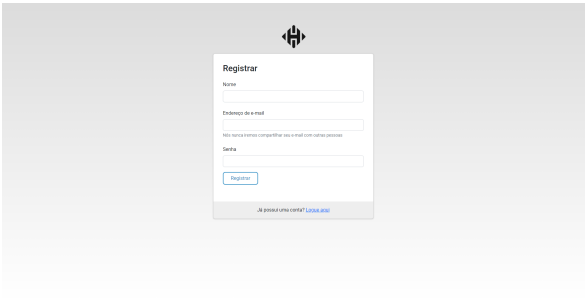
O repositório com os arquivos e demais informações sobre o projeto se encontra em:

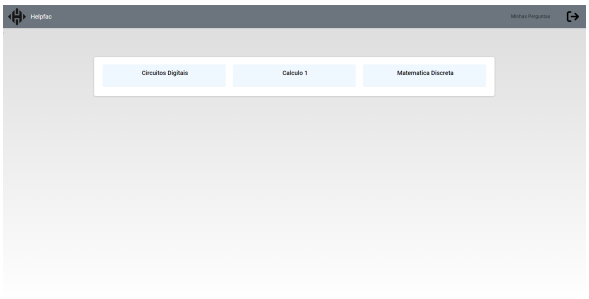
`https://github.com/LuisFSouza/helpfacProject`

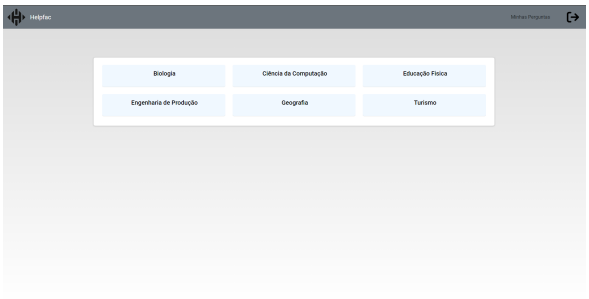
5.8 Imagens

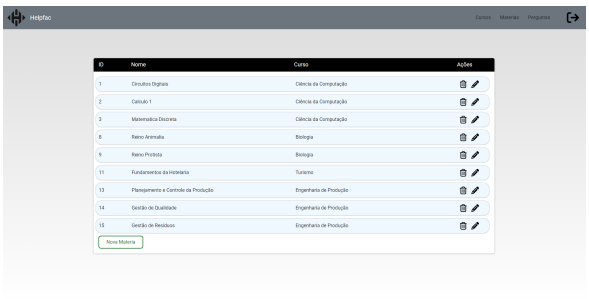

Login
Endereço de e-mail
Nós nunca iremos compartilhar seu e-mail com outras pessoas
Senha
Login
Não possui uma conta? Registre-se

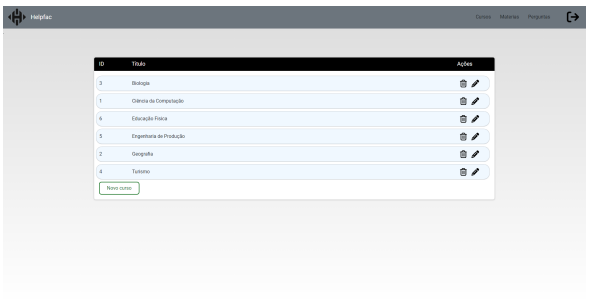

Helpfac
Cursos
Materias
Perguntas
Editar Curso
Nome do Curso
Biologia
Editar
Cancelar

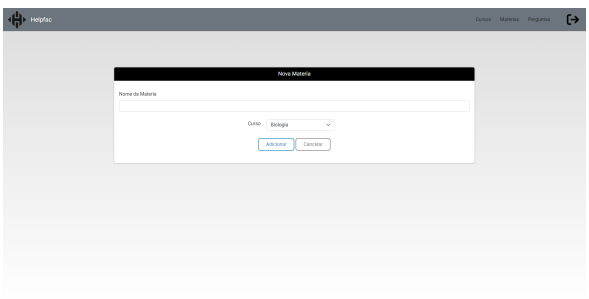

Registrar
Nome
Endereço de e-mail
Nós nunca iremos compartilhar seu e-mail com outras pessoas
Senha
Registrar
Já possui uma conta? Logue aqui

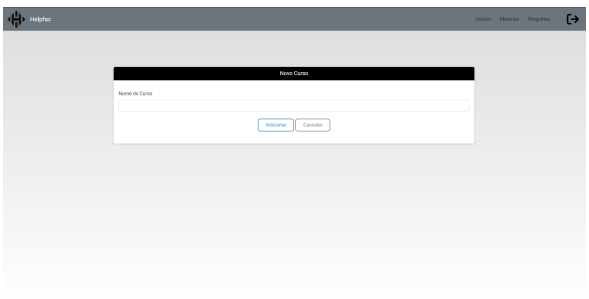

Helpfac
Minhas Perguntas
Circuitos Digitais
Calculo 1
Matematica Discreta

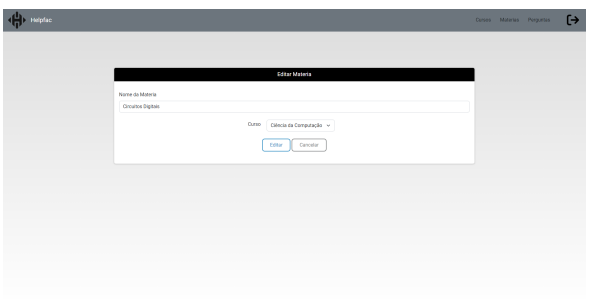

Helpfac
Minhas Perguntas
Biologia
Ciência da Computação
Educação Física
Engenharia de Produção
Geografia
Turismo

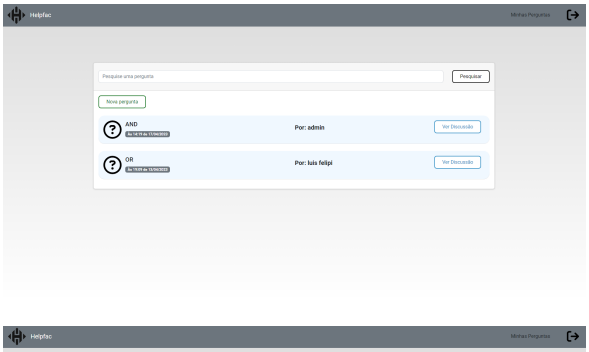

Helpfac
Cursos
Materias
Perguntas
ID
Nome
Curso
Ações
1 Circuitos Digitais Ciência da Computação
2 Calculo 1 Ciência da Computação
3 Matematica Discreta Ciência da Computação
8 Reino Animalia Biologia
9 Reino Protista Biologia
11 Fundamentos da Hotelaria Turismo
13 Planejamento e Controle da Produção Engenharia de Produção
14 Gestão de Qualidade Engenharia de Produção
15 Gestão de Resíduos Engenharia de Produção
Nova Materia

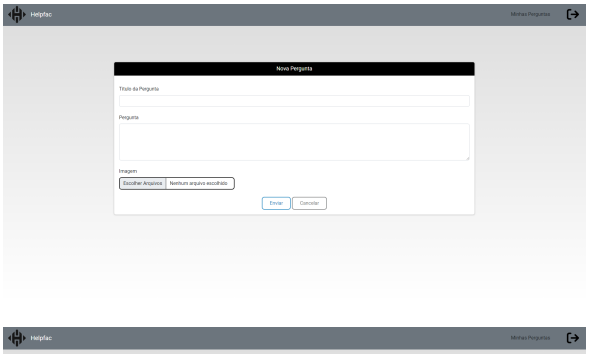

Helpfac
Cursos
Materias
Perguntas
ID
Título
Ações
3 Biologia
1 Ciência da Computação
6 Educação Física
5 Engenharia de Produção
2 Geografia
4 Turismo
Novo curso

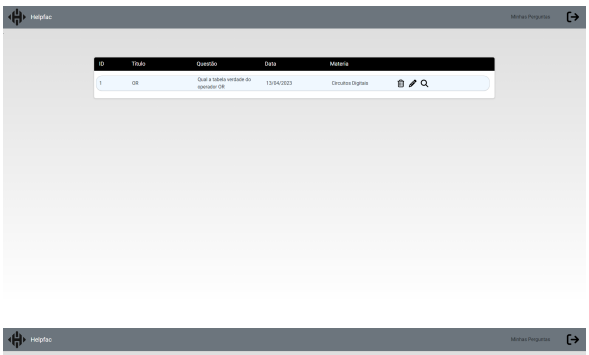

Helpfac
Cursos
Materias
Perguntas
Nova Materia
Nome da Materia
Curso
Biologia
Adicionar
Cancelar

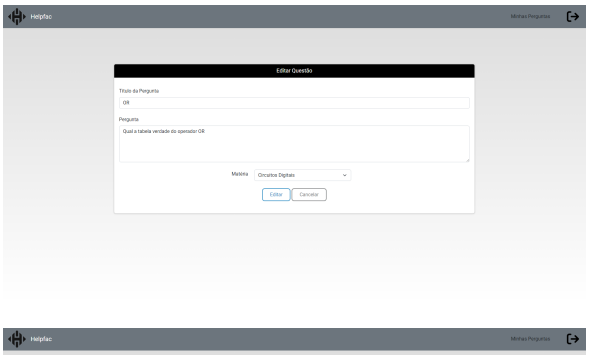

Helpfac
Cursos
Materias
Perguntas
Novo Curso
Nome do Curso
Adicionar
Cancelar

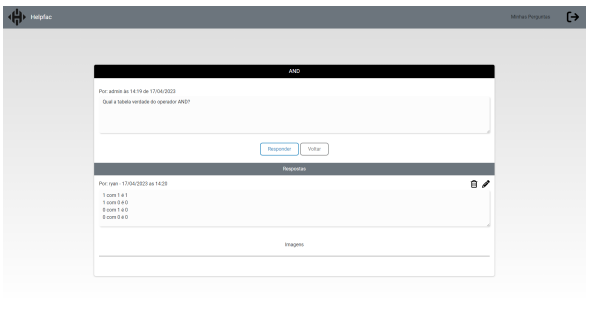

Helpfac
Cursos
Materias
Perguntas
Editar Materia
Nome da Materia
Circuitos Digitais
Curso
Ciência da Computação
Editar
Cancelar

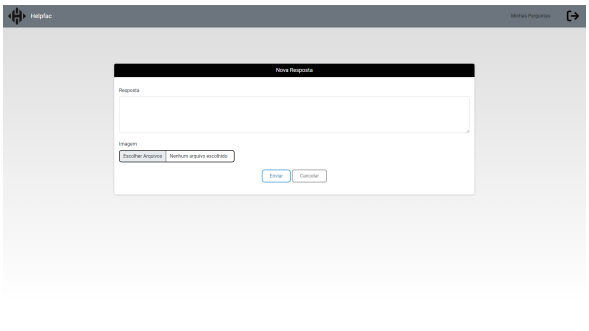

Helpfac
Minhas Perguntas
Pesquise uma pergunta
Pesquisar
Nova pergunta
AND
Por: admin
Ver Discussão
OR
Por: luis felipi
Ver Discussão

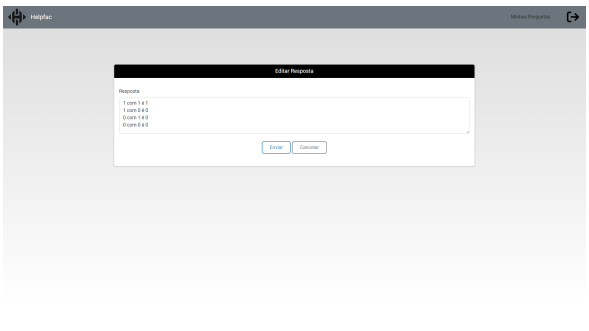

Helpfac
Minhas Perguntas
Nova Pergunta
Título da Pergunta
Pergunta
Imagem
Escolher Arquivos
Nenhum arquivo escolhido
Enviar
Cancelar


Helpfac
Minhas Perguntas
ID
Título
Questão
Data
Materia
1
OR
Qual a tabela verdade do operador OR
13/04/2023
Circuitos Digitais


Helpfac
Minhas Perguntas
Editar Questão
Título da Pergunta
OR
Pergunta
Qual a tabela verdade do operador OR
Matéria
Circuitos Digitais
Editar
Cancelar


Helpfac
Minhas Perguntas
AND
Por: admin às 14:19 de 17/04/2023
Qual a tabela verdade do operador AND?
Responder
Voltar
Respostas
Por: ryan - 17/04/2023 as 14:28
1 com 1 é 1
1 com 0 é 0
0 com 1 é 0
0 com 0 é 0
Imagens


Helpfac
Minhas Perguntas
Nova Resposta
Resposta
Imagem
Escolher Arquivos
Nenhum arquivo escolhido
Enviar
Cancelar


Helpfac
Minhas Perguntas
Editar Resposta
Resposta
1 com 1 é 1
1 com 0 é 0
0 com 1 é 0
0 com 0 é 0
Enviar
Cancelar

Capítulo 6

Globo Fishing

Vinícius Fernandes Terra Silva

6.1 Conceito do projeto

No cerne deste projeto encontra-se a concepção de um website com enfoque no desenvolvimento de conscientização e segurança digital. A proposta central é proporcionar aos usuários uma experiência educativa e prática, visando ampliar a compreensão sobre os métodos utilizados por agentes maliciosos na internet, com ênfase especial em ataques de phishing. Para isso foi criado do zero um website que simula o globo.com, mas que na realidade seria um site malicioso que rouba as informações dos usuários desavisados

Link para o site criado: `https://globo-fishing.vercel.app/`

6.2 Pré-requisitos e recursos utilizados

No Front-end foi utilizado HTML, CSS e javascript, além da hospedagem gratuita da Vercel No Back-end foi utilizado Node.js, MongoDB além de bibliotecas como cors, express, joi, entre outras. Sua hospedagem foi feita gratuitamente com o Render

6.3 Passo a Passo

1. Estudei o site da Globo

2. Construí o HTML e CSS separando por seções

3. Construí um Back-end simples e com banco de dados para poder armazenar os logins

4. Após vários testes fiz o deploy

6.4 Instalação

O projeto já possui um deploy, mas caso queira instalá-lo em sua máquina aqui vai o passo a passo

Para o Front basta ter uma copia do projeto e arrastar o arquivo HTML para o seu navegador

Para o Back-end é necessário ter o MongoDB em sua máquina

1. Crie um arquivo .env e o complete-o como no arquivo .envExample

2. Execute o comando *npm install* no terminal, dentro da pasta do projeto

3. Execute o comando *npm start*

6.5 Execução

Você pode começar explorando a aplicação pela página de início do Front-end. Lá, existe um botão escondido que, ao ser clicado, envia discretamente todas as informações já coletadas pelo site para o seu navegador. Para fazer isso, basta clicar no anúncio de pescaria (que é um botão secreto), em seguida, usar o botão direito do mouse, clique em "Inspecionar". Procure pela seção "Console"; assim, você terá acesso às informações.

Para adicionar novas informações ao banco de dados, basta agir como um usuário comum que deseja fazer login na conta da Globo.

6.6 Bugs/problemas conhecidos

O site está hospedado em serviços gratuitos, então existe uma lentidão para a resposta do servidor, é preciso ser paciente.

6.7 Repositório

O repositório com os arquivos e demais informações sobre o projeto se encontra em:

`https://github.com/Vinicius-Terra/Globo-Fishing`

Capítulo 7

Repositório de algoritmo de ordenação

Daniella Yuka Hirosue
Pedro Henrique Alves de Araujo Silva

7.1 Conceito do projeto

Este projeto foi realizado com a finalidade de auxiliar os calouros da UFSCar-So e novos estudantes de programação sobre complexidade de algoritmos, ao reunir diversos dos algoritmos de ordenação mais básicos e conhecidos em um repositório que busca os ensinar de forma sucinta e prática seu funcionamento.

7.2 Pré-requisitos e recursos utilizados

O grupo utilizou linguagem C para desenvolver a implementação completa do projeto, utilizando apenas as bibliotecas:

1. <stdio.h>
2. <stdlib.h>
3. <time.h>

Foram usados como base os algoritmos ensinados em aula, na matéria Estrutura de Dados, lecionada pelo professor dr. Mario Lizier, na Universidade Federal de São Carlos, campus Sorocaba.

7.3 Passo a Passo

Para o desenvolvimento do projeto foram estudados o funcionamento dos seguintes algoritmos de oredenação:

- Bubblesort
- InsertionSort
- SelectionSort
- MergeSort
- QuickSort
- HeapSort

7.4 Execução

Para executar o projeto em seu terminal, vá até a pasta destino dos arquivos baixados e execute:

```
gcc main.c -o main
```

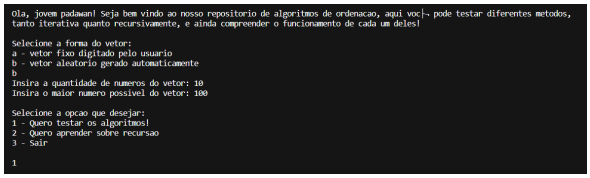
```
./main
```

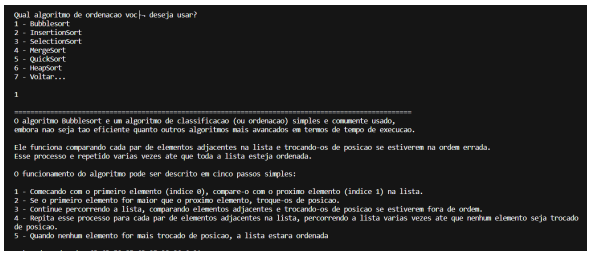
7.5 Repositório

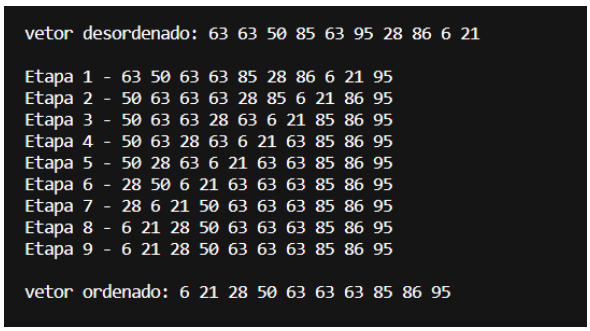
O repositório com os arquivos e demais informações sobre o projeto se encontra em:

`https://github.com/pedrohaas/RepoOrdenacao`

7.6 Imagens


Ola, jovem padawan! Seja bem vindo ao nosso repositorio de algoritmos de ordenacao, aqui voc|~ pode testar diferentes metodos,
tanto iterativa quanto recursivamente, e ainda compreender o funcionamento de cada um deles!
Selecione a forma do vetor:
a - vetor fixo digitado pelo usuario
b - vetor aleatorio gerado automaticamente
b
Insira a quantidade de numeros do vetor: 10
Insira o maior numero possivel do vetor: 100
Selecione a opcao que desejar:
1 - Quero testar os algoritmos!
2 - Quero aprender sobre recursao
3 - Sair
1


Qual algoritmo de ordenacao voc|~ deseja usar?
1 - Bubblesort
2 - InsertionSort
3 - SelectionSort
4 - MergeSort
5 - QuickSort
6 - HeapSort
7 - Voltar...
1
O algoritmo Bubblesort e um algoritmo de classificacao (ou ordenacao) simples e comumente usado,
embora nao seja tao eficiente quanto outros algoritmos mais avancados em termos de tempo de execucao.
Ele funciona comparando cada par de elementos adjacentes na lista e trocando-os de posicao se estiverem na ordem errada.
Esse processo e repetido varias vezes ate que toda a lista esteja ordenada.
O funcionamento do algoritmo pode ser descrito em cinco passos simples:
1 - Comecando com o primeiro elemento (indice 0), compare-o com o proximo elemento (indice 1) na lista.
2 - Se o primeiro elemento for maior que o proximo elemento, troque-os de posicao.
3 - Continue percorrendo a lista, comparando elementos adjacentes e trocando-os de posicao se estiverem fora de ordem.
4 - Repita esse processo para cada par de elementos adjacentes na lista, percorrendo a lista varias vezes ate que nenhum elemento seja trocado
de posicao.
5 - Quando nenhum elemento for mais trocado de posicao, a lista estara ordenada


vetor desordenado: 63 63 50 85 63 95 28 86 6 21
Etapa 1 - 63 50 63 63 85 28 86 6 21 95
Etapa 2 - 50 63 63 63 28 85 6 21 86 95
Etapa 3 - 50 63 63 28 63 6 21 85 86 95
Etapa 4 - 50 63 28 63 6 21 63 85 86 95
Etapa 5 - 50 28 63 6 21 63 63 85 86 95
Etapa 6 - 28 50 6 21 63 63 63 85 86 95
Etapa 7 - 28 6 21 50 63 63 63 85 86 95
Etapa 8 - 6 21 28 50 63 63 63 85 86 95
Etapa 9 - 6 21 28 50 63 63 63 85 86 95
vetor ordenado: 6 21 28 50 63 63 63 85 86 95

Capítulo 8

Repositório de criptografia

Daniella Yuka Hirosue
Pedro Henrique Alves de Araujo Silva

8.1 Conceito do projeto

Este projeto foi realizado com a finalidade de apresentar aos calouros da UFSCar-So e entusiastas da computação, diferentes tipos de criptografias de mensagens ou palavras, demonstrando a funcionalidade de 4 tipos diferentes de cifras mais básicas e famosas e uma criada exclusivamente para esse repositório, para mostrar que criptografia não precisa ser complicado e qualquer um pode fazer a sua própria cifra.

8.2 Pré-requisitos e recursos utilizados

O grupo utilizou linguagem C para desenvolver a implementação completa do projeto, utilizando apenas as bibliotecas:

1. <stdio.h>
2. <stdlib.h>
3. <ctype.h>
4. <string.h>

Foram usados também como base para a a melhor compreensão do assunto um encontro do HackoonSpace sobre criptografia e pesquisas na internet.

8.3 Passo a Passo

Para o desenvolvimento do projeto, foi preciso a pesquisa para entender o funcionamento das seguintes cifras básicas e famosas:

- Cifra de César
- Cifra de Vigenere
- Cifra de Substituição

Além disso, também foi debatido sobre o funcionamento de uma quarta cifra original criada pelo grupo.

8.4 Execução

Para executar o projeto em seu terminal, vá até a pasta destino dos arquivos baixados e execute:

```
gcc main.c -o main
```

```
./main
```

8.5 Repositório

O repositório com os arquivos e demais informações sobre o projeto se encontra em:

`https://github.com/pedrohaas/RepoCriptografia`

8.6 Imagens/Screenshots

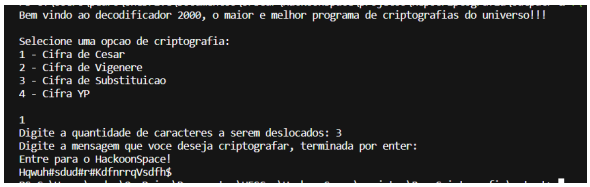


Figura 8.1: realizando cifra de césar

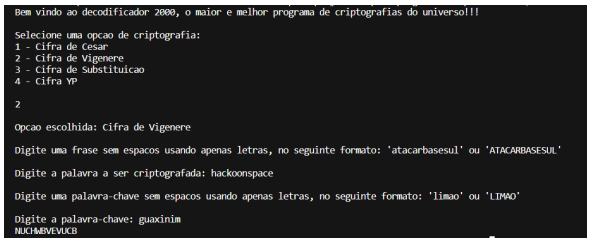


Figura 8.2: realizando cifra de vigenére

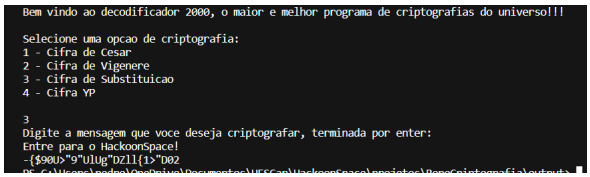


Figura 8.3: realizando cifra de substituição

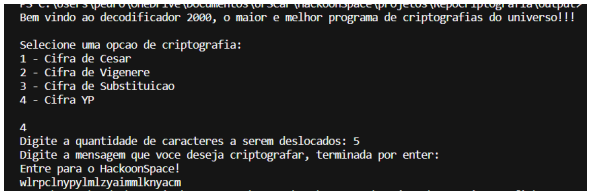


Figura 8.4: realizando cifra YP, de autoria própria

Capítulo 9

PCB Factory

Leonardo Morales

9.1 Conceito do projeto

Printed circuit boards (PCBs) ou Placas de Circuito Impresso são componentes essenciais no design de circuitos eletrônicos estes que são importantes na cultura maker. Este documento é um tutorial de uso e explicação do processo de fabricação de um PCB caseiro com qualidade industrial.

O PCB Factory é um pacote que consiste em 2 componentes, um é a estação de tratamento de luz ultravioleta, e outra é a estação de centrifugação de placas.

Pré-requisitos e recursos utilizados

O projeto não requer pré-requisitos além dos recursos utilizados e do conhecimento necessário para manusear os mesmos. Abaixo segue a lista de materiais usada para a criação do projeto:

- 2x Caixa de Plástico grande com tampa Removível
- 4x Esgana Gato
- 1x Secador de Cabelo Pequeno
- 1x Lâmpada de Luz Ultravioleta
- 1x Tabua de Madeira
- 1x Cabo de vassoura
- 1x Motor de Maquina de Costura
- Parafusos e Porcas
- 1x Soquete para lâmpada com fio
- 1x Lata de Tinta Preta

As ferramentas utilizadas foram

- Furadeira
- Serra
- Ferro de Solda
- Parafusadeira

9.2 Passo a passo

1. Foram cortados dois pedaços de tabua de madeira um pouco menor do que o espaço interno da caixa.
2. Uma vassoura foi serrada para obter 8 pezinhos de madeira para suspender as tabuas dentro das caixas. Fixamos ambos juntos e na caixa por meio de parafusos.
3. Um limitador foi colocado no pedal do motor da maquina de costura.
4. Foi furado um buraco no meio de umas das tabuas, assim possibilitando a fixação do motor de maquina de costura nessa mesma tábua.
5. O resto do buraco foi lixado até ser possível encaixá-lo novamente com folga no mesmo buraco que foi tirado.
6. Utilizando 4 parafusos, este circulo restante foi fixado ao motor de maquina de costura.

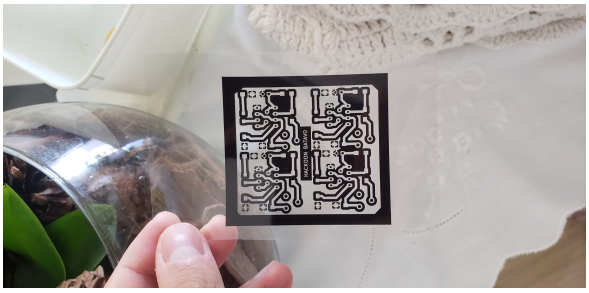
7. Dois buracos foram feitos na caixa para a fixação do secador de cabelo e a passagem dos cabos do motor.
8. Dois esgana gatos foram usados para fixar um dos lados da tampa a caixa, impedindo de abrir a caixa e deixar aberta.
9. Na outra caixa foi feito a mesma coisa com os pés de cabo de vassoura e tabua, contudo nenhum buraco foi feito na tabua.
10. Apenas um buraco foi feito na caixa para a fixação de um soquete de lampada.
11. Neste soquete foi fixado uma lampada de luz UV.
12. A caixa foi pintada de preto por dentro menos um pequeno buraco para indicar que a luz está funcionando.

9.3 Execução

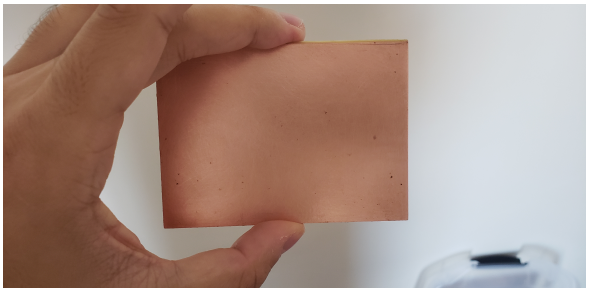
Lista de materiais para criação de uma PCB:

- Placa de Cobre do tamanho desejado para suportar o circuito.
- Modelo do circuito em qualquer software de design de circuitos eletrônicos.
- Tinta Fotossensível.
- Papel Translucido.
- Barrilha.
- Soda Caustica.
- Percloreto de Ferro, solução para corroer o cobre.
- Furador de PCB (Pode ser substituído por uma furadeira com broca pequena).

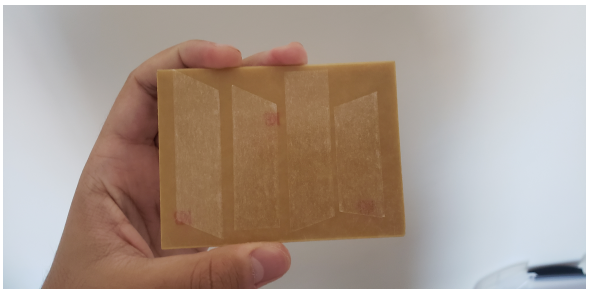
1. Imprima o design do circuito no papel translucido. Este papel sera utilizado como mascara no processo de cura. Geralmente a impressora deve possuir opção *Ink Jet*, o papel deve ser configurado como transparente ou então *photo glossy*. E a qualidade de impressão deve ser a maior possível (Melhor trocar tempo por qualidade neste caso);



2. Lixe a placa de cobre e a limpe bem;

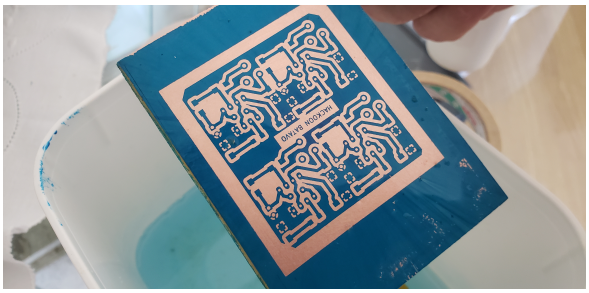


3. Insira a placa na estação de centrifugação. Coloque vários pedaços de fita dupla face na parte sem cobre da placa, limpe com álcool a base da centrifuga e então cole na base da estação de centrifugação;

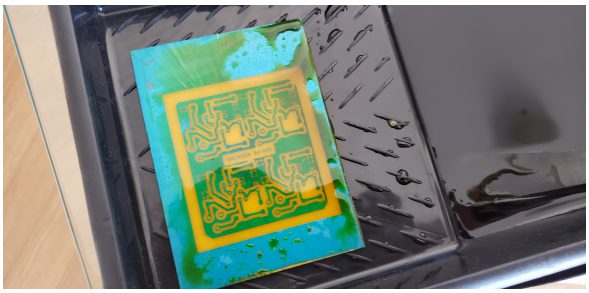


4. Aplique uma camada de tinta fotossensível na placa de cobre, procure deixar uma maior concentração de tinta no centro da placa;

5. Feche e ligue a centrifuga por 90 segundos;
6. Ligue o secador da estação de centrifugação e deixe por 15 minutos, após este tempo deixe mais 10 minutos com a tampa aberta sem o secador ligado e repita este processo inteiro mais uma vez;
7. Retire da estação de centrifugação;
8. Coloque a mascara impressa em cima da placa coberta com a tinta, garanta que esta corretamente alinhada e fazendo contato direto com a superfície pintada; Para obter mais precisão pode ser inserida uma camada de papel transparente diretamente em cima da tinta, e então a camada com o máscara assim possibilitando manusear a máscara para alinhar perfeitamente;
9. Insira a placa na estação de luz ultravioleta pelo tempo especificado na tinta fotossensível. Ou no geral 4 minutos;
10. Misture 200 ml de água com meia colher de Barrilha;
11. Insira a placa sem a mascara na solução misturada no passo anterior. Com um rolo de tinta pequeno, esfregue a placa gentilmente para retirar a tinta que não foi exposta a luz ultra violeta, deixando a superfície de cobre exposta;

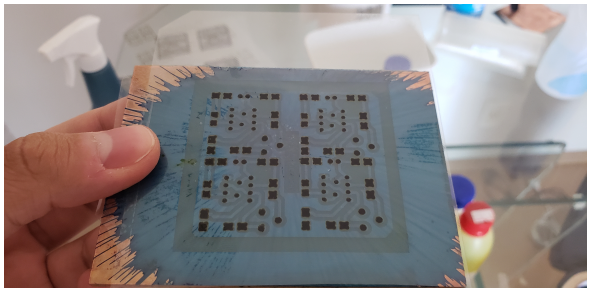


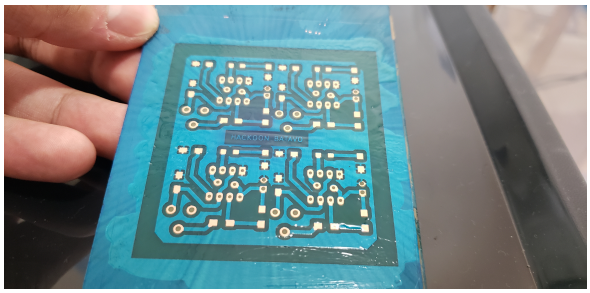
12. Reinsira a placa na estação de cura ultravioleta por mais 2 minutos;
13. Faça o mesmo processo do passo 11 porem com o percloreto de ferro. O percloreto de ferro vai corroer todo cobre exposto. Deixe no banho de percloreto por 10 minutos e mexa o percloreto durante o processo;
14. Lave a placa para limpar o percloreto de ferro com cuidado, **procure não jogar percloreto em nenhuma tubulação!** O restante do percloreto pode ser colocado em uma garrafa e reutilizado na próxima PCB;



15. Ferva 200ml de água e misture com 2 colheres de soda caustica. **Cuidado, não toque na mistura.** Insira a placa nesta solução. Utilizando uma colher ou outra ferramenta, manuseie a placa para descolar a tinta ainda presente na mesma;
16. Com cuidado limpe novamente a placa com água;

17. Faça a impressão de outra mascara, porem desta vez apenas com as ilhas de solda e com essa mascara alinhada as ilhas já presentes na placa de cobre repita os passos 2 a 12 porem com a tinta fotossensível para mascara, e a cura na estação UV deve ser de apenas 30 segundos;





18. Corte e lixe a placa ate ficar do tamanho do circuito que foi impresso;
19. Utilizando um furador de PCB ou uma furadeira com a broca pequena, fure os buracos onde os componentes serão inseridos no caso de componentes com pontos de solda perfurados. No caso de componentes montados na superfície não ha necessidade de furacão;
20. Teste a placa utilizando um multímetro e o modo de teste de continuidade;
21. Dilua Breu em álcool isopropílico (No projeto entregue foi fornecido um concentrado de breu, dilua o concentrado em 3 partes de álcool para 1 de breu; Ou então faça uma mistura de 10 partes de álcool para 1 de breu em pó) e então pingue nas ilhas de solda com um cotonete (ou passe na placa inteira, criando uma camada de verniz), isso ira formar uma camada anticorrosão também nas ilhas, que ira derreter quando for aplicado calor para solda;
22. Insira os componentes em seus lugares;
23. Solde os componentes;

9.4 Bugs/problemas conhecidos

O projeto tem alguns riscos com materiais químicos e também com a centrifuga, portanto é recomendado muito cuidado a operar. Além disso o maior ponto de cuidado é a operação da centrifuga. Se a vibração da centrifuga for suficiente para mover a caixa, diminua a pressão no pedal.

9.5 Repositório

O repositório com os arquivos e demais informações sobre o projeto se encontra em:
`https://github.com/LeozeraVal/pcb-factory`

9.6 Bibliografia

[1] Estudo do blog **ezcontents**.

[2] Referência ao vídeo **How to make PCB at home using photoresist Dry Film.**

[3] Referência ao vídeo **How to make PCB using Photoresist Dry Film**.

Capítulo 10

Integração Arduino-Discord

Marcus Vinícius Natrielli Garcia

10.1 Conceito do projeto

Este projeto é uma prova de conceito que visa mostrar como possibilitar a comunicação entre uma placa *Arduíno* e um *bot de Discord* que utiliza o wrapper *Discord.js*.

Tanto o *Arduíno* quanto o conceito de *bots* de forma geral estão muito ligados com usos para automação e modernização da tarefas. Entretanto, o primeiro é mais focado em hardware e interações com o mundo real, enquanto o segundo é mais aplicado à software e conhecido no mundo virtual.

Dessa forma, este projeto se propõe a integrar ambos os conceitos, estabelecendo o envio de dados entre *Arduíno* e *bot* e vice-versa.

10.2 Comandos e funções

Do *bot* para o Arduíno:

- Usar comando de */pisca-pisca* para fazer o LED piscar algumas vezes.
- Usar comando de */musica* para começar a tocar uma música que poderá ter seu volume alterado pelo Arduíno.

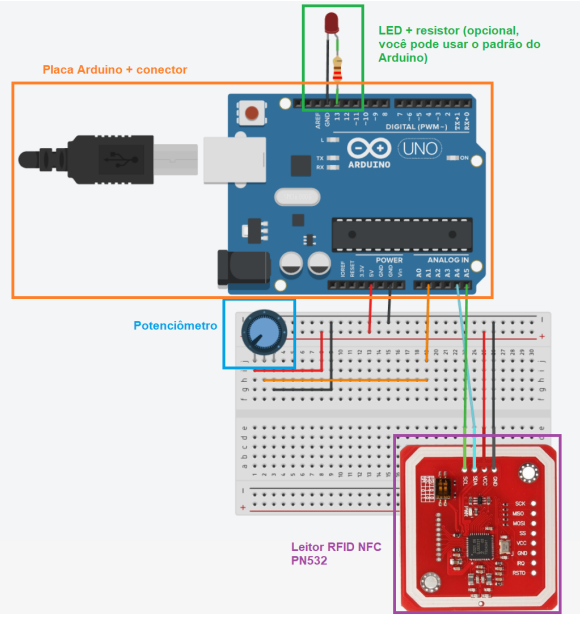
Do Arduíno para o *bot*:

- Usar uma tag NFC/RFID para fazer o LED piscar.
- Usar uma tag NFC/RFID para enviar uma mensagem de acesso autorizado para o *Discord*.
- Alterar o volume de uma música sendo tocada no *Discord* usando o potenciômetro do Arduíno.

10.3 Pré-requisitos e recursos utilizados

Os recursos utilizados neste projeto foram:

- **Hardware**
  1. **Arduíno Nano** (outras placas podem ser utilizadas, utilizadas as devidas adaptações)
  2. **Potenciômetro linear de 10KΩ**, para controle de volume
  3. **Leitor RFID NFC PN532** (ou similar), para leitura de tags NFC ou similares
  4. **Jumpers/cabos** para fazer a conexão entre os componentes
  5. Cabo para conectar o Arduíno em um computador
  6. (Opcional) LED e resistor para funções de pisca-pisca (geralmente as placas Arduíno já possuem um LED embutido. É este que estou usando no projeto mas você pode usar o seu próprio)
  7. Imagem do circuito comentado:



- **Software**

    1. **Node.js** (linguagem padrão do *bot* e meio pelo qual se estabelece a conexão serial com o Arduíno. O projeto foi testado na versão 17)
    2. **NPM** (gerenciador de dependências do Node)
    3. **Arduino IDE 2.0** (para fazer *upload* do código para o Arduíno)
    4. **Discord.js** (para conexão com o *Discord*, incluso nas dependências do NPM)
    5. **PN532_I2C.h**, para usar o leitor NFC com I2C. Para usar com outras formas de comunicação, baixar a biblioteca correta e fazer as adaptações necessárias

10.4 Passo a passo

1. Escrevi o código para o funcionamento do *bot* de *Discord*
    - Baseado em experiências passadas, esta parte foi relativamente fácil. Decidi quais seriam as funções principais do *bot* para comprovar que pude estabelecer a comunicação entre *Arduino-Node.js*
2. Comprei/encontrei os componentes do circuito, como a placa Arduíno e o leitor NFC
    - Dica: Diferentemente de software, o hardware pode vir com defeitos de fábrica que tornem necessário testar outras peças ou fazer trocas. Logo, quando algo não funciona, nem sempre foi um erro seu. Verifique se o componente está queimado ou se funciona em outro circuito mais simples
3. Pesquisei materiais na internet sobre comunicação serial do Arduíno com código *Node.js*
4. Instalei e testei as dependências do *Node.js* no meu código
5. Realizei testes de envio e recebimento de dados entre ambas os lados
6. Implementei os últimos detalhes necessários no código para que tudo funcionasse bem

10.5 Instalação

Variáveis de ambiente

Este arquivo usa variáveis de ambiente que deverão ser definidas antes da execução do programa. As variáveis usadas são:

- **BOT_TOKEN:** *token* do seu *bot* de Discord (pode ser gerado no painel do desenvolvedor)
- **CHANNEL_ID:** ID de canal do Discord onde o *bot* enviará mensagens
- **GUILD_ID:** ID de servidor do Discord que o *bot* usará de base para se conectar aos canais de voz
- **AUTHORIZED_TAG:** valor de tag NFC/RFID para enviar mensagem de acesso autorizado
- **BLINK_TAG:** valor de tag NFC/RFID para piscar o LED
- **CLIENT_ID:** ID do *bot* de Discord (para registrar os *slash commands*)
- **ARDUINO_PORT:** valor da porta do computador a qual o Arduino se conectou

Programação

É recomendável fazer esta parte primeiro, começando pelo Arduino.

Você precisará do valor da porta do Arduino para configurar o Node, e você precisará que o código seja enviado para o Arduino para ligar o circuito por completo.

- **Parte do Arduino**

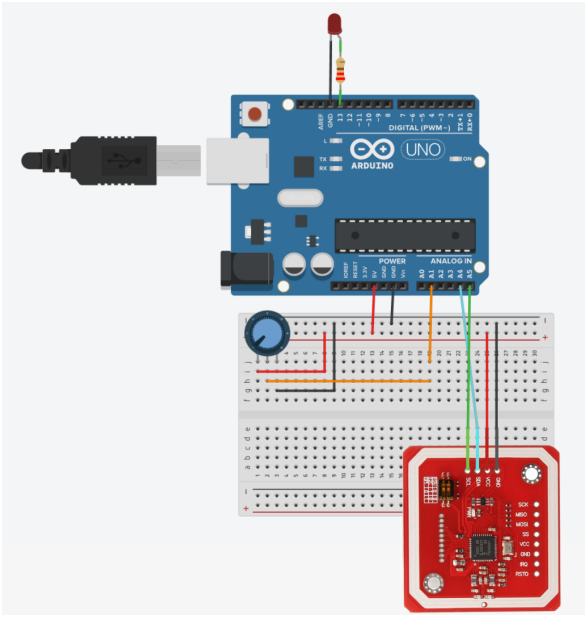
    1. Abra a IDE do Arduino, com o código do arquivo *Sensor.ino*
    2. Conecte a sua placa Arduino ao computador
    3. Na IDE, selecione o modelo correto de Arduino e a porta na qual se encontra a placa
    4. Realize a compilação e envio do código para a placa

- **Parte do Node.js**

    1. Baixe o projeto do repositório
    2. Execute o comando *npm install* para instalar as dependências
    3. Crie um arquivo *.env* com as variáveis de ambiente necessárias, baseado em *.env.template*
    4. Execute o comando *npm run start* para testar o programa

Montagem do circuito

Imagem do circuito completo:



É altamente recomendado você montar o circuito com a energia desligada e sem qualquer cabo conectado entre sua placa Arduino e o computador.

A partir do momento em que o código for enviado para o Arduino e o circuito montado e conectado ao computador, a aplicação deverá funcionar.

Arquivos

Além disso, para usar a funcionalidade de música, é necessário criar uma pasta */media* com um arquivo *music.ogg*. Fique a vontade para escolher a música que você quiser.

Recomendação: Em *bots* mais complexos, é possível tornar esse sistema dinâmico para pegar sons e músicas de algum outro lugar, por exemplo.

10.6 Execução

Para executar o projeto, é necessário:

- Ter o circuito corretamente montado
- Ter feito *upload* do código para a placa Arduino

- Estar rodando a aplicação Node.js, sem erros
  - Se algum erro ocorrer, provavelmente:
    * Alguma variável de ambiente não foi setada
    * Alguma variável de ambiente está com valor inválido
    * A comunicação entre Arduino-PC ou entre Arduino-Node não foi estabelecida
    * Não foi possível conectar o código Node.js com o seu *bot* de Discord
- Ter em mãos as tags NFC/RFID setadas nas variáveis de ambiente não foi setada
- Estar conectado no servidor de Discord setado nas variáveis de ambiente

Para isso, basta seguir os passos descritos na seção de **Instalação**.

Observação: também há a possibilidade de algum componente do circuito estar queimado ou não conectado direito. Tome cuidado na hora de montar o circuito!

O comando para executar a aplicação Node.js é *npm run start* (ou derivado disso).

10.7 Bugs/problemas conhecidos

Atualmente, o projeto se encontra com algumas limitações:

- É necessário ter o Arduino conectado (via cabo) ao mesmo computador que está rodando o *bot* de Discord
- É necessário mudar manualmente no arquivo *.env* o valor da porta na qual o Arduino está conectado (que pode mudar dependendo da entrada em que você conecta a placa)

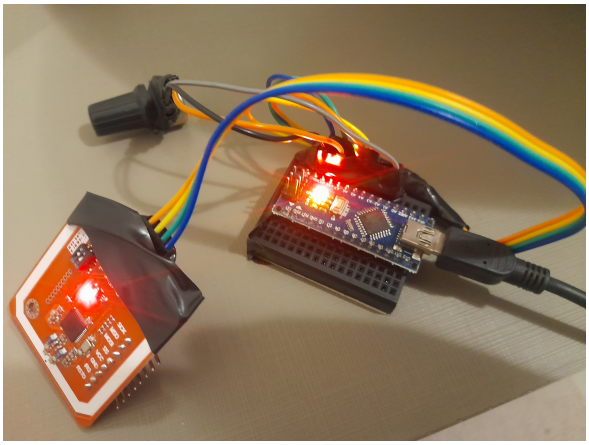
É esperado que em futuras versões, a conexão entre a placa e o código seja feita de forma sem fio, por Bluetooth ou Wi-fi. Isso resolveria ambos os problemas apresentados, mas necessitaria de diversas alterações no código e no circuito.

10.8 Repositório

O repositório com os arquivos e demais informações sobre o projeto se encontra em:

`https://github.com/InfiniteMarcus/Arduino-Discord-Integration`

10.9 Imagens/screenshots



10.10 Bibliografia

[1] Artigo **Interfacing PN532 NFC RFID Module with Arduino.**

[2] Artigo **Lab: Serial Input to an Arduino from Node.js.**

[3] Artigo **Arduino and Node.js via Serial port.**

[4] Documentação **Discord.js.**

Capítulo 11

MultiVXRemover

Rafael Gimenez Barbeta

Victor Motta

11.1 Conceito do projeto

Projeto de mini-antívirus, que consegue detectar assinaturas de malwares conhecidos assim como colocá-los em *quarentena*. Foi feito inteiramente em python com ajuda da biblioteca de regras YARA. Adicionalmente, inclui uma breve pesquisa sobre o vírus de computador conhecido como Neshta, com sua análise dinâmica e estática.

Todos os softwares desenvolvidos contém padrões exclusivos, que permitem indentificá-los em meio a tantos outros. O mesmo acontece com malwares. Um dos mecanismos mais úteis dos antivírus modernos é justamente a detecção por assinatura, que consiste em catalogar diversos desses "padrões" de vírus em uma base de dados e compará-los com os arquivos presentes no computador do usuário. Um *match* provavelmente significa que aquele arquivo é um malware já conhecido e que portando deve ser isolado pela segurança do cliente.

Dois tipos de assinatura são utilizados para detecção de software maliciosos nesse projeto:

- **Regras YARA:** Consiste em um conjunto de expressões regulares aplicadas ao binário. Consegue dar *match* tanto em bytes *raw*, strings de texto e combinar os diferentes *matches* que ocorreram com expressões lógicas. É o mecanismo principal de detecção. Consegue detectar malwares e possíveis variantes desse software malicioso.
- **Hash MD5:** O hash é como um *somatório do conteúdo de um arquivo*. A função hash, no caso o MD5, recebe um arquivo de tamanho arbitrário e produz uma string fixa que é potencialmente exclusiva para esse arquivo. É bastante inflexível, uma mínima alteração muda completamente o hash, sendo muito específica para identificar um software único. A vantagem é que apenas softwares já catalogados com esse hash serão efetivamente detectados, com menor probabilidade de um *match* errôneo.

A interface do programa é inteiramente textual. Após a execução, será fornecido um terminal para inserção dos comandos, que seguem o padrão: *nome_comando* < *parâmetro* >

11.2 Pré-requisitos e recursos utilizados

O *MultiVXRemover* foi feito inteiramente em *python*, sendo necessário um interpretador da linguagem para poder executar o programa. Foram utilizadas as seguintes bibliotecas:

- **OS**, para operar com arquivos, padrão do python
- **lib-yara**, biblioteca que da suporte as regras yara

- **yara-python**, integração da biblioteca com python
- **sqlite3**, para construção do BD de hashes md5, padrão do python

O projeto inclui um set de regras YARA obtidos da *Reversing LABS*. O projeto original pode ser encontrado no link:

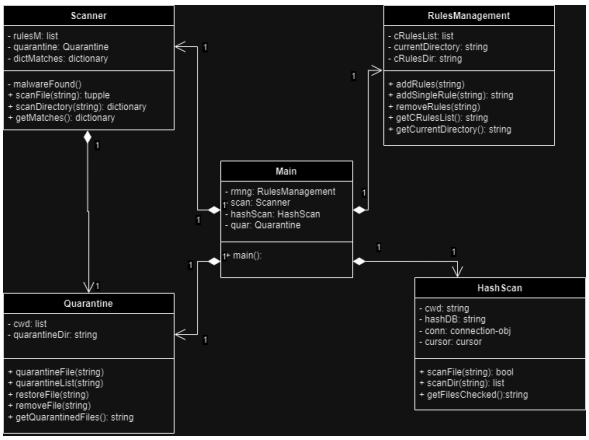
`https://github.com/reversinglabs/reversinglabs-yara-rules`

Além disso, inclui um *scraper* escrito em bash *downloader.sh* para hashes MD5 do site VirusShare. Há uma coleção **gigantesca** de assinaturas de vírus coletadas da internet. O download de todos os arquivos pode demorar algum tempo, mas se desejar, adicione seus próprios *hashes* para teste.

11.3 Passo a passo

Para a realizar o projeto, fizemos a seguinte sequência de etapas:

1. **Pesquisa inicial sobre o funcionamento de malwares e sua detecção:** Aqui entra nosso estudo de caso sobre o vírus *Neshta*. O vírus em particular continha uma string bastante única que o identificava em relação a outros softwares. Foi aqui que descobrimos o mecanismo de detecção por assinatura dos antivírus. As análises podem ser vistas na pasta *Virus.Win32.Neshta.A Malware analysis*.
2. **Implementação:** Seria necessário alguma biblioteca que permitisse executar ações de *match* contra arquivos. No ramo de segurança, a biblioteca YARA é bastante utilizada para esse fim.
3. **Projeto do programa:** Seguimos o paradigma de programação orientado a objetos. Para isso, criamos um programa principal *Main.py* responsável apenas por comunicação com usuário. O *Main* integra os outros módulos. Entre eles temos:
    - **RulesManagement**, responsável por gerenciar as regras YARA.
    - **Scanner**, que analisa um diretório e retorna uma *associação* entre caminho do arquivo com o nome técnico do malware detectado (*se houver*).
    - **Quarantine**, que remove um arquivo e o transforma em *base64*, de forma que ele não seja executável.
    - **HashScan**, que faz um scan alternativo com hashes. Utiliza o banco de dados *sqlite* tentando encontrar um *match*.
    - Diagrama de classes abaixo:



4. **Codificação:** Uma vez definida a estrutura passamos para a implementação, utilizamos a documentação oficial do YARA e do python para nos auxiliar.

11.4 Instalação

Alguns preparativos precisam ser feitos para que seja possível executar o programa. Os passos serão dados para o Linux, supõe-se que esse aplicativo será executado em um ambiente seguro, desconectado da internet e em uma máquina virtual.

1. Instale o *yara*:

```
sudo apt install yara
```

2. Instale o gerenciador de pacotes do python, o *pip*:

```
sudo apt install python3-pip
```

3. Instale o módulo *yara* para o pyhton:

```
pip install yara-python
```

4. Execute o programa para atestar se tudo está certo:

```
python3 Main.py
```

OBS: Talvez seja necessário instalar *libssl-dev* primeiro, em caso de erro. Baixe o conjunto de hashes do *VirusShare*, basta executar:

```
./donwloader.sh
```

OBS: Esta etapa pode demorar cerca de 30 minutos. Interrompa a qualquer momento o download se não quiser baixar todos os *hashes md5*. Monte o banco de dados *sqlite*. Esta etapa é bem mais rápida que a primeira:

```
python3 hashDBBuilder.py
```

11.5 Execução

A execução é bastante direta, apenas digite:

```
python3 Main.py
```

E o programa estará em execução. Adicione as regras incluidas no app:

```
addRules rawRules
```

O programa suporta adição constante de novas regras. Para fins de teste, vamos inserir uma regra para detectar o falso vírus EICAR.

Em outro terminal, insira essa comando para gerar o *virus*:

```
mkdir teste
echo 'X5O!P%@AP[4\PZX54(P^)7CC)
    7}$EICAR-STANDARD-ANTIVIRUS-
    TEST-FILE!$H+H*' > ./teste/
    eicar
```

E escanear com:

```
scan teste
```

O *vírus* será neutralizado e colocado em quarentena. No caso, ele simplesmente é codificado para base64, um formato puramente textual e não executável. Outras informações são adicionadas para permitir sua restauração no mesmo local onde foi deletado.

Veja sua quarentena com:

```
viewQuarantine
```

Restaure o eicar:

```
restore eicar
```

O scan de hash funciona da mesma forma. Atente-se com uma demora extra para verificar os arquivos, uma vez que será necessário comparar o hash com uma grande base de dados.

11.6 Bugs/problemas conhecidos

Por ter sua detecção baseado apenas em assinaturas, o *MultiVXRemover* não é capaz de detectar os chamados malwares metamórficos/polimórficos ou ameaças recentes. De forma resumida, malwares que mudam sua forma (sua assinatura) *espontaneamente* ou que acabaram de ser criados não serão detectados por regras estáticas, que ficam muito limitadas aos softwares que as deram origem. Soluções de antivírus modernas incluem análise heurística e em tempo real da atividade de cada software no PC, se concentrando mais na ação do programa do que no seu código.

A compilação das regras YARA geralmente será rápida, porém, o download e criação do banco de dados SQL dos hashes pode levar algum tempo, cerca de 30 minutos. A detecção com hashes também sofre do mesmo problema, demorando relativamente bastante tempo para comparação do hash de um arquivo com o banco de dados. Tentativas de otimização foram feitas para acelerar o processo, sendo que a organização em banco de dados *sqlite* foi a mais promissora.

A organização em forma de classes possui um alto *acoplamento* entre as classes, o que torna o código mais difícil de ser expandido futuramente.

A quarentena utilizando o scan com regra yara está um pouco confusa. A quarentena é utilizada diretamente pelo *Scanner*, mas é o *Main* que deveria fazer isso.

11.7 Demais observações

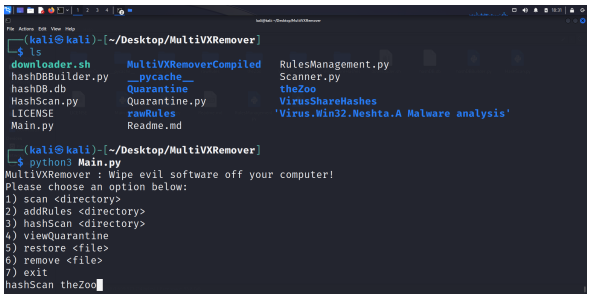
Encoraja-se fortemente o uso dessa ferramenta em um ambiente controlado e seguro como uma máquina virtual. Esse projeto NÃO substitui um antivírus profissional. Os criadores não serão responsáveis por quaisquer incidentes que ocorram devido ao uso desse software.

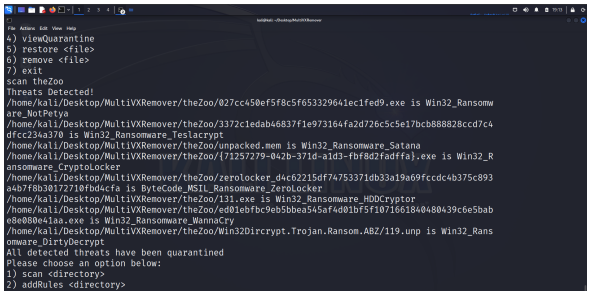
11.8 Repositório

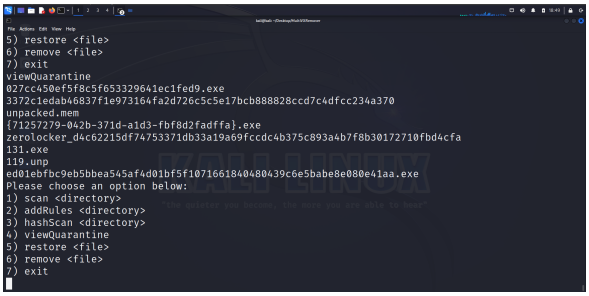
O repositório com os arquivos e demais informações sobre o projeto se encontra em:

`https://github.com/rafaelbarbeta/MultiVXRemover`

11.9 Imagens/screenshots







11.10 Bibliografia

[1] Documentação **Python.**

[2] Documentação **YARA.**

[3] Utilização dos hashes do **VirusShare.**

Capítulo 12

Kindlepad

Lauan dos Santos Souza

Maria Anita de Moura

12.1 Conceito do projeto

Este projeto tem como objetivo permitir que os leitores acessem e leiam artigos, livros, mangás, entre outros, de forma offline, a partir da extração dessas histórias de diferentes origens.

Para isso ele foi projetado para funcionar com módulos, onde cada módulo terá as funcionalidades necessárias para extração do conteúdo.

Nesse repositório se encontra disponível para demonstração o módulo do MangaLivre.net, uma plataforma online que oferece mangás com anúncios aos leitores. A proposta é extrair as imagens dos mangás, baixá-las e reuni-las em um único arquivo, facilitando a leitura em aplicativos dedicados à leitura de quadrinhos mesmo sem conexão com a internet.

Funcionamento:

O projeto consiste em um sistema que realiza as seguintes etapas:

- Acesso ao mangalivre.net: O sistema acessa o site MangaLivre.net para obter os mangás disponíveis para leitura online.
- Extração de imagens: A partir dos mangás selecionados, o sistema extrai as imagens dos capítulos, garantindo que todo o conteúdo visual seja capturado com fidelidade.
- Download das imagens: Após a extração, as imagens são baixadas para o computador do usuário, tornando-as acessíveis mesmo em ambientes sem conexão à internet.
- Geração de arquivos próprios para leitura: Todas as imagens baixadas são agrupadas em arquivos CBR (Comic Book Reader), separados por capítulo e seguindo a ordem original do mangá. Dessa forma, o usuário pode ler o mangá de forma contínua, sem interrupções.

Benefícios:

Este projeto traz diversos benefícios para leitores que precisam acessar um conteúdo de forma offline, em especial os entusiastas de mangás, graças ao módulo disponibilizado, como:

- Leitura offline: Permite a leitura mesmo em locais sem acesso à internet, tornando-se uma solução ideal para viagens ou locais remotos.
- Organização: As informações extraídas são agrupadas em um único arquivo, facilitando a organização e evitando a necessidade de ter que carregar partes de um capitulo dinamicamente.
- Acessibilidade: Usuários que não possuem conexão à internet podem desfrutar de seus mangás favoritos sem restrições.
- Economia de dados: Uma vez que o mangá é baixado, não é mais necessário utilizar dados móveis ou conexão Wi-Fi para a leitura.

- Melhor experiência: O conteúdo gerado vem livre de anúncios e interrupções, além de possibilitar o compartilhamento e-readers.

Conclusão:

Em suma, esse projeto é uma solução prática e útil para leitores que buscam maior flexibilidade e comodidade em sua experiência de leitura.

12.2 Pré-requisitos e recursos utilizados

Para o desenvolvimento do projeto foi utilizado Node.js com as seguintes bibliotecas:

- puppeeteer: Utilizado para conectar com um browser chromium e simular a interação do usuário com o browser.
- axios: Utilizado para o fazer requisições http.
- archiver: Utilizado para agrupar os arquivos obtidos e gerar um único arquivo cbr.

Além disso, foi utilizadas as bibliotecas e funções disponibilizadas pelo node para manipulação de arquivos e buscas por padrões com expressões regulares.

12.3 Passo a passo

Para a implementação do projeto, primeiro é necessário entender o funcionamento do site mangalivre.net:

É possível resumir as tarefas necessárias para ler um mangá em alguns passos: Acessar a plataforma e escolher um título; Selecionar o mangá e ser direcionado a página com a lista dos capítulos; Escolher algum capítulo para leitura; Navegar entre as páginas do capítulo até o final, onde há um link para o próximo.

Assim para o desenvolvimento do código podemos abstrair alguns passos como:

- Fazer a requisição HTTP para o site
- Extrair do html do site os links para cada capítulo
- Extrair do html do site as imagens do mangá
- Extrair do html do site o link para a próxima página do capítulo
- Dentro de cada capítulo de mangá, a diferenciação da uma página para outra se dá pelo link, como por exemplo:
  `https://mangalivre.net/ler/one-piece/online/483101/1089#/!page1`
  `https://mangalivre.net/ler/one-piece/online/483101/1089#/!page2`

No exemplo acima podemos diferenciar uma página da outra apenas pelo sufixo do link, então para acessarmos todas as páginas do capítulo, bastaria iterarmos sobre esse sufixo.

- Já a diferenciação de capítulos, não se dá da mesma forma:
  `https://mangalivre.net/ler/one-piece/online/479971/1088#/!page0`
  `https://mangalivre.net/ler/one-piece/online/483101/1089#/!page0`

No exemplo acima temos os links para os capítulos 1088 e 1089 do mangá One Piece, e como pode ser observado, não há um padrão para o link, de forma que para acessarmos todos os capítulos uma simples iteração sobre o link não seria possível

Devido a tal problema, para acessarmos o link de cada capítulo, teríamos que acessar o link da página do mangá e então acessar os links para cada capítulo que se encontra então no html.

Finalizando, após a extração das imagens, foi necessário apenas colocar todas as páginas dentro de um diretório de forma ordenada e compactar a pasta.

Dessa forma foram implementado o arquivo *index.js* na pasta *src/modules/mangalivre*, para fazer a extração das imagens do site e o arquivo *index.js* da pasta *src/utils* para fazer a compactação do arquivo.

12.4 Instalação

Para rodar o projeto, é necessário:

- Instalar o Node.js (versão v14 ou superior) e o npm/yarn:

  - O npm vem instalado por padrão junto com o Node.js

- Realizar o clone do repositório
- Instalar as dependências do projeto

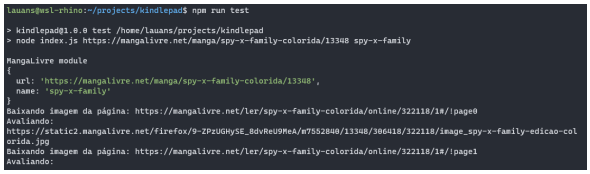
```
npm i
```

Com isso o projeto estará instalado e pronto para ser executado.

12.5 Execução

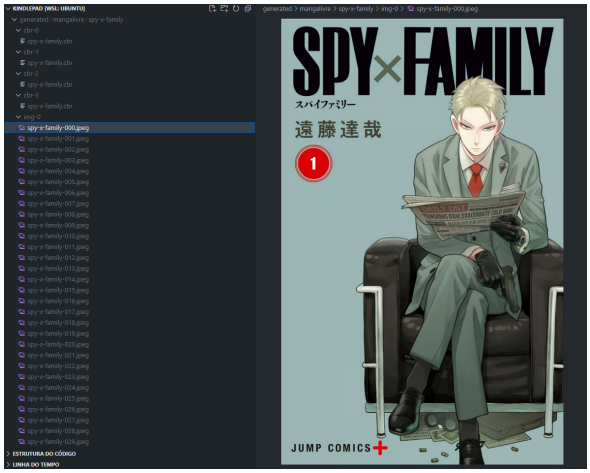
Para rodar o projeto, baixando um título de demonstração, basta executar o comando:

```
npm run test
```



Esse comando irá executar o módulo do Mangalivre.net para baixar um manga.

Ele ficará disponível em *generated/mangalivre/spy-family*



Para baixar outro título, é necessário rodar rodar o comando abaixo atualizando os valores:

```
node index.js <URL_DO_MANGA> <
    NOME_DO_ARQUIVO_DE_SAIDA>
// Exemplo:
// node index.js https://
    mangalivre.net/manga/spy-x-
    family-colorida/13348 spy-x-
    family
```

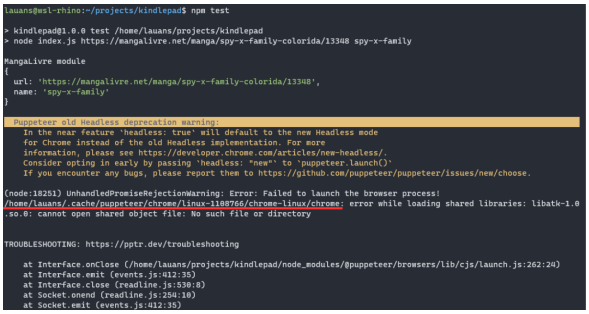
Possíveis complicações no Linux

Bibliotecas para executar o chromium não encontradas

O módulo disponível nesse projeto (Mangalivre.net) utiliza a biblioteca puppeteer para instanciar um browser (Chromium) e assim conseguir baixar o manga.

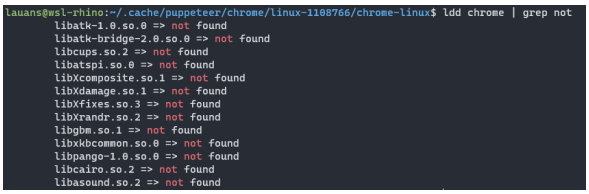
Para o browser ser executado corretamente, ele depende de algumas bibliotecas no seu sistema.

Se ao tentar executar o projeto você se deparar com um erro semelhante ao exemplificado na imagem, será preciso realizar a instalação dessas bibliotecas.



As bibliotecas necessárias podem ser consultadas ao executar o seguinte comando na pasta de instalação do navegador (por padrão é instalado em uma pasta em */.cache/puppeteer/**, destacamos em vermelho na imagem acima o local onde aparece o caminho completo)

```
ldd chrome | grep not
```



Para instalar esses pacotes, execute

```
sudo apt update -y
sudo apt install ${
    bibliotecas_faltantes} -y
```

Uma lista completa das dependências também podem ser consultado na documentação do puppeteer

12.6 Bugs/problemas conhecidos

O projeto possui uma limitação de que alguns mangás do site, não possuem mesma padronização de nome, e portanto, o regex implementado para obter as imagens e links do sites, pode não funcionar para determinado título, além disso, caso haja alguma atualização no site mangalivre, é possível que o projeto para de funcionar pois não está configurado para essa novas configurações do site.

12.7 Imagens

Leitura de um mangá pelo leitor Cover

12.8 Repositório

O repositório com os arquivos e demais informações sobre o projeto se encontra em:

`https://github.com/lauanS/kindlepad`

Capítulo 13

Luof - Gerenciador de links

Murillo Justino dos Santos

13.1 Conceito do projeto

Este projeto foi criado com a intenção de ser outro meio de guardar links de páginas web, assim como no Favoritos de um browser, pelo terminal linux de modo hierárquico, sendo fácil de adicionar e gerenciá-los. Possibilitando armazenar links de páginas visitadas em diferentes navegadores em um lugar só, independente da quantidade, deixando apenas os mais importantes e recorrentes no Favoritos do browser usado.

13.2 Atualizações

- 12/05/2023:
  - Licença adicionada.
- 02/05/2023:
  - Agora pode-se passar argumentos na linha de comando.
- 24/04/2023:
  - make reinstall adicionado.
  - Agora a categoria a ser visulizada pode ser passada direto na linha de comandos do terminal com os comandos -lc e -lcs.
  - Atualizações criado.

Para mais detalhes ver updates.txt

13.3 Pré-requisitos e recursos utilizados

Em geral foi utilizado a Linguagem C para desenvolver este projeto, possuindo também um arquivo makefile para ajudar na compilação do programa. Para compila-lo usando o makefile é necessário ter o programa make e o compilador gcc instalados no sistema, caso contrário, qualquer compilador C deve funcionar.

13.4 Passo a Passo

1. Definição de quais seriam as estruturas usadas no código, e os padrões de nomenclaturas.
2. Definido como o banco de dados seria organizado, e criado a função responsável por iniciá-lo. Foi definido então que o banco de dados seria implementado usando arquivos e as funções da biblioteca padrão C para manipulá-los.
3. Implementado funções para adicionar favoritos e categorias, e depois para remover favoritos e categorias.
4. Implementado funções para listar favoritos de uma categoria, e depois para listar todas categorias e favoritos em árvore.
5. Implementado funções para modificar favoritos e categorias.
6. Implementado funções de criar e restaurar backup e de exportar favoritos.

7. Implementado a função help, a função import e feito ajustes gerais no código.

13.5 Instalação

Primeiramente deve-se clonar o repositório do luof e entrar em seu diretório, isso pode ser feito com os comandos a seguir:

```
git clone https://github.com/
    mutannejs/Luof.git
cd Luof
```

Usando make

Para instalar o programa usando o makefile, dentro da pasta do projeto, basta rodar o comando:

```
make install
```

Ao utilizar o comando acima será requisitado a senha de seu usuário. Este comando compila o programa e move o arquivo binário para a basta /usr/bin, para que possa executá-lo em qualquer diretório dentro do sistema sem a necessidade de informar seu caminho completo, além de criar o banco de dados na home do usuário. Caso não queira usar o luof em outros diretórios, deve-se rodar o comando "make compile" em conjunto com o esquema descrito nas linhas 14 a 19 no arquivo dbluof.c (se ele não for executado, ocorrerá erro na execução do programa, o esquema deve ser feito antes de usar o comando make). Desse modo não será criado nenhum arquivo em outros diretórios, nem será requisitado a senha do usuário.

Sem usar make

Para compilar o programa sem o makefile, só com o gcc, basta entrar na pasta do projeto e rodar os comandos:

```
gcc -o luof main.c add-remove.c
    list.c modify.c dbluof.c
    dbcat.c modulos.c help.c
    backup.c import-export.c
    lista-iterador.c teste.c
```

Para poder executá-lo em qualquer diretório dentro do sistema sem a necessidade de informar seu caminho completo, após rodar o comando acima será necessário usar o comando:

```
sudo mv luof /usr/bin
```

Caso não queira usar o luof em outros diretórios, antes de compilar o programa deve ser feito o esquema descrito nas linhas 14 a 19 no arquivo dbluof.c (se ele não for executado, ocorrerá erro na execução do programa). Desse modo não será criado nenhum arquivo em outros diretórios, nem será requisitado a senha do usuário.

Se não foi usado make install

Na primeira execução do programa será necessário criar um novo banco de dados, basta teclar 's' seguido de enter para responder que o novo banco deve ser criado. A seguir há uma imagem que descreve este cenário, usando o modo de instalação sem o make, será usado as funções --import" e -lt" apenas para melhorar o exemplo:

Figura 13.1: Criando um novo banco de dados

13.6 Desinstalando

Caso tenha sido usado o comando "make install" para instalar o luof, para desinstalá-lo basta usar o comando:

```
make desinstall
```

Este comando remove o arquivo binário da pasta */usr/bin* e exclui o banco de dados da home do usuário que executou o comando.

13.7 Execução

Neste tópico será mostrado algumas das principais funções que o programa nos permite utilizar. Primeiramente será apresentado uma imagem contendo um texto explicando o básico para utilizar o programa, ele poderá ser visto utilizando o comando "luof –help" após o luof ter sido instalado em seu sistema:

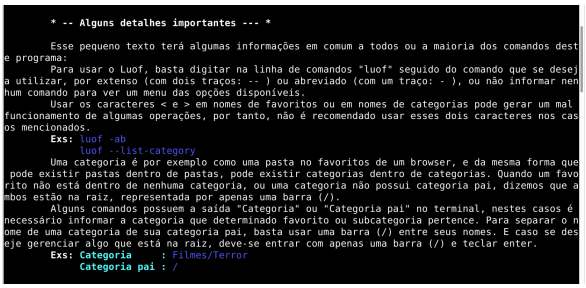


Figura 13.2: painel informativo do luof

Além desse texto explicativo, a função ëlp" mostra todas as opções possíveis de usar no luof, incluindo funções não apresentadas neste tópico.

Após instalado o programa, será adicionado um primeiro favorito nele. O favorito será o repositório deste projeto, e será inserido na raiz (até então, não há nenhuma categoria inserida, logo, todos favoritos só poderão ser inseridos na raiz). Para fazer isso, é necessário usar o comando " - -add-bookmark" ou sua abreviação " -ab" e informar o nome, link, categoria e um texto sobre o favorito quando requisitados:

Agora será inserido a primeira categoria (na raiz), chamada linux. Logo depois, dentro da categoria criada será inserida outra categoria, chamada debian. Para fazer isso, é necessário

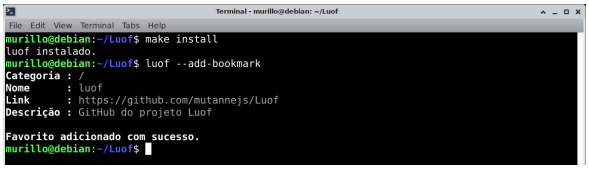


Figura 13.3: adicionando bookmark favorito

usar o comando " - -add-category" ou sua abreviação " -ac" e informar a categoria a qual ela pertence e seu nome:

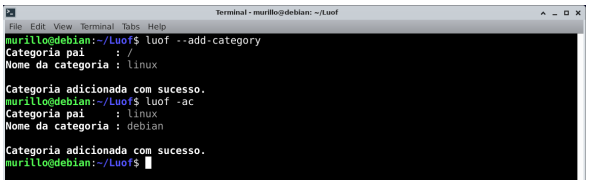


Figura 13.4: adicionando categoria

A próxima imagem mostra como adicionar favoritos às categorias existentes no programa, será inserido então o favorito news na categoria linux/debian, e o favorito foca linux na categoria linux. Dessa vez será usado o comando abreviado " -ab" que possui o mesmo efeito do comando " - -add-bookmark" usado anteriormente:

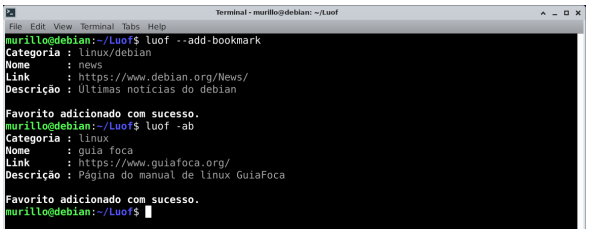


Figura 13.5: adicionando novo bookmark

Os comando anteriores não seriam úteis se não houvesse uma maneira de rever as informações guardadas. Para isso existe duas opções. A função " - -list-tree" ou sua versão abreviada " -lt" que mostra tudo que está armazenado em forma de árvore, obedecendo a hierarquia das categorias. E a função " - -list-category" ou sua versão abreviada " -lc" que mostra os favoritos e

subcategorias de uma categoria específica, a qual deve ser informada quando requisitado, neste exemplo será visualizado a categoria linux:

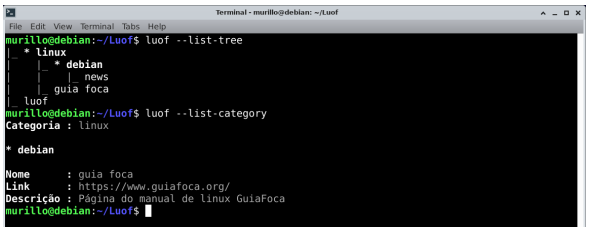


Figura 13.6: comandos list tree e list category

Neste passo, o favorito luof adicionado no início desse tópico será removido, para isso é usado a função " - -remove-bookmark" (poderia ser usado sua abreviação " -rb") e informado o nome e categoria do favorito quando requisitado. São usados as duas funções descritas no passo anterior para mostrar o resultado da remoção desse favorito, ambas agora na versão abreviada:

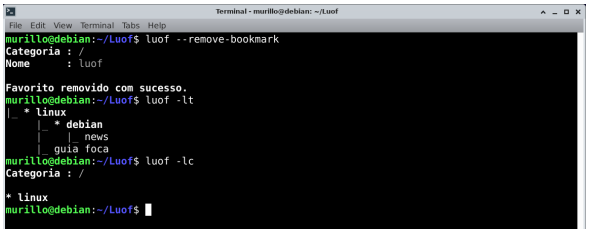


Figura 13.7: comando remove bookmark

Por fim, vemos para que serve e como usar a função " - -help" , fazemos isso usando a própria função " - -help" e passando a mesma como argumento, mas nesse exemplo no argumento ela está em sua forma abreviada " -h" :

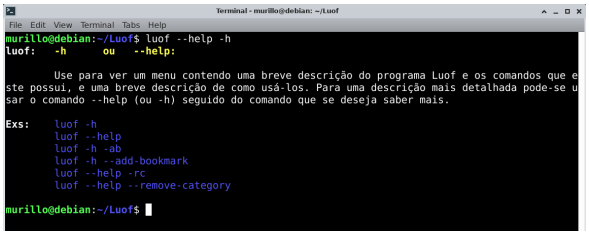


Figura 13.8: comando help

13.8 Bugs/problemas conhecidos

Este projeto foi feito pensando no sistema de diretórios do linux, portanto, não deve funcionar em outros sistemas operacionais.

13.9 Repositório

O repositório com os arquivos e demais informações sobre o projeto se encontra em:

`https://github.com/mutannejs/Luof`

Capítulo 14

Certificate Issuer

Alex Junior

João Victor Elias

14.1 Conceito do projeto

Atualmente o processo de validação de créditos da Universidade é muito burocrático. Temos uma enorme dependência em papéis e documentos que muitas vezes perdemos ao longo do curso, tanto quanto, uma dependência na secretaria do curso para a validação dos créditos de todos os estudantes.

O intuito desse projeto é utilizar os contratos inteligentes da rede distribuída do Ethereum para validar certificados da Universidade. Assim podemos garantir que o certificado foi emitido por uma entidade específica, e também, garantir que os documentos não serão perdidos, pois ficarão distribuídos na rede para sempre.

Atualmente o projeto esta funcionando para uma única entidade (Hackoonspace), mas o objetivo é aprimora-lo para pode ser utilizado por qualquer entidade.

14.2 Pré-requisitos e recursos utilizados

O projeto e as dependências são divididos em duas partes, backend e frontend respectivamente. Para o backend nós usamos Solidity para criação do contrato e Javascript e NodeJs(v16.19.1) para o script de teste e de deploy. Em relação as dependências, é necessário instalar apenas a biblioteca *hardhat* que é responsável por permitir a criação de uma blockchain local.

Já para o frontend nós utilizamos React e algumas bibliotecas comuns para ajudar na criação da interface que podem ser instaladas através do comando *npm install*.

Por fim, será necessário possuir a Metamask instalada no navegador para conseguir testar a aplicação.

14.3 Instalação

Primeiro certifique-se que tem o Javascript e o Node instalados na sua máquina. A versão do node utilizada no projeto é a v16.19.1.

OBS: os passos a seguir são para executar o contrato em uma rede local, mas o intuito é oficializa-lo na rede oficial do Ethereum.

1. Instale as dependências: garanta que está dentro da pasta raiz */cerificate-issuer*, em seguida, instale as dependências dos módulos back-end e front-end;

```
cd front-end
npm install

cd ../back-end
npm install
```

2. Verifique se o projeto está funcionando corretamente: para garantir que tudo está funcionando normalmente, dentro do caminho */back-end*, execute o seguinte comando para compilar o contrato e realizar um teste de conexão com a Blockchain local.

```
npx hardhat run scripts/
    run.js
```

Após rodar o comando deve aparecer um prompt similar a esse, podendo ter diferenças nos endereços printados.

```
Inicializando contrato.
Dono do contrato: 0xf39fd6e51aad88f6f4ce6ab8827279cfffb92266
Contract deployed to: 0x5FbDB2315678afecb367f032d93F642f64180aa3
Contract deployed by: 0xf39Fd6e51aad88F6F4ce6aB8827279cfffFb92266
[
  [
    BigNumber { value: "760921" },
    BigNumber { value: "60" },
    'Projeto BlockChain 1.0',
    'https://drive.google.com/file/d/1FBYXtkKDeTE1Fe5sYTc7mJDKLEXq8jTP/view?usp=share_link',
    BigNumber { value: "1680270798" },
    RA: BigNumber { value: "760921" },
    hoursDone: BigNumber { value: "60" },
    name: 'Projeto BlockChain 1.0',
    link: 'https://drive.google.com/file/d/1FBYXtkKDeTE1Fe5sYTc7mJDKLEXq8jTP/view?usp=share_link',
    issueDate: BigNumber { value: "1680270798" }
  ],
  ...
]
```

14.4 Execução

Para executar a aplicação localmente vá para pasta */back-end* e siga os passos abaixo:

1. Suba uma rede Ethereum local: o comando abaixo irá subir a rede e printar uma lista de carteiras criadas pelo Hardhat, para conseguirmos testar o nosso contrato, e manterá uma rede local rodando enquanto o terminal não for finalizado.

```
npx hardhat node
```

2. Faça o deploy do contrato: em outro terminal, continuando no caminho /back-end, execute o comando abaixo para fazer o deploy do contrato na rede local.

```
npx hardhat run scripts/
    deploy.js --network
    localhost
```

3. Faça a conexão entre a Metamask e a Blockchain local e adicione a carteira 0 da Blockchain de teste na Metamask: isso pode ser feito seguindo esse tutorial seguindo esse tutorial.
   OBS: quando for recriar a Blockchain local após realizar algumas operações será preciso reiniciar a Metamask, para isso siga os passos abaixo.
   - Dentro da Metamask, clique na sua foto;
   - Vá em Settings;
   - Vá em Advanced;
   - Clique em Reset Account.

4. Suba servidor do front-end: esse comando irá subir a aplicação que se comunica com a rede Ethereum local em *http://localhost:5173/*.

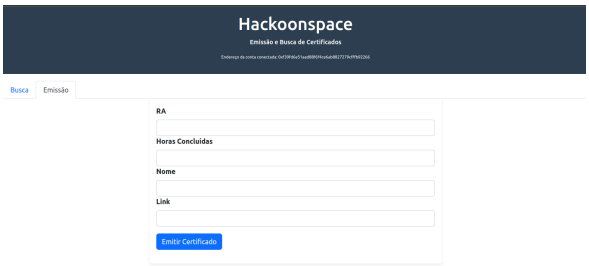
```
cd ../front-end
npm run dev
```

Como usar

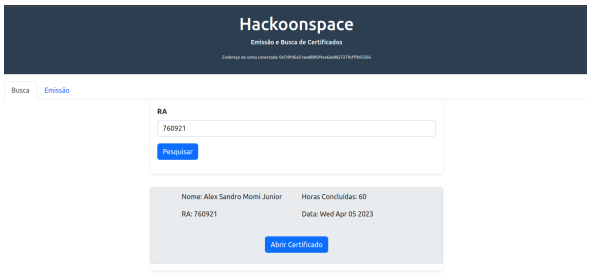
A interface da aplicação é muito simples possuindo apenas duas abas principais, Busca e Emissão. Ao abrir a aplicação você verá uma tela muito similar a está:

Para conectar a carteira e conseguir realizar emissão ou busca de certificados clique no botão "Conectar Carteira"e confirme a operação na Metamask. Após isso, você poderá realizar a emissão ou busca de certificados a vontade, precisando apenas preencher os campos de cada aba.

Emissão de Certificados



Busca de Certificados



14.5 Bugs/problemas conhecidos

Até o momento o projeto possui algumas limitações, como por exemplo a emissão do certificado que pode ser feita apenas pelo dono do contrato. No futuro vamos adicionar a funcionalidade para cadastrar outras entidades com essa permissão.

14.6 Repositório

O repositório com os arquivos e demais informações sobre o projeto se encontra em:

`https://github.com/AlexJunior01/certificate-issuer`

Capítulo 15

CHIP-8 Emulator

Leonardo Cavalcante da Silva

15.1 Conceito do projeto

CHIP-8 é uma linguagem de programação interpretada, desenvolvida por Joseph Weisbecker e inicialmente utilizada nos microcomputadores de 8-bits COSMAC VIP e Telmac 1800 em meados de 1970. Programas do CHIP-8 são executados numa máquina virtual do CHIP-8.

Esse projeto consiste em um emulador para a linguagem interpretada original do CHIP-8. Há a implementação de alguns conceitos fundamentais de Arquitetura de Computadores, como a estrutura da CPU, ciclo de instruções, entrada/saída e frequência do sistema.

15.2 Pré-requisitos e recursos utilizados

Rust

Esse emulador é construído utilizando a linguagem de programação Rust

SDL2

A biblioteca SDL2 é utilizada para lidar com a entrada do usuário, áudio e gráficos.

15.3 Passo a passo

O CHIP-8 conta com 34 opcodes (tecnicamente 35, embora um deles seja ignorado por interpretadores modernos), que têm dois bytes de tamanho.

Para garantir que todos os opcodes e a estrutura completa da CPU funcionem como planejado, todos os links na seção de Referências podem - e devem - ser utilizados. Eles apresentam informação suficiente para compreender como o a execução do emulador deve ocorrer do começo ao fim.

Em geral, inicialmente o mais importante é compreender como o ciclo busca-decodificação-execução funciona. O emulador executa em um loop infinito. Em cada loop, o emulador:

1. Busca a próxima instrução da memória de acordo com o contador de programas (PC) atual;

    - Como cada instrução tem tamanho de dois bytes, 2 deve ser somado ao contador de programas para que ele aponte para a próxima instrução na memória.

2. Decodifica a instrução para saber o que deve fazer;

3. Executa a instrução decodificada.

Testes

Em projetos de emulação, testes podem ser desafiadores. Para garantir que tudo estivesse funcionando corretamente, ROMs de teste foram utilizadas enquanto as instruções eram implementadas, assim como testes unitários para os métodos da CPU e cada instrução individual.

ROMs de teste

ROMs de teste podem ser encontradas no diretório test_roms/ deste repositório, e podem ser executadas normalmente pelo emulador assim como qualquer outra ROM do CHIP-8.

A ROM do logo da IBM foi o programa que inicialmente guiou a implementação das instruções da CPU. Tudo o que esse programa faz é apresentar o logo da IBM, utilizando apenas seis instruções:

- 00E0 (clear screen)
- 1NNN (jump)
- 6XNN (set register 'VX')
- 7XNN (add value to register 'VX')
- ANNN (set index register I)
- DXYN (display/draw)

Esse programa auxilia na implementação mais segura e rápida das instruções obrigatórias para seu funcionamento, além de testar a instrução DXYN, a instrução mais difícil de se implementar. Garantir que essa instrução funcione corretamente permite utilizar o SDL2 para apresentar os resultados na tela, o que é exigido pelas outras ROMs de teste.

Testes unitários

Testes unitários foram criados para todas as instruções do CHIP-8 e alguns métodos da CPU, num total de 40 testes unitários disponíveis em cpu_test.rs.

Para executar os testes, utilize

```
cargo test
```

Todos eles devem ser aprovados adequadamente.

15.4 Instalação

- Clone e mova o repositório através dos comandos git clone e cd:
  - Se você deseja clonar o repositório inteiro (todos os ramos), clone e mova para o ramo atual utilizando

```
git clone --
    branch sdl2-
    development
    git@github.
    com:
    leleosilva/
    CHIP-8-
    Emulator.git
cd CHIP-8-
    Emulator
```

  - Se você deseja clonar apenas o ramo atual (que usa SDL2), utilize:

```
git clone --
    branch sdl2-
    development
    git@github.
    com:
    leleosilva/
    CHIP-8-
    Emulator.git
git clone --
    branch sdl2-
    development
    --single-
    branch
    git@github.
    com:
    leleosilva/
    CHIP-8-
    Emulator.git
cd CHIP-8-Emulator
```

- Construa o projeto através do seguinte comando que usa o Cargo, o gerenciador de sistemas e pacotes do Rust:

```
cargo build --release
```

Por padrão, um arquivo binário será criado em ./target/release. Esse binário é completamente independente, e pode ser movido ou copiado para outro local em seu computador.

15.5 Como usar

Execução

A partir do diretório CHIP-8-Emulator/ o emulador pode ser executado com:

```
# Mac/Linux
./target/release/chip-8 <PATH TO ROM>

# Windows
.\target\release\chip-8 <PATH TO ROM>
```

Se você moveu ou copiou o arquivo binário, vá para o diretório que contém o binário e execute o emulador com:

```
# Mac/Linux
./chip-8 <PATH TO ROM>

# Windows
.\chip-8 <PATH TO ROM>
```

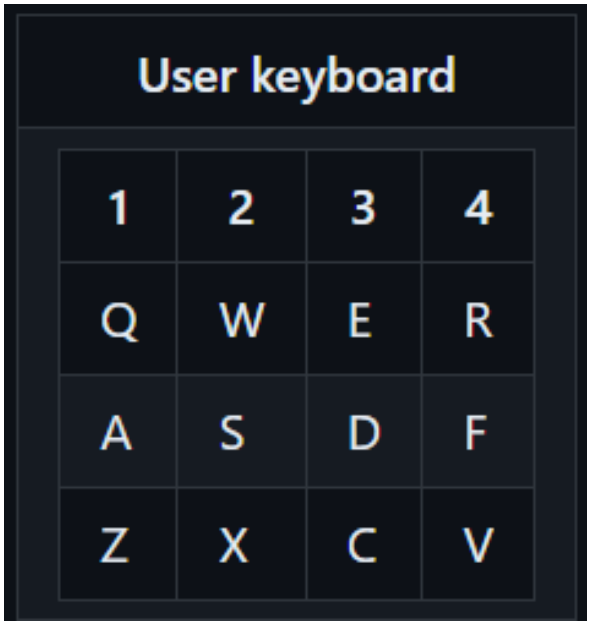
```
[leleo@fedora CHIP-8-Emulator]$ ./target/release/chip-8
error: the following required arguments were not provided:
  <ROM>

Usage: chip-8 <ROM>

For more information, try '--help'.
```

```
[leleo@fedora CHIP-8-Emulator]$ ./target/release/chip-8 --help
CHIP-8 Emulator

Usage: chip-8 <ROM>

Arguments:
  <ROM>  path to ROM file

Options:
  -h, --help  Print help
```

Controles

O CHIP-8 usa um teclado hexadecimal de 16 teclas rotuladas de 0 a F, organizadas em um grid 4x4, com o seguinte layout:

É habitual mapear o teclado original utilizando o lado esquerdo de um teclado QWERTY. Neste projeto, as teclas utilizadas pelo usuário têm o seguinte layout:



15.6 Repositório

O repositório com os arquivos e demais informações sobre o projeto se encontra em:

`https://github.com/leleosilva/CHIP-8-Emulator`

15.7 Referências

Cowgod's Chip-8 Technical Reference v1.0
Guide to making a CHIP-8 emulator
How to write an emulator (CHIP-8 interpreter)
Building a CHIP-8 emulator [C++]
An Introduction to Chip-8 Emulation using the Rust Programming Language
Awesome CHIP-8
CHIP-8 - Wikipedia
/r/EmuDev

Capítulo 16

Comparador de preços com Web Scraping

Erik Gabriel Rodrigo da Silva

Rafael Mori Pinheiro

Thiago Kraide de Lima Fernandes

16.1 Conceito do projeto

O projeto em questão tem como objetivo demonstrar o funcionamento de um *Bot* de *web scraping* o qual possui como premissa a realização da coleta de dados da web de maneira totalmente automatizada e a estruturação desses dados para melhor manejo da informação, além disso, haverá a implementação do robô em questão para que este consiga obter valores de produtos na internet. Ademais, vale salientar que é cada vez mais interessante esta estratégia de acesso aos dados pela gigantesca quantidade de informação presente na internet, viabilizando a criação de melhores estratégias para lidar com as experiências dos usuários e até mesmo para o próprio crescimento das empresas.

16.2 Implementação e recursos

Durante a Implementação do robô os principais recursos utilizados foram: a plataforma de código aberto Anaconda 3 aliado ao *Jupyter* Notebook para o desenvolvimento do código assim como a organização do mesmo, as bibliotecas *Openpyxl*, Pandas para que se viabilizasse a manipulação de tabelas em que se estruturam os dados gerados pelo *Bot*, a biblioteca *Selenium* como principal ferramenta para automação dos testes em navegadores web, as bibliotecas *Matplotlib*, *Seaborn* para a construção do gráfico gerados com os dados coletados e por fim a biblioteca *Tkinter* que possibilitou toda a progressão quanto as interfaces de início e de final do programa.

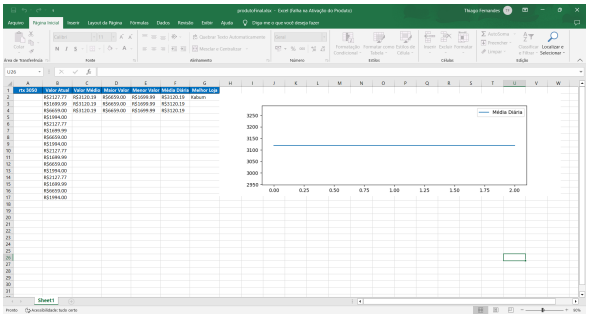
16.3 Passo a passo

Sob tal lógica, o passo-a-passo para o desdobramento do projeto foi dado por: processo de criação dos *prompts*, ou seja, das interfaces para recolher as informações que o usuário deseja buscar e também a janela que "retorna" o devido funcionamento da aplicação, com um *feedback* de suas buscas. Após isso, foi elaborado o código que proporciona a obtenção dos valores de um site, ou por melhor dizer o próprio *web scraping*, e concomitantemente manipula os dados inseridos e realiza os cálculos das métricas adequadamente. Em adição, foi desenvolvida uma *Newsletter* que irá informar o usuário que o produto analisado pelo robô a seu pedido está em seu preço desejado.

16.4 Execução e como utilizar

A utilização da aplicação é bem intuitiva! Basta o usuário baixar os arquivos do repositório e "rodar" o arquivo executável do programa, inserindo os dados nos campos requisitados e aguardando o trabalho do *Bot*. Há outra maneira de utilizar a aplicação, menos intuitiva porém funcional de mesmo modo, onde são baixados os arquivos e há a compilação e execução do próprio código fonte.

16.5 Bugs/problemas conhecidos



Os *Bugs* e problemas conhecidos de nosso projeto cerceiam a possibilidade de alterações de *pathing*, seja pela manutenção dos sites ou alterações nestes, e a necessidade de "reinicializar"a planilha inicial quando for necessário alterar um produto, ou seja, quando se deseja procurar um produto diferente. Além destes, é necessário, para um mesmo produto, rodar a aplicação 2 vezes, a fim de obter o funcionamento adequado!

16.6 Repositório

O repositório com os arquivos e demais informações sobre o projeto se encontra em:

`https://github.com/Rafael-Mori-2022/Comparador-de-Precos`

16.7 Imagens/screenshots