



# Game Chaining

Relatório de Estágio

João Pedro Neves Gonçalves
2018014306

Orientadores:
Prof. Anabela Jesus Gomes | ISEC
Diogo Vasconcelos | Nerd Monkeys
Alexandre Barbosa | Nerd Monkeys

Licenciatura em Engenharia Informática
Ramo de Desenvolvimento de Aplicações
Instituto Superior de Engenharia de Coimbra
Instituto Politécnico de Coimbra

Setembro de 2021

# Agradecimentos

Começo já a agradecer ao *Instituto Superior de Engenharia de Coimbra* (*ISEC*) pelo ensino, aprendi muito nestes 3 anos tudo graças, não só aos professores, muitos dos quais estavam sempre disponíveis a ensinar, como também aos colegas que encontrei, que me ajudaram e apoiaram. Agradeço especialmente à minha orientadora Anabela Jesus Gomes que estava sempre disponível para ajudar.

Agradeço a todas as equipas na Nerd Monkeys, especialmente a minha Team 7, Team Patas e à Team Golden Snub.

Quero agradecer em especial ao Alexandre Barbosa por me ter integrado na empresa e ajudar em tudo o que era preciso, ao Diogo Vasconcelos que me orientou e apoiou em todo o meu percurso, ao Iuri Martinho por insistir comigo e ajudar-me no que era preciso e à Mafalda Duarte.

Agradeço aos meus colegas de curso Ana Filipa Alves, Marco Domingues, Eduardo Pina e Rafaela Santos pela ajuda e apoio ao longo desta jornada.

Por último, gostaria de agradecer à minha família por ter-me dado o apoio necessário.

# Resumo

No mundo dos jogos de vídeo, têm vindo a ser experimentadas variadas tecnologias, sendo as mesmas posteriormente evoluídas de forma a melhor satisfazerem o objetivo da sua utilização.

A *Nerd Monkeys*, uma empresa desenvolvedora de jogos de vídeo com interesse em explorar novas tecnologias, foi a empresa onde o estágio apresentado neste trabalho se desenvolveu.

A primeira tecnologia usada no estágio foi a *blockchain*, tal como estava planeado na proposta de estagio. Relativamente ao *blockchain* foi maioritariamente estudada a tecnologia subjacente, o seu modo de funcionamento, os seus protocolos e as suas aplicações genéricas e concretamente nos jogos de vídeo.

Para experimentar alguns dos conceitos acumulados na pesquisa sobre *blockchain* foi realizado um projeto designado de *AC7ION* do tipo servidor-cliente. Ambos foram desenvolvidos em *Rust*. O servidor faz a troca de informações com a *Application Programming Interface* (*API*) do *Twitter*. Este servidor envia os *tweets* obtidos da *API* para uma biblioteca partilhada quando requisitados. A biblioteca pode ser usada por qualquer *game engine* que suporte uma *Application Binary Interface* (*ABI*) para a linguagem *C*.

O último projeto efetuado foi um detetor de batimentos cardíacos com o sensor de *Infrared* (*IR*) da *Nintendo Switch* desenvolvido em *C++*. Este teve como objetivo dar a conhecer o *hardware* e *software* desta consola de jogos de vídeo, permitindo conhecer o espaço de desenvolvimento de uma consola e o que pode ser nela usado e desenvolvido.

***Palavras-chave:*** *Blockchain, Rust, IR Sensor, Jogos de vídeo*

# Abstract

Various technologies are tried out in the world of video games, being later evolved in ways to better satisfy the purpose of its use.

Nerd Monkeys, the company where the internship took place, is a video game developer company that has the interest in exploring new technologies.

The first technology used in the internship was blockchain, as it was planned in the internship proposal. For the blockchain it was essentially explored its underlying technology, its working mode, its protocols and its general and specific applications in video games.

To try out some of the concepts accumulated in the blockchain research a server-client type program was performed. Both developed in Rust for exchanging information with the Twitter API. This program sends the information obtained from the API to a shared library. The shared library, also performed at this stage can be used by any game engine that supports an ABI for the C language.

The last project carried out was a heart beats detector for Nintendo Switch IR sensor developed in C++. This was developed to make known the hardware and software of this video games console allowing to know its development space and what can be used on it.

***Keywords:*** *Blockchain, Rust, IR Sensor, Video Games*

# Índice

# Lista de Figuras

# Lista de Tabelas

# Capítulo 1

# Introdução

Na área dos jogos de vídeo várias são as tecnologias exploradas de forma a melhorar as experiências dos utilizadores ao jogar.

O trabalho apresentado no presente relatório representa também uma contribuição nesta temática. Consiste numa sumarização do trabalho desenvolvido, tanto a nível teórico como prático, no estágio curricular no âmbito da unidade curricular de Projeto ou Estágio na empresa Nerd Monkeys, de 1 de março de 2021 a 18 de julho de 2021, para efeitos de conclusão da Licenciatura de Engenharia Informática, do *Instituto Superior de Engenharia de Coimbra* (*ISEC*).

Neste capítulo, é apresentado o contexto do estágio, os seus objetivos, entidades envolvidas na realização do mesmo, bem como a estrutura do relatório.

## 1.1 Instituto Superior de Engenharia de Coimbra

O *ISEC* é uma das unidades constituintes do *Instituto Politécnico de Coimbra (IPC (Educação))*. A missão do *IPC (Educação)* corresponde à criação, transmissão e difusão de cultura, ciência e tecnologia. Para isso, é ministrada formação de nível superior para o exercício de atividades profissionais no domínio da Engenharia, promovendo o desenvolvimento da região em que se insere. Foi fundado em 1974, tendo sido adicionado ao *IPC (Educação)* em 1988.

O *IPC (Educação)* tem uma variada oferta formativa disponível ao nível de mestrados, licenciaturas e *Curso Técnico Superior Profissional* (*CTeSP*). O *Departamento de Engenharia Informática e de Sistemas* (*DEIS*) dedica-se, desde 1989, à formação, investigação, desenvolvimento e prestação de serviços na área da Engenharia Informática. O *ISEC* prima pelo caráter prático dos cursos que ministra, traduzido pelo elevado número de aulas laboratoriais, projetos e estágios o que, aliado à qualidade científica e pedagógica, resulta num elevado índice de empregabilidade. [20]

## 1.2 Nerd Monkeys

A Nerd Monkeys foi fundada em 2013 por Diogo Vasconcelos e Filipe Duarte Pina [24]. A empresa desenvolve e publica jogos, pelo que está sempre a investir em novas ideias para um jogo, tanto em histórias como tecnologias e paradigmas.

Como exemplo de jogos desenvolvidos na *Nerd Monkeys* podem ser citados os jogos "*Inspector Zé e Robot Palhaço em: Crime no Hotel Lisboa*" (2014), "*Inspector Zé e Robot Palhaço em: O Assassino do Intercidades*" (2018), "*Out of Line*" (2021) e o "*Monkey Split*" (2021) [21] publicados na Steam, uma plataforma de distribuição e venda de jogos.

A empresa também tem um histórico de adaptar jogos para a *Nintendo Switch*, dos quais são exemplos os jogos "*Inspector Zé e Robot Palhaço em: Crime no Hotel Lisboa*"[19], e um jogo de outro programador publicado pela *Nerd Monkeys* cujo o nome do jogo é "*Traffix*" [37]. Porém, uma das áreas que a empresa quer explorar é a aplicação de *blockchain* em jogos *online*. Neste sentido, o estágio decorre no contexto da pesquisa de novas tecnologias para usar em novos jogos, neste caso especifico utilizando *blockchain*.

## 1.3 Proposta de Estágio

A proposta de estágio (anexo A) elaborada pela *Nerd Monkeys* visou a investigação da aplicação de *blockchain* em jogos de vídeo, com a adição da utilização do motor de jogo *Unreal Engine* com *C++*.

O planeamento do desenvolvimento ficou divido nas seguintes tarefas:

- T1 - Pesquisa e experiência prática – Durante este período o estagiário deverá realizar uma pesquisa extensiva sobre todas as componentes envolvidas no projeto e experimentação prática para validar os resultados da pesquisa. (5 semanas)
- T2 - Criação de ferramentas - tendo em vista as etapas T3 e T4. (4 semanas)
- T3 - Criação de protótipo – Criar um protótipo com base na pesquisa desenvolvida com um objetivo definido pelo orientador.(3 semanas)
- T4 - Desenvolvimento do videojogo – Com base no protótipo e ferramentas criadas deverá ser desenvolvido um videojogo em, pelo menos, formato de vertical *slice* (demonstração jogável de todas as mecânicas). (10 semanas)

- T5 - Elaboração de relatório de estágio. (2 semanas)

O planeamento temporal das várias fases é apresentado na tabela 1.1.

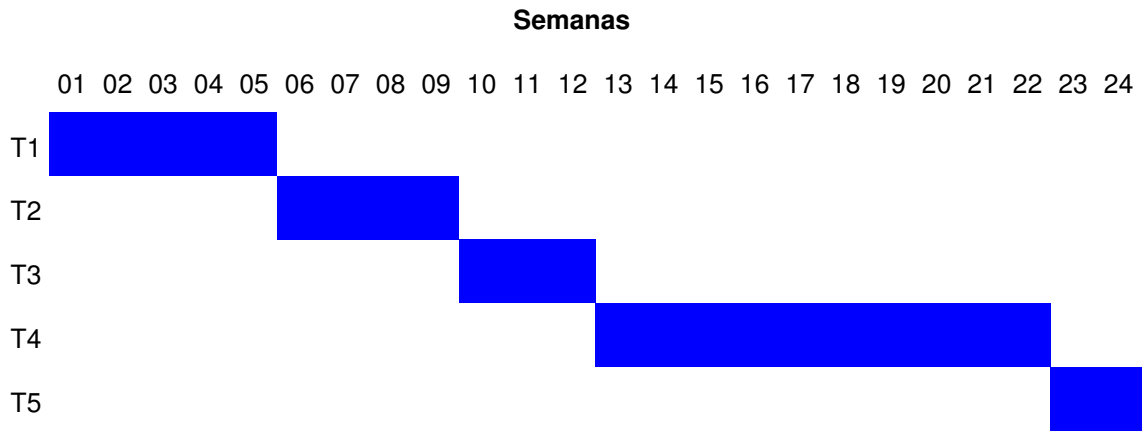


Tabela 1.1: Diagrama de Gantt representativo do planeamento temporal

## 1.4 Objetivos e Plano de Trabalhos

Os objetivos iniciais deste trabalho, consistiam em aferir as capacidades da tecnologia *blockchain* para além da sua utilização comum atual e aplicação concreta em jogos de vídeo com a subsequente criação de um prototipo funcional. Porém, foi considerado de interesse que o trabalho fosse principalmente de pesquisa, dadas as enormes capacidades e potencialidades desta tecnologia e assim aferir quais as melhores opções para a aplicar no contexto da empresa.

## 1.5 Estrutura do relatório

Este documento, para além da introdução, é composto pelos seguintes capítulos:

- Capítulo 1 - Este é o primeiro capítulo, começa com a descrição da entidade de acolhimento e da instituição de ensino, seguindo-se a apresentação da proposta de estágio, os objetivos definidos e plano de trabalhos. Termina com esta secção que descreve a estrutura do relatório, bem como, a metodologia e notas extras.
- Capítulo 2 - Neste capítulo é realizada a apresentação da primeira parte do estágio, onde foi realizada pesquisa sobre o tema de *blockchain*. É realizada a descrição das várias tecnologias relacionadas com *blockchain*, bem como análises e conclusões de como estas podem ser aplicadas num jogo.

- Capítulo 3 - Este capítulo apresenta a segunda parte do estágio, onde é relatado o desenvolvimento de um programa denominado *AC7ION*. A função deste programa é a de receber *tweets* para serem enviados a um cliente que faz comunicação com jogos.
- Capítulo 4 - Este capítulo apresenta a terceira parte do estágio, onde foi desenvolvido um detector de batimentos cardíacos, para a *Nintendo Switch*, que usa o *Infrared* (*IR*) Sensor desta para os detetar.
- Capítulo 5 - Este capítulo trata da conclusão. No mesmo são tecidas algumas ideias finais sobre o trabalho desenvolvido e é analisado o sucesso do trabalho realizado.

Em complemento e como suporte de informação a estes capítulos, são também apresentadas as referências bibliográficas e um anexo, estes podem ser vistos no final do documento.

## 1.6 Metodologia e Notas Extras

Os projetos aqui apresentados foram todos desenvolvidos como projetos internos da *Nerd Monkeys*. Em projetos internos da empresa os membros das equipas têm grande liberdade criativa, o que significa que todos têm a liberdade de discutir e mudar o rumo do que está a ser desenvolvido se assim o líder do projeto concordar e for bem justificado.

Sendo que a metodologia de desenvolvimento era ágil e comparável com a *Extreme Programming*, em que existe constante *feedback*, reuniões todos os dias, em que a equipa está em constante contacto.

Os requisitos descritos nos vários capítulos foram inseridos com o propósito de definir os objetivos do trabalho a realizar. Tendo isso em conta, esses requisitos servem apenas como guias/ ideias base, pelo que podem não ser seguidos à risca durante o processo de desenvolvimento.

# Capítulo 2

# Blockchain

## 2.1 Introdução

Como requerido na proposta de estágio este trabalho centrou-se na pesquisa de *blockchain*. O objetivo desta pesquisa foi de averiguar a pertinência e possibilidades da área de *blockchain* e do mundo de tecnologias descentralizadas para a área dos jogos de vídeo. Esta pesquisa permitiria concluir se é possível ou não desenvolver jogos descentralizados baseados em *blockchain* em que estes só precisam do poder computacional dos dispositivos dos jogadores para fazer verificação de dados e transições num jogo *online*.

Os jogos feitos com o projeto *Game Chaining* devem ser o menos centralizados possível, reduzindo a utilização de servidores da empresa, permitindo que seja a comunidade do jogo a gerir os servidores do mesmo. Este tipo de independência não permite a utilização de mecanismos de *pay-to-win*, sendo que a empresa só lucrará com a compra dos conteúdos dos jogos como música, modelos, texturas e outros.

O capítulo divide-se em várias partes, sendo estas:

- O que é *blockchain*? - Nesta secção é explicado o conceito de *blockchain* e termos relacionados.
- Como *Blockchain* é aplicado no mercado? - Nesta secção é listada a maneira como esta tecnologia é usada no mercado.
- Como aplicar Blockchain no contexto de um jogo de vídeo? - Nesta secção é indicada a forma como a tecnologia *blockchain* pode ser aplicada em jogos de vídeo.
- Conclusões - Nesta secção podem ser vistas conclusões sobre os assuntos expostos neste capítulo.

## 2.2 O que é *blockchain*?

Na sua aplicação mais básica é um conceito bastante simples de explicar:

"*Blockchain é um tipo de Base de Dados Distribuída que guarda um registo de transações permanente e à prova de violação. Uma base de dados em blockchain consiste em dois tipos de registos: transações individuais e blocos.*

*Um bloco é a parte concreta da blockchain onde são registadas algumas ou todas as transações mais recentes e uma vez concluídas são guardadas na blockchain como base de dados permanente. Sempre que um bloco é concluído um novo é gerado. Existe um número incontável de blocos na blockchain que são ligados uns aos outros - como uma cadeia - onde cada bloco contém uma referência para o bloco anterior.*" [40]

Duma forma mais visual pode se usar o esquema referido na figura 2.1, em que cada bloco contem o *hash* do bloco anterior e a raiz da *Merkle Tree*, que é uma árvore de *hashs* para indexar o conteúdo.

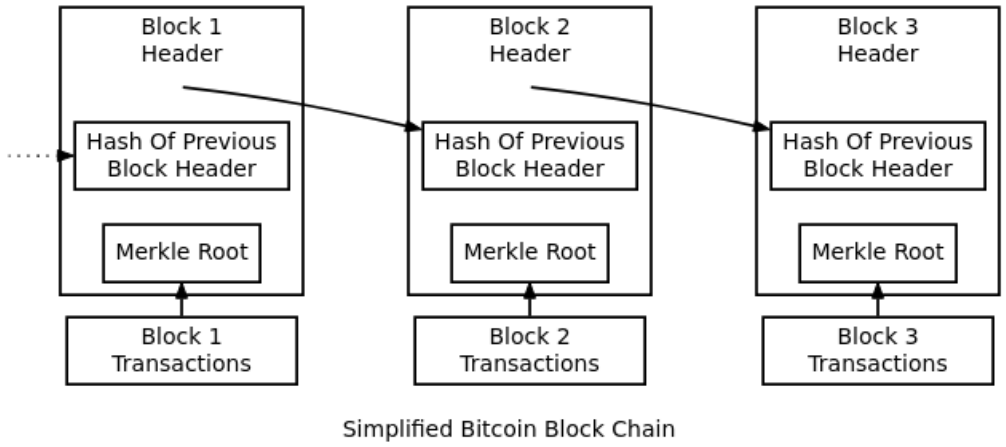


Figura 2.1: Visão geral do *blockchain* no *Bitcoin* [30]

O conceito de *blockchain* foi introduzido por David Chaum em 1982 [15] e foi conceptualizado por uma pessoa ou grupo conhecido como "Satoshi Nakamoto"em 2008 que publicou o design e implementação do que hoje é conhecido como *Bitcoin*. [26]

## 2.3 Como *Blockchain* é aplicado no mercado?

No mercado existe uma grande variedade de implementações de *blockchain*. Algumas implementações são aplicadas como bases de dados, sendo que outras são aplicadas como criptomoedas.

### 2.3.1 Criptomoedas

Criptomoedas são a forma mais popular de usar *blockchain*. Existe um grande número de criptomoedas, porém baseiam-se num conceito base que foi evoluindo. Devido à evolução da ideia, criptomoedas podem-se separar em 3 gerações: ouro digital, contratos inteligentes e escalabilidade.

#### ***Bitcoin*: Ouro Digital**

A primeira geração de criptomoedas é a aplicação de *blockchain* para fins monetários, sendo o maior exemplo desta aplicação a criptomoeda *Bitcoin* (figura 2.2).

Figura 2.2: Logótipo do Bitcoin

"*Bitcoin é uma criptomoeda descentralizada, sendo um dinheiro eletrónico para transações ponto-a-ponto. O primeiro artigo descrevendo uma implementação do Bitcoin foi criado em 2008 sendo apresentado no começo de 2009 a lista de discussão The Cryptography Mailing por um programador ou grupo de programadores sob o pseudónimo Satoshi Nakamoto. Bitcoin é considerada a primeira moeda digital mundial descentralizada, constituindo um sistema económico alternativo, e responsável pelo ressurgimento do sistema bancário livre.*" [46]

#### ***Ethereum*: Contratos Inteligentes**

*Ethereum* (figura 2.3) é uma criptomoeda conceptualizada em 2013, inicializada em 2015 por Vitalik Buterin. Esta tem a especificidade de ter capacidade de executar contratos inteligentes e é a segunda cropitomoeda mais valorizada, estando atrás de *Bitcoin*, no que toca a valor de mercado. [9]

Figura 2.3: Logótipo do Ethereum

Um contrato inteligente é um protocolo de computador auto executável criado com a popularização das criptomoedas. Contratos inteligentes são feitos para facilitar e reforçar a negociação ou desempenho de um contrato, proporcionando confiança em transações *online*. Contratos inteligentes permitem que pessoas desconhecidas façam negócios de confiança entre si, pela Internet, sem a necessidade de intermédio de uma autoridade central.

*Ethereum* também começou a implementação do seu *upgrade*, chamado de *Ethereum 2.0*, que inclui técnologias como *Proof of Stake* (*PoS*) e *sharding*, que serão analisada em mais detalhe nas subsecções 2.3.2 e 2.3.3 respectivamente.

### *IOTA*: Escalabilidade

Publicada em 2016 por Serguei Popov e com o propósito de resolver o problema de baixo número de transações por segundo, [43] , *IOTA* (figura 2.4) baseia se na ideia de *Tangle*, isto é, em vez de usar uma *blockchain* com uma única linha de blocos usa grafos acíclicos dirigidos.

Figura 2.4: Logótipo do IOTA

Nesta rede não existem mineiros, a verificação de cada transição é feita pelos dois nós anteriores. [25] Explicando de uma forma mais visual, pode se verificar na figura 2.5 que numa *blockchain* normal, são adicionados blocos um a um sendo que 51% da rede tem de estar em concordância resultando numa velocidade constante. No caso da *IOTA* a velocidade estala com o número de blocos na rede já que blocos podem ser verificados em paralelo.

Para se alcançar consenso existe uma entidade central da fundação *IOTA* que faz transições sem valor para verificar se os nós são legítimos.

*IOTA* tem a desvantagem de ser centralizada pelo facto de ter coordenadores que são entidades confiadas e centralizadas.

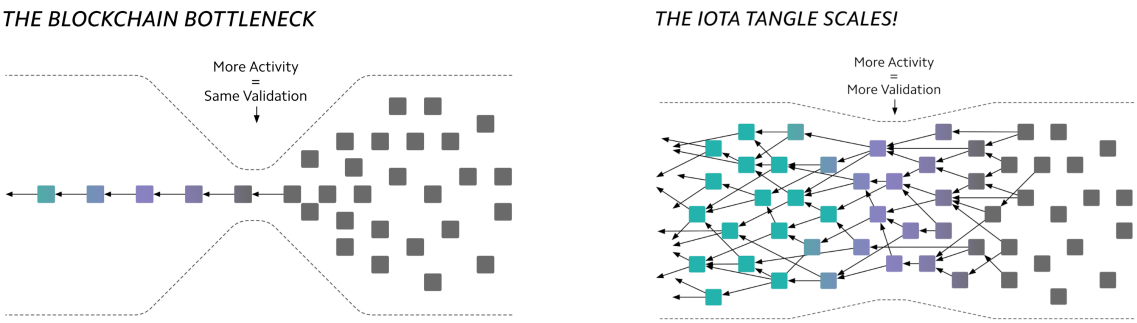


Figura 2.5: Uma *Blockchain* comum comparada com o *Tangle* do *IOTA* (editada) [43]

### 2.3.2 Consensos

"*Um problema fundamental em sistemas de processamento distribuído é alcançar a confiança geral do sistema com a existência de processos defeituosos. Ademais, isso, normalmente, requer que processos concordem que algum dado ou valor seja necessário durante uma computação. Exemplos de aplicações de consenso distribuído incluem confirmar ou não uma transação para uma base de dados indefinida e concordar em relação à identidade de um líder.*" [41]

Noutras palavras *concensus* ou consenso na área da computação é o processo de confirmar que existe uma percentagem mínima de aplicações na rede que concordam com uma certa informação.

Existem vários algoritmos que podem fazer esse trabalho sendo que os de seguida apresentados parecem ser os mais utilizados:

- *Proof of Work (PoW)* - é um sistema que cria um problema computacional cuja resolução tem um elevado custo de processamento mas é fácil de verificar. Como exemplo de implementação desta teoria refira-se a heurística de *Fiat-Shamir* [42] e o *hashcash* que é o algoritmo usado pelo *Bitcoin*. [17] É se chamado aqueles que fazem as verificações de mineiros. [31]

- *Proof of Stake (PoS)* - é um sistema que, em vez de criar um problema computacional para se verificar uma transição, coloca em causa o dinheiro daqueles que querem verificar. Se alguém quiser manipular de forma maliciosa uma transação terá, não só de ter 51% do dinheiro que existe numa *blockchain*, como também deve estar pronto a arriscar perder o dinheiro e fazer o seu valor descer, desencorajando assim ataques por causa do alto risco. Outra vantagem reside no facto de não ser preciso dispositivos de alto poder computacional para efetuar verificações. [12]

### 2.3.3 *Sharding*

*Sharding* é o conceito de subdividir uma base de dados para que se melhore a sua escalabilidade. Isto é, quanto maior for o número de utilizadores na rede, melhor deve ser a divisão de trabalho. No contexto de *blockchain*, quanto maior o número de utilizadores, maior o número de transições por segundo. [14]

A implementação de *sharding* é mais comum em bases de dados e criptomoedas com entidades confiadas pela rede como *IOTA*, sendo a presença de *sharding* em criptomoedas públicas e descentralizadas sem elementos confiados ou favorecidos muito menos comum.

### 2.3.4 Framework e Protocolos

Também foram encontradas várias aplicações e *frameworks* que podiam ser utilizadas ou poderiam servir como inspiração para aplicar no projeto. Uma lista dessa aplicações/ *frameworks* poderá ser vista na tabela 2.1.

| Nome | Descrição |
|---|---|
| *Hyperledger* | Uma *pool* de projetos, apoiada pela *Linux foundation*, relacionados com *blockchain*. [11] |
| *BigchainDB* | Base de dados em *blockchain*. Construída com base em *consensus* de um líder. [2] |
| *InterPlanetary File System* (*IPFS*) | "*Sistema de Ficheiros Interplanetário é um protocolo e uma rede projetada para criar um armazenamento associativo ponto-a-ponto endereçável ao conteúdo de armazenamento e compartimentação, de hipermédia, num sistema de ficheiros distribuído.*" [39] |
| *Ethereum Casper* | *Casper* é o protocolo que os pesquisadores de *Ethereum* estão a estudar e a desenvolver para implementar *PoS* e *sharding* em *Ethereum*. [33] |

Tabela 2.1: *Frameworks* e Protocolos

## 2.4 Como aplicar Blockchain no contexto de um jogo de vídeo?

Nesta secção é indicada a forma como a tecnologia *blockchain* pode ser aplicada em jogos de vídeo.

### 2.4.1 Mercado Existente

No mercado de jogos de vídeo, a aplicação de *blockchain* encontrava-se maioritariamente em jogos que usavam as *Application Programming Interfaces* (*APIs*) do *Ethereum* e outros com interesses monetários [3] em que os jogos são construídos mais como um casino virtual ou um local de trocas de objetos num mundo virtual com dinheiro real. É disso exemplo o jogo *CryptoKitties* em que houve uma venda de 100 mil dólares por um item do jogo [4], o que não é o tipo de jogo que a *Nerd Monkeys* tem em mente como já explicado na introdução deste capítulo. Para tal começou-se a pensar em implementar algo, à parte, com o objetivo de ser usado exclusivamente como mecanismo de troca de informações entre instâncias do jogo ou jogos caso se queira utilizar a mesma rede entre jogos diferentes.

### 2.4.2 Requerimentos

Para o desenvolvimento da biblioteca *Game Chaining* existem os seguintes requerimentos:

- **R1** - Deve ser agnóstica quanto ao motor de jogo e sistema operativo, isto é, não deve ter quaisquer dependências ou requerimentos que a impeça de ser usada noutras plataformas, exceto se as plataformas em questão forem muito restritas relativamente ao que se pode executar.

- **R2** - Deve ser rápida e responsiva, sem grandes requisitos para ser executada, isto é, evitar que use muito do *Central processing unit* (*CPU*) ou da memória, de forma que não afete muito o desempenho do jogo. Mais especificamente, pretende-se que responda em menos de 10 milissegundos relativamente à chamada de uma função, para não afetar o jogo, exceto se assim for pretendido.

- **R3** - Deve ser o mais descentralizada e autónoma possível, o que significa que deve existir o mínimo de manutenção da parte da empresa.

- **R4** - Não pode estar presa a licenças comerciais, de forma que possa ser de código aberto para uso livre pela comunidade quando assim se pretender.
- **R5** - Ser estável e com um design robusto, isto porque, depois de lançado será difícil mudar algo por causa do funcionamento de uma *blockchain*.

Estes são os requerimentos que guiam a procura do que é mais apropriado para implementar este projeto, desde a linguagem de implementação, aos algoritmos, arquitetura e à forma como irá interagir com os motores de jogos.

### 2.4.3 Implementação

Começou-se então a pesquisar sobre como seria desenvolvida essa biblioteca e quais as ferramentas a usar.

### 2.4.4 Desenvolvimento em C++

Figura 2.6: Logótipo de *C++*

A primeira linguagem a considerar para o projeto seria *C++* (figura 2.6). A razão para tal era o seu suporte existente em qualquer plataforma de implementação de jogos, o facto de não ter um *garbage collector*, o código não ser compilado para bytecode ou executado em *runtime*, sendo que em termos de execução o programa seria o mais rápido o possível.

Para o desenvolvimento da biblioteca estudou-se o que seria preciso para criar uma *blockchain* e quais os requesitos, usando como exemplo o *bitcoin* (também implementado em *C++*). Em primeiro lugar, estudou-se o *ZeroMQ*, para a parte de comunicação de rede, que também foi usado no desenvolvimento de *Bitcoin*, e que integrava outros conceitos úteis para o projeto.

#### ZeroMQ

Figura 2.7: Logótipo do *ZeroMQ*

*ZeroMQ* começou o seu desenvolvimento em 2007 pela iMatix, uma empresa de pesquisa e desenvolve software, para substituir o predecessor *AMQP* [7] e foi anunciado pronto para produção em 2010 [18]. É uma biblioteca de comunicação assíncrona, com o objetivo de ser usada para aplicações concorrentes ou distribuídas.

Em *ZeroMQ* existem tipos especiais de *sockets*, que se ligam a outros tipos compatíveis formando padrões.

Por exemplo Pedido-Resposta (figura 2.8), em que se liga um par de *sockets REQ-REP* para que quando um pedido for feito seja fornecida uma resposta, este padrão tem uma relação 1-1, em que um pedido é respondido por uma resposta.

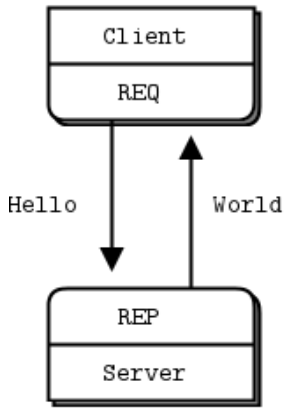


Figura 2.8: Padrão Pedido-Resposta

No padrão Publicador-Subscritor, representado na figura 2.9, cria-se um *socket PUB* e múltiplas *sockets SUB*, sendo que quando uma mensagem é enviada pelo PUB é recebida por todas as *sockets SUB*, tendo uma relação 1-X, sendo X um número positivo de publicadores.

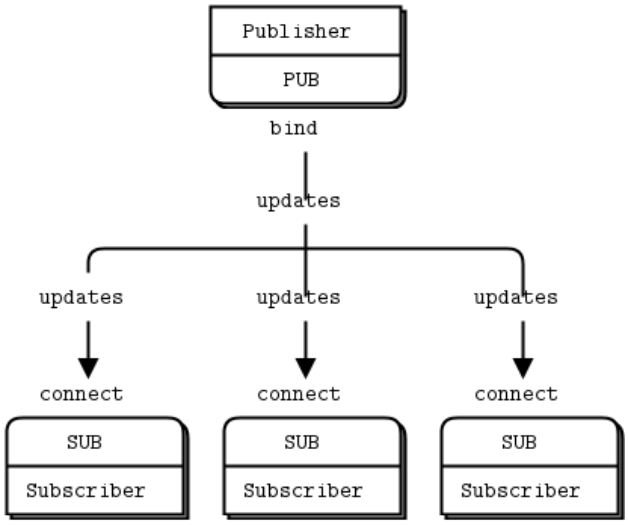


Figura 2.9: Padrão Publicador-Subscritor

No padrão Pipeline Paralelo, representado na figura 2.10, usam-se *sockets* PULL e PUSH para melhor distribuição de trabalho. O *Ventilator* envia trabalhos para os trabalhadores e, quando terminados, enviam-se os resultados à *Sink*.

Existe também o Pedido-Resposta Estendido, representado na figura 2.11, em que se estende o padrão Pedido-Resposta para algo mais paralelizado.

Foi seguido o *zguide*, um guia para aprender a usar o *ZeroMQ*, para o estudo destes padrões existindo também explicações para outros padrões mais avançados, como também para fazer aplicações completas utilizando *ZeroMQ*. [47]

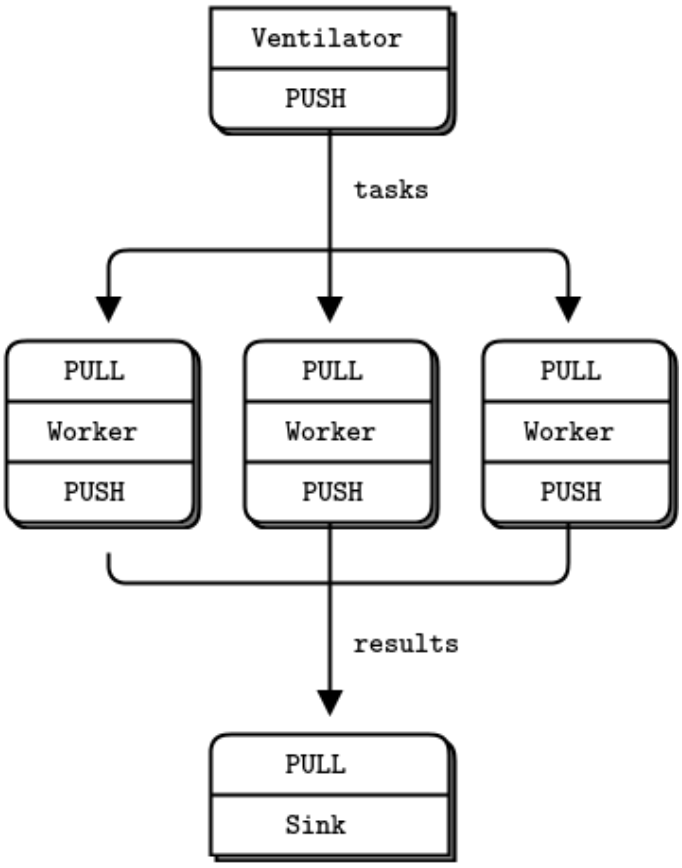


Figura 2.10: Padrão Pipeline Paralela

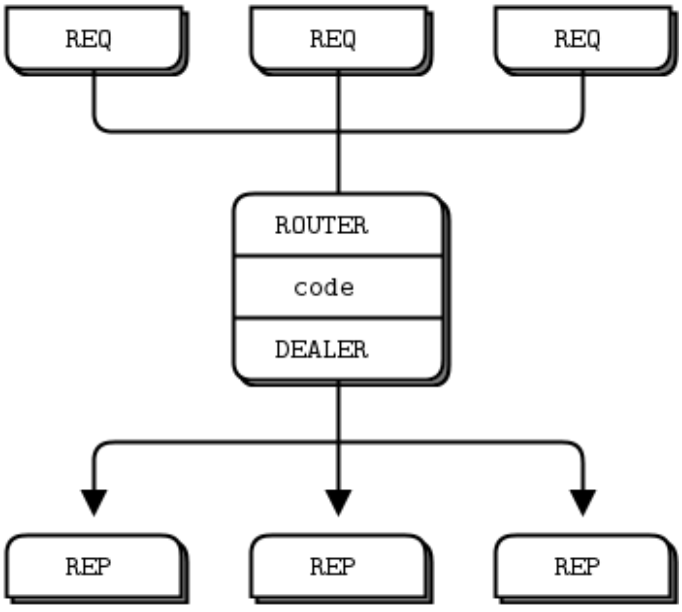


Figura 2.11: Padrão Pedido-Resposta Estendido

### Formato dos dados

Para que seja mais fácil de transferir dados entre implementações e linguagens diferentes deve-se usar um padrão aberto de troca de dados, tanto para ser utilizado através da rede como para comunicar informação entre a biblioteca e os motores de jogos.

Para este problema encontraram-se duas soluções notáveis.

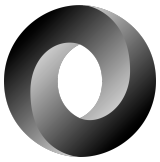

Figura 2.12: Logótipo do *JSON*

A primeira solução consiste em usar *JavaScript Object Notation* (*JSON*) (figura 2.12). *JSON* é um formato compacto, simples e legível por humanos, é bastante usado e comum em vários sistemas. Muitos motores de jogos e linguagens de programação incluem formas de interagir com este formato, seja com plugins, bibliotecas ou o que pode já estar integrado na ferramenta em questão. O *C++* não integra este aspeto na sua *Standard Library* (*std*), porém, existem múltiplas bibliotecas *online*, de que é exemplo a *json* (mantida por *nlohmann* no *Github* [28]) tendo sido a escolhida por ser entendida como a linguagem mais apropriada.

A segunda solução designada *msgpack* ou *MessagePack* é mantida pela Google. Consiste numa solução equivalente ao *JSON* mas muito mais compacta. [16]. Pode-se visualizar uma comparação entre os dois na figura 2.13, em que se tem a mesma informação em ambos mas com formatos diferentes, o do topo *JSON*, o de baixo *MessagePack*. A figura explica como a estrutura do *MessagePack* está estruturada e a diferença de tamanhos entre as duas.

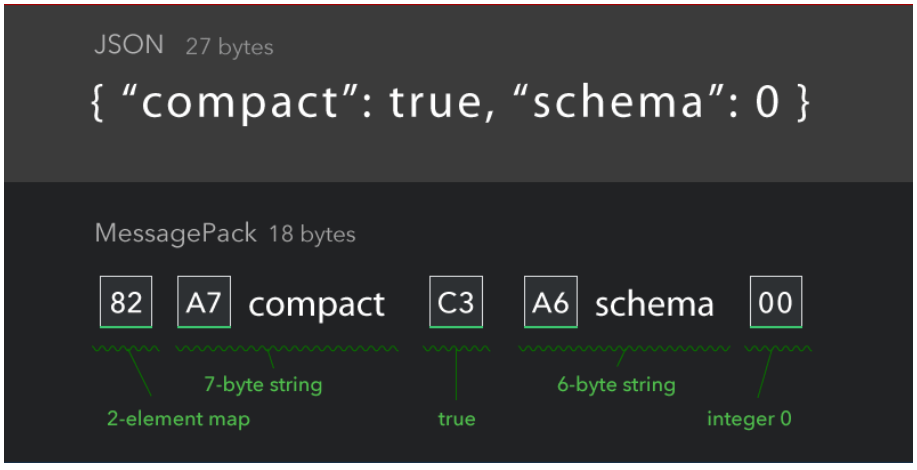


Figura 2.13: Comparação entre MessagePack e *JSON* [16]

Dada a conveniência do formato *JSON* e pelo facto de ser muito mais utilizado e compreensível por humanos decidiu-se escolher esta opção.

### Hashing

Como o objetivo é fazer uma *blockchain* é necessário conseguir criar *hashs* dos blocos e também útil para outras funcionalidades algorítmicas.

*Hashing* é o processo de dando o mesmo exato conteúdo resulta sempre o mesmo *hash*. Com um *hash* pode se verificar se dados foram modificados ou corrompidos.

Para tal, escolheu-se o *picosha2* como biblioteca de *hashing* com *SHA256*.

### *packaging*

Recapitulando, já se escolheu *C++* como a linguagem de programação, *ZeroMQ* como a biblioteca para *networking*, *JSON* como o formato de envio de mensagens e *picosha2* para *hashing*.

Com isto já se tem um número considerável de bibliotecas. Em *C++* não existe um gestor de pacotes por predefinição mas existem bastantes possibilidades de escolha como o *vcpkg*, o *conan*, o *hunter* entre outros. No entanto, como nem sempre existem os pacotes que se precisa nestes gestores, pode até haver a necessidade de usar vários ou, o que mais realisticamente acontece, ser o programador a fazer inclusão manual dos pacotes.

Como é previsível que este projeto seja complexo existe a necessidade de um gestor de pacotes para gerir software externo e um gestor do projeto para *packaging*, testes e automatização de *build*. Assim, para este objetivo a escolha óbvia seria o *CMake*. Este gere processos de compilação independentes do compilador a ser usado, testes, verificação de dependências entre outros. Também pode ser integrado com gestores de pacotes como *hunter* que é escrito em CMake e que é fácil de integrar num projeto de CMake o que é o melhor candidato considerando que tem ZeroMQ na sua lista de pacotes, sendo esse o único pacote que não é apenas uma biblioteca *header-only* [1].

### Alternativas a considerar

*C++* é uma óptima linguagem para velocidades de execução e em grande percentagem das plataformas mas poderá ter vários problemas de segurança de memoria, principalmente em aplicações *multithreaded* o que pode aumentar o tempo a depurar o projeto.

[1] bibliotecas *header-only* são compostas por apenas ficheiros .h/.hpp em *C/C++*. Estes existem por causa de serem mais fáceis de integrar num projeto mas com o custo de tempos de compilação

*ZeroMQ* é uma boa biblioteca para desenvolver a *networking stack* porém, durante a pesquisa feita, o *libp2p* começou a ser considerada como uma melhor opção. A libp2p é explicada mais abaixo neste documento.

Surgiu assim a ideia de estudar o que outras *blockchains* usavam para desenvolvimento, concluindo-se que a maior parte desta área usava *Go*, uma linguagem de programação simples e relativamente rápida, compilada para código maquina e com um *garbage collector*, fala se mais detalhadamente sobre *Go* na subsecção 2.4.6. Porém, encontrou se outra linguagem de desenvolvimento perfeita para os requisitos, *Rust*, resolve muitos dos problemas apresentados sobre o *C++*.

### 2.4.5 Rust

*Rust* é uma linguagem de programação multi-paradigma compilada projetada para ser ”segura, concorrente e prática”. Pode se ver um exemplo da sua sintaxe na figura 2.14. Desenvolvida inicialmente pela *Mozilla* para desenvolver o *Firefox*, isto porque em *C++* e noutras linguagens é difícil de depurar código *multithreaded*. A primeira versão estável de *Rust* foi lançada em 2015. No dia 8 de fevereiro de 2021 foi estabelecida a *Rust Foundation* com a *Amazon*, *Huawei*, *Microsoft*, *Google* e *Mozilla* como fundadores. [45]

```
ola.rs
fn main() {
    println!("Olá Mundo!");
}
```

Figura 2.14: Olá Mundo! em *Rust*

*Rust* tem um gestor de pacotes e projeto integrados, este chamado de *cargo* [35], a linguagem compila para código maquina como *C* e *C++* e possui o conceito de *ownership* e *lifetimes* das variáveis integrado na própria linguagem, o que permite detetar com facilidade erros de concorrência e de *use after deletion*. Adicionalmente, muitas bibliotecas relacionadas com *blockchain* tem implementações desenvolvidas com *Rust*.

Muitas empresas mostram interesse em *Rust*, sendo que este está a ser considerado como a segunda linguagem de programação a ser suportada na *Linux Kernel* e a *Google* utiliza-a para desenvolver novos módulos para *Android* e outros sistemas. Isto porque *Rust* é extremamente seguro. De acordo com estudos 70% dos *bugs* severos de segurança são causados por problemas de

memoria (figura 2.15) [6] [5]. Isto pode parecer um problema que acontece só com linguagens que mexem diretamente com memoria, como o *C* e o *C++* mas isto também se pode aplicar a *data races* [2]. *Rust* não tem estes problemas face às suas regras restritas de *ownership*.

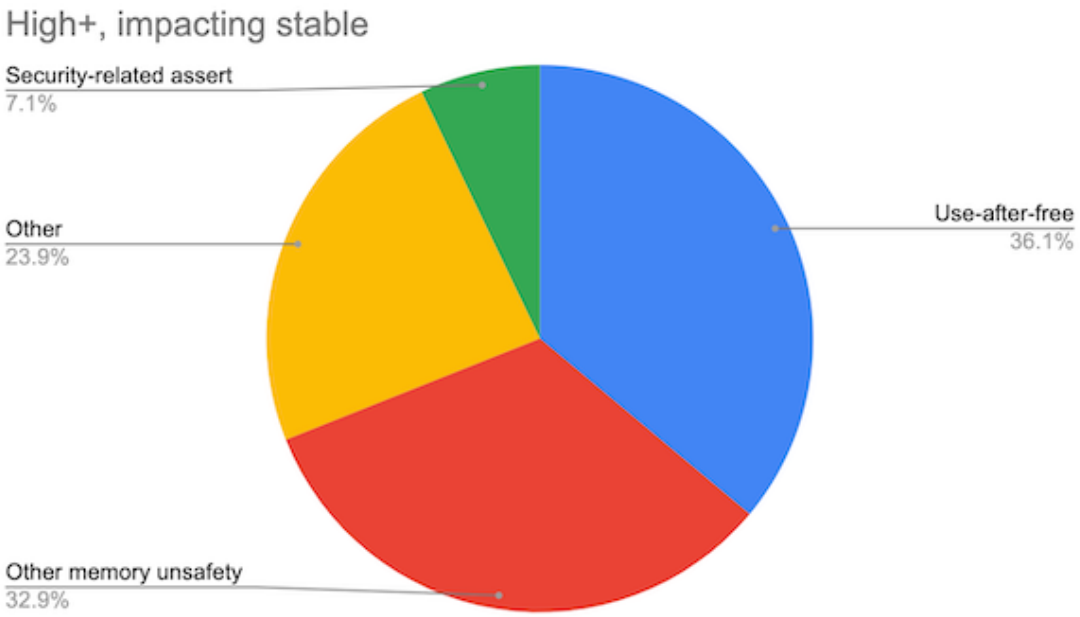


Figura 2.15: Analise em base de 912 *bugs* de segurança de severidade alta ou crítica desde 2015, que afetavam o canal estável do *Chromium* [5]

*Ownership* funciona com a ideia de que cada espaço de memória, seja este um numérico, uma estrutura ou outro tipo de dados, é propriedade de uma variável. Este espaço de memória pode ser dado a outra variável ou emprestado (borrowed). Uma variável pode ser ou não mutável, isto é, aquando da sua declaração, pode ser definido se o programador consegue ou não mudar o seu valor.

E com estas propriedades é introduzido o *borrow checker*. Durante a compilação de um programa escrito em *Rust*, o *borrow checker* verifica estas propriedades e a forma como são usadas, provocando erros de compilação caso ocorram condições que podem resultar em *bugs* ou *crashs* provocados por acessos incorretos à memoria. Por exemplo, uma variável que deu posse do seu conteúdo a outra não pode mais ser usada porque não tem conteúdo. Em *Rust*, isto daria um erro de compilação caso fosse tentado, em vez de, na execução, um *segment fault* em *C* e *C++* ou uma *NullException* em linguagens com um *garbage collector* como *Java*. Com o *borrow checker* também é possível verificar erros

[2] *ocorrência de múltiplas linhas de execução acederem ao mesmo espaço de memoria sem a apropriada sincronização e pelo menos uma escrita nesse espaço*

relacionados com *data races*, o que para acontecer seria necessário que fossem feitos dois acessos à memoria que

1. apontassem para a mesma memoria,
2. estivessem em linhas de execução simultâneas,
3. pelo menos um fosse uma escrita e
4. não estivessem sincronizados.

Porém, o *borrow checker* consegue verificar quando é que as variáveis

1. têm acesso à mesma memória,
2. estão em linhas de execução diferentes,
3. são mutáveis e
4. estão sincronizadas [3]

resultando num erro de compilação quando estas condições ocorressem.

Outras empresas estão a considerar utilizar esta linguagem para projetos em que seja importante a segurança e rapidez, o que reforça a impressão que é uma linguagem que vai ser bem suportada no futuro com novos projetos que noutras circunstâncias usariam *C* ou *C++*.

Depois destes pontos, e considerando o requerimento de estabilidade em R5 decidiu-se usar o *Rust* em vez de *C++*. Sobre as bibliotecas, encontrou-se facilmente *bindings* para *ZeroMQ* e as outras duas bibliotecas tinham alternativas como *sha2* para *hashing* com *SHA-256* e *serde* para serializar e desserializar *JSON*, isto é, transformar as estruturas de *Rust* em *JSON* e vice-versa.

### 2.4.6 Go

*Go* também foi considerado como outra opção de linguagem de programação a utilizar. Esta está em grande utilização na implementação de bases de dados em *blockchain* e criptomoedas portanto também se encontram bastantes bibliotecas e exemplos para a implementação das mesmas.

*Go* é uma linguagem feita a pensar em simplicidade com poucas regras de sintaxe e funcionalidades com o objetivo de diminuir o tempo de desenvolvimento e facilitar a sua leitura. Tal como *C* é muito simples de aprender, tem um *garbage*

[3] isto é possível com os *traits Sync* e *Send* que são detalhes de como *mutexes* e outros estruturas de sincronização são implementadas em *Rust*, dando o contexto necessário ao *borrow checker* [34]

*collector* para gestão de memoria para que assim o programador não tenha a preocupação de gerir essa memoria.

A razão da escolha de *Rust* em vez de *Go* foi essencialmente devido à sua segurança em termos de gestão de memoria, o que já foi explicado na subsecção anterior e as seguintes razões:

- *Rust* não têm um *garbage collector*, a gestão de memória é feita por um conceito chamado *Ownership* parecido com o *Resource Acquisition Is Initialization (RAII)* em C++, não existindo assim problemas associados ao *garbage collectors*.
- O compilador de *Rust* verifica *data races* em tempo de compilação e erros comuns em que torna o código compilado muito mais seguro, o que evita gastar tempo em depuração e cria programas mais estáveis.

### 2.4.7 libp2p

Durante a pesquisa foi encontrada uma *framework* designada *libp2p* que está a ser desenvolvida em conjunto com o *InterPlanetary File System* (*IPFS*) [23], um protocolo de rede *peer-to-peer* para guardar e partilhar informação num *File System* (*FS*) distribuído. [39]

A *libp2p* foi criada com o objetivo de ajudar a criar aplicações com protocolos de comunicação descentralizados ao contrario do recorrente paradigma de servidor-cliente, que se tem vindo a usar nas ultimas décadas, demonstrando assim uma ótima biblioteca para ajudar no desenvolvimento da aplicação.

### 2.4.8 *Network Address Translation* (*NAT*) *Traversal*

Um problema que surgiu durante as escolhas de *networking* foi o de como se iria produzir um jogo completamente descentralizado sem que os utilizadores tivessem a necessidade de ”abrir portas”ou fazer outras operações no seu *router* em casa? Isto é um problema com que os utilizadores se deparam com a atual *Internet*. Dificilmente se faz uma ligação direta entre computadores, sem que existam portas abertas em, pelo menos, um dos lados. Em alguns casos não é possível sem utilizar um *router* alternativo ao que o fornecedor de *Internet* oferece.

Para este problema a solução mais apropriada é *NAT traversal*, isto é uma técnica de estabelecer uma conexão entre *gateways* que implementam *NAT* [44].

Estudaram-se algumas técnicas, em especial *hole punching* [13], usadas em aplicações como *Skype*, *Voice over Internet Protocol* (*VoIP*), aplicações *Peer-to-*

*peer* (*P2P*), entre outros. *Hole punching* é a ação de, através de um servidor externo com um IP estático público, estabelecer a conexão entre 2 ou mais nós na rede atrás de uma *NAT*. [29]

*Libp2p* também está a desenvolver a sua implementação deste aspeto mas ainda não está finalizado. [22]

### 2.4.9 Comunicação

Para a comunicação entre nós do *Game Chaining* foi considerado o *libp2p*. Se for usado um serviço externo inicializado quando o jogo inicia ou que esteja constantemente ligado, deve ser feito por *Inter-Process Communication* (*IPC (Computação)*) entre o serviço e uma biblioteca que se liga ao jogo apenas para habilitar o jogo a comunicar com o serviço. A biblioteca comunicará com o jogo através de *Application Binary Interface* (*ABI*).

O serviço pode ser usado como um servidor que fica conectado constantemente na rede. Este poderia ser comunicado com outras aplicações, não só com instâncias de jogo, como também para verificar valores ou o estado do jogo (com uma interface com o utilizador, caso se queira observar o que está a ocorrer).

Também se pode usar o serviço como um nó para fazer *hole punching* caso assim seja visto como necessário.

As ligações podem ser visualizadas na figura 2.16 em que um cliente (ou nó) liga se a outros nós e ao seu jogo respetivo.

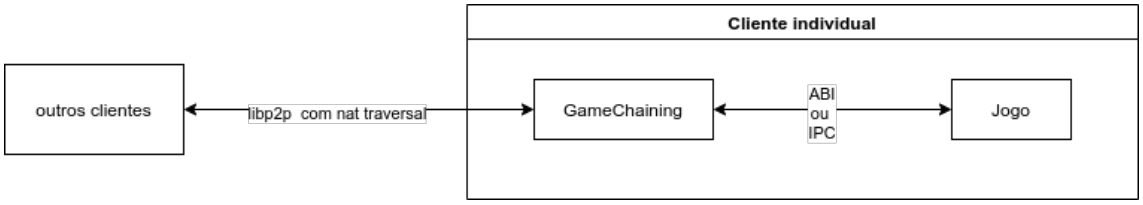


Figura 2.16: Comunicações do *Game Chaining*

## 2.5 Conclusões

Após este estudo consideraram-se os seguintes conceitos, estudados em detalhe e abordados anteriormente, para possibilitar a implementação do projeto em causa:

- *Rust* - como a linguagem de programação pelas razões já referidas em 2.4.5
- *PoS* - verificação de transações, em que se podem atribuir itens do inventario do utilizador em *stake* 2.3.2
- *Sharding* - para poupar utilização de memória distribuindo a *blockchain* entre múltiplos dispositivos 2.3.3
- *libp2p* - uma biblioteca a utilizar para a *stack* de *networking* ser mais facilmente desenvolvida
- *zstd* - método de compressão em tempo real desenvolvido pelo *Facebook* para rápidas descompressões [10], podendo ajudar a diminuir o tamanho que a *blockchain* ocupa no disco caso seja necessário.
- *Hole punching* - para não obrigar os utilizadores(ou pelo menos não todos) a terem portas abertas
- *IPC (Computação)* ou *ABI* - para comunicação com o jogo propriamente dito.

Com as opções apresentadas pode-se concluir que é possível construir o *Game Chaining* (nome dado ao projeto referido no anexo A) com tecnologias e algoritmos já existentes facilitando o seu desenvolvimento.

Existem algumas preocupações sobre o espaço que pode ser ocupado no disco e os requerimentos de *networking* mas podem-se limitar essas preocupações com *sharding*2.3.3 e compressão para resolver o problema de armazenamento e

*hole punching* para não obrigar todos os utilizadores a modificar as suas redes.

Estas modificações podem trazer preocupações pelo facto de obrigarem a um certo nível de centralização em que iria existir a necessidade de confiar num certo número de indivíduos para que funcionasse. Isto contradiz o requisito R3 mas é aceitável pois é um conceito parecido com jogos com *hosts*, como *Minecraft*, em que um utilizador iria ter a responsabilidade de ter um servidor pronto para outros jogarem, o que remove a centralização de uma empresa única.

As propriedades do *Game Chaining* consideradas anteriormente ajudariam os jogadores menos proficientes tecnicamente a jogar enquanto que outros membros da comunidade mais capazes poderiam ter benefícios no jogo por ajudar a manter o jogo activo. Também podem ser feitos serviços em que um jogador aloca um espaço num servidor externo para ter a sua própria instância do serviço do jogo, como fazem algumas empresas como a *Azure* e o *Linode* com servidores de *Minecraft*, por exemplo.

Blockchain pode ajudar em termos de estabilidade do mundo em que podem existir múltiplos *hosts* para o mesmo mundo e existe a verificação e registo de mudança de dados o que pode servir como um *Anti-cheat*, podendo se criar mundos ativos muito maiores do que seria possível num jogo como *Minecraft* ou outros que têm um único servidor.

Uma pesquisa intensa ajudou em muito a obter informação de como seria a aplicação desta tecnologia neste contexto. Porém, existem ainda várias questões práticas que se devem considerar e rever. Por exemplo, ainda existem duvidas quanto à existência de servidores da comunidade para resolver os problemas referidos anteriormente.

Pelo facto de este ser um projeto mais extenso do que o esperado, como se pode observar comparando a proposta de estágio e os requisitos descritos neste capítulo, o projeto tornou se mais amplo do que inicialmente planeado, isso porque o interesse aumentou ao longo da pesquisa e nas sucessivas reuniões.

Como não se podia concluir com certeza se haveria uma implementação para apresentar no final de estágio decidiu-se mover o estagiário para outros projetos com uma estrutura mais comum para implementar.

# Capítulo 3

# AC7ION

## 3.1 Introdução

Depois de estudado o conceito de blockchain investiu-se tempo a desenvolver uma ferramenta que teria uma estrutura parecida com o que se desejava que o *Game Chaining* fosse em termos de comunicação com os motores de jogos. Assim, o *AC7ION*, tornou-se numa prova de conceito neste aspeto.

O *AC7ION* é um recetor de *tweets* com uma # (*hashtag*) específica e enviando-os a um jogo que os requisite.

## 3.2 Requisitos

Neste projeto existiam requisitos semelhantes aos apresentados no projeto *Game Chaining*:

- **R1** - Deve ser agnóstico quanto ao motor de jogo e sistema operativo, isto é, não deve ter dependências ou requerimentos que o impeçam de ser usado noutras plataformas, exceto se as plataformas em questão forem muito restritas relativamente ao que se pode executar
- **R2** - Deve ser rápido e responsivo, sem grandes requisitos relativamente à execução, isto é, evitar que se use muito *CPU* ou memória, de forma que não afete muito o desempenho do jogo
- **R3** - Não pode estar presa a licenças comerciais sendo possível ser de código aberto caso se queira publicar
- **R4** - Receber os *tweets* em tempo real no servidor quando estes são enviados

- **R5** - Guardar até x *tweets* recebidos para serem enviados para o cliente

Com estes requisitos é possível fazer algo comparável com a planeada estrutura de comunicação entre um jogo e a biblioteca do *Game Chaining*.

## 3.3 Tecnologias

Neste trabalho aplicaram-se certos conhecimentos que se adquiriram ao longo do estudo sobre *blockchain*, nomeadamente a linguagem *Rust* e uma das suas bibliotecas para comunicação assíncrona designada *Tokio*.

### 3.3.1 *Tokio*

*Tokio* é um *runtime* assíncrono para a linguagem de programação *Rust*. O *runtime* é composto por um *condutor de entradas e saídas*, gestor de tarefas, temporizador e uma piscina de bloqueamento, em que todos estes mecanismos juntos criam uma camada para gestão e execução, de forma assíncrona, de tarefas.

Foi criada para ocupar o espaço que faltava na *std* no espaço de funções assíncronas e o *runtime* para as executar concorrentemente, chegou a uma versão estável em janeiro de 2021. Esta pode ser usada em vários sistemas incluindo sistemas embutidos ou grandes servidores com vários cores em que a biblioteca escala-se bem sem grande esforço do programador.

Facilita o desenvolvimento de programas eficientes e concorrentes de redes graças às funcionalidades do *Tokio* com *Rust* já descritas anteriormente. [38]

### 3.3.2 Outras bibliotecas usadas

Para que o servidor consiga comunicar com a *API* do *Twitter* usou se a biblioteca *egg-mode* que foi desenvolvida para esse propósito. [8] *Egg-mode* foi escolhida por ter a maior popularidade no *github* e pela facilidade de utilização.

Para serialização e desserialização de *JSON* usaram-se os pacotes *serde* e *serde_json*. Estas bibliotecas foram usadas para converter *tweets* recebidos no formato *JSON* para serem enviadas aos motores de jogo.

Para aspetos de data e hora foi usada a biblioteca *chrono*. Esta era usada para desenvolver operações como verificar as idades dos *tweets*, tempos de envio, etc.

Para constantes e funções do padrão de *C* usou-se a biblioteca *libc*. Esta serviu para criar as funções que iriam ser usadas por *ABI* pelos motores de jogo.

Para uma implementação de estruturas de sincronização mais rápidas do que as existentes na *std* usou-se a biblioteca *parking_lot*. Esta demonstrava melhores resultados na estrutura *mutex*, uma estrutura que guarda um recurso para que seja usado entre linhas de execução para prevenir condições de corrida. [1]

Para abstrações de programação assíncrona foi usada a biblioteca *futures*. Esta foi utilizada para fazer operações em iteradores assíncronos, como a *stream* de *tweets* em que usou-se

para processar cada *tweet* individual.

## 3.4 Ferramentas

Para desenvolvimento usou-se *Clion*. Um *Integrated Development Environment* (*IDE*) criado pela *JetBrains*, com o objetivo de ser um *IDE* para *C/C++*. A *JetBrains* publica na loja de *plugins* do *Intellij* e *Clion* o *plugin Rust* para adicionar suporte para desenvolvimento deste nestes *IDEs* como demonstrado na figura 3.1.

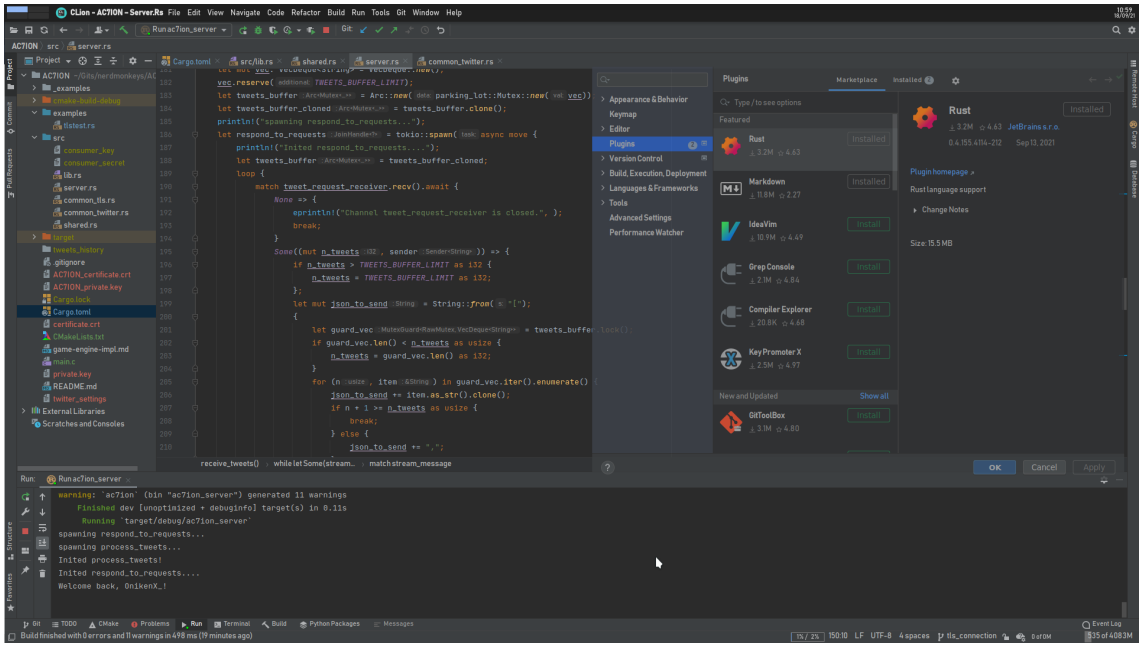

Figura 3.1: Ambiente de desenvolvimento: *Clion* com o *plugin* para *Rust*

## 3.5 Arquitetura

Usa-se um modelo servidor-cliente, em que um servidor se coneta com as *APIs* do *Twitter*, recebem-se os *tweets* necessários e enviam-se para os clientes

que seriam bibliotecas partilhadas para os jogos usarem.

Na prática os jogos fazem o pedido de x *tweets* e o cliente faz o pedido ao servidor para receber as informações num formato de *JSON*, sendo que no lado do jogo não é necessário preocupações com qualquer funcionalidade de *networking*.

Na figura 3.2 podemos ver de uma forma mais visual como tudo está conectado.

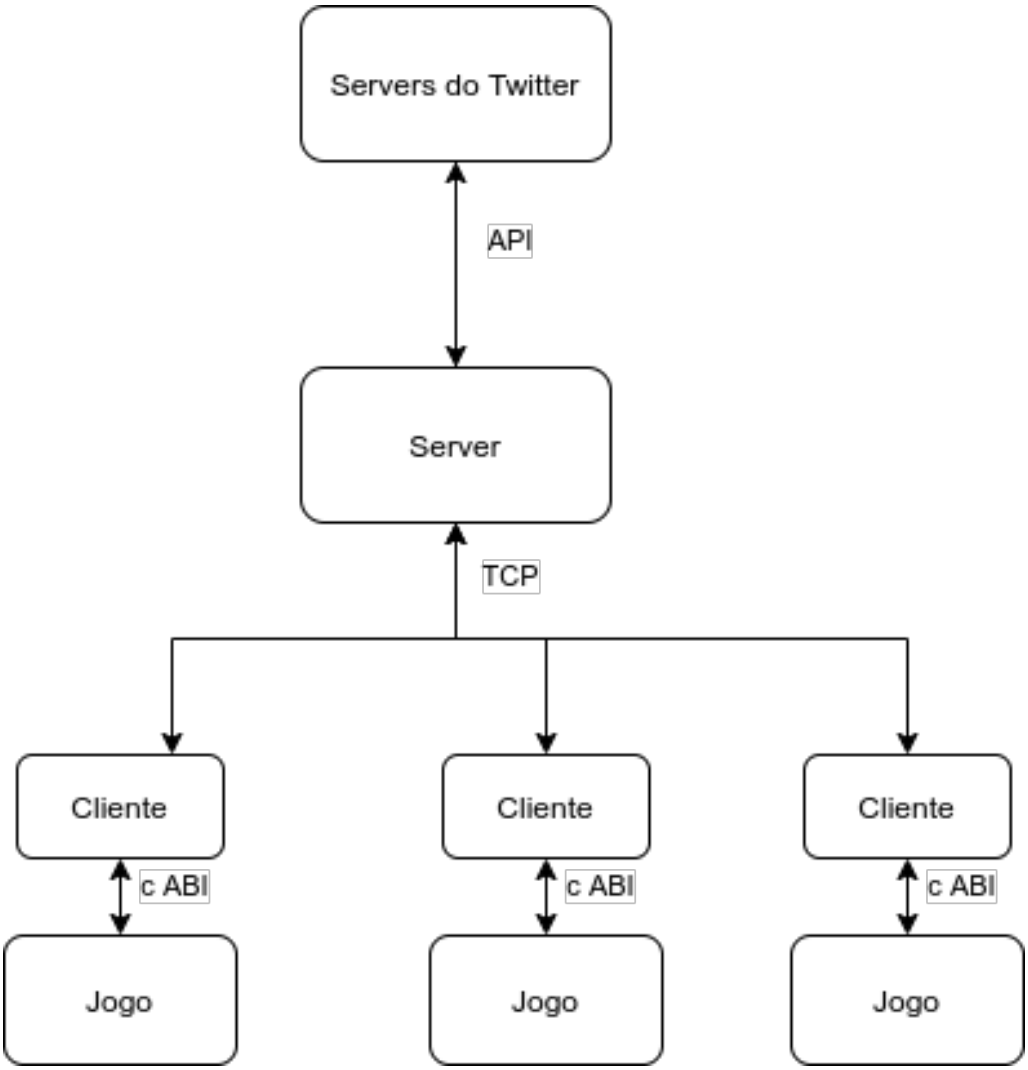


Figura 3.2: Arquitetura do *AC7ION*

O server é *multithreaded* e assíncrono, o que significa que ele irá usar automaticamente todos os *cores* do *CPU* do sistema em que está a ser executado. O programa também é *memory safe* e não tem *data races* já que a verificação destes problemas são verificados em tempo de compilação. [38]

## 3.6 Conclusão

Com o objetivo de testar a arquitetura pretendida para o *Game Chaining*, completou-se o *AC7ION* num espaço de 2 semanas. Este projeto comprovou a viabilidade desta arquitetura, proporcionando mais experiência ao estagiário em *blockchain* e na criação de bibliotecas para motores de jogos de vídeo.

# Capítulo 4

# Detector de batimentos cardíacos para a *Nintendo Switch*

## 4.1 Introdução

Como tarefa pré-final, foi realizada a missão de experimentar desenvolvimentos para consolas de jogos, tendo-se a ideia de fazer um detetor de batimentos cardíacos para a *Nintendo Switch* e um jogo com ele designado de *DMS*. Para este projeto, o único requisito era o de detetar os batimentos cardíacos de alguém com o polegar no sensor.

## 4.2 Nintendo Switch

A *Nintendo Switch* (figura 4.1) é uma consola de jogos de vídeo híbrida produzida pela empresa *Nintendo* com várias funcionalidades inovadoras no mercado, aspeto que caracteriza esta bem conhecida empresa.

Existem várias funcionalidades nesta consola não muito comuns noutras, como por exemplo dois comandos removíveis nas laterais, designados *Joy-Cons*, podendo ser usados como comandos individuais ou como um só em conjunto.

Em ambos existem botões que se espelham um ao outro na sua quase totalidade, exceto alguns botões com funções especiais para servirem de comandos individuais quando necessário. Ambos têm um giroscópio e um acelerómetro para rastrearem movimentos; existe um *Near-Field Communication* (*NFC*) no *Joy-Con* esquerdo e um sensor de *IR* no lado direito, este ultimo é aquele que vai ser utilizado para o *DMS*.


Figura 4.1: *Nintendo Switch* [27]

## 4.3 DMS

O *DMS* é um jogo, desenvolvido pela *Nerd Monkeys*, cujo o objetivo é o de um jogador, que esteja a segurar num *Joy-Con* direito com o polegar em cima do sensor (ver figura 4.2), tentar manter a calma enquanto outros jogadores tentam aumentar o ritmo cardíaco do outro jogador. Isto pode ser conseguido da forma que acharem mais apropriada, com historias ou mecanismos que assustem ou destabilizem o outro jogador (ex: fazer cócegas).

No contexto do jogo, o estagiário ficou responsável por fazer o detetor de batimentos cardíacos de forma a tornar esta ideia possível.

## 4.4 Sensor de IR

O *input* do sensor de *IR* pode ser facilmente acedido por uma aplicação ou biblioteca na consola, de varias formas possíveis.


Figura 4.2: *IR Sensor* da *Nintendo Switch* [36]

Existem várias formas de usar o sensor mas a forma mais conveniente é detetar a intensidade da luz recebida, o que pode ser usado para detetar a quantidade de oxigénio no sangue que passa no polegar. O tempo que demora entre a intensidade de subir e descer pode ser vista como o tempo de um batimento, com este tempo pode-se calcular os batimentos por minuto.

## 4.5 Arquitetura

Pode-se observar na figura 4.3 a arquitetura do projeto, que este é desenvolvido de forma independente do jogo. O jogo só necessita de incluir a biblioteca e chamar a função para atualizar os dados, recomenda-se que esta seja chamada 60 vezes por segundo ou numa frequência superior para uma melhor deteção do valor.

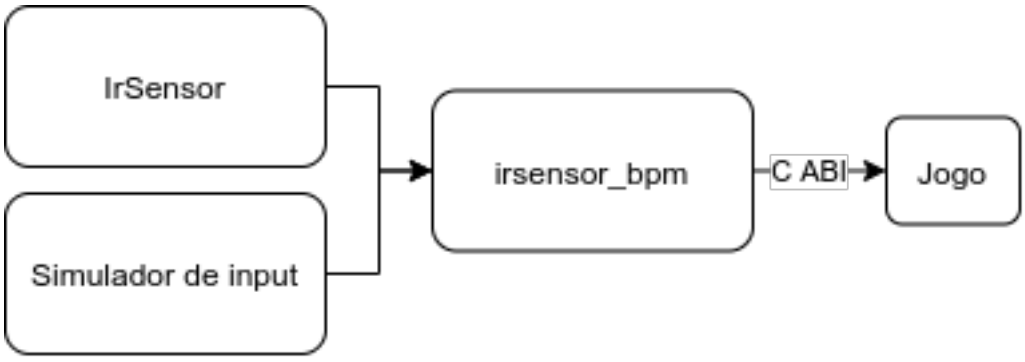


Figura 4.3: Arquitetura do sensor de batimentos cardíacos

## 4.6 Tecnologias e ferramentas usadas

Para o desenvolvimento desta biblioteca utilizou-se as ferramentas referidas na tabela 4.1.

| Nome | Descrição | Utilização no projeto |
|---|---|---|
| C++ | Uma linguagem de programação compilada multi-paradigma e de uso geral. | Linguagem usada para programar a biblioteca. |
| *Nintendo SDK* | Ferramentas usadas para desenvolver para sistemas da *Nintendo*. | Usadas para acessar as *APIs* da *Nintendo Switch* e compilar a biblioteca. |
| *Visual Studio* | Um *IDE* criado pela *Microsoft*. | Usado para executar e testar código. |
| *Clion* | Um *IDE* criado pela *JetBrains*. | Usado como principal ambiente de desenvolvimento para escrever o algoritmo. |

Tabela 4.1: Tecnologias e ferramentas usadas no detector de batimentos cardíacos

## 4.7 Implementação

Para a implementação houve 3 etapas principais para o sucesso deste projeto sendo estas a Receção de dados, a Normalização do gráfico e por último Deteção de ciclos.

### 4.7.1 Receção de dados

O processo começou por conseguir detetar batimentos com o sensor que não pode ser descrito.

Depois de se conseguir quando obtêm-se os valores corretos para detetar batimentos, usou-se a média total da intensidade de luz refletida como um valor único de input da câmara e o tempo passado desde que o programa começou.

Como tal, fez-se um gráfico na consola para verificar os valores, obtendo-se o que se pode ver na figura 4.4.

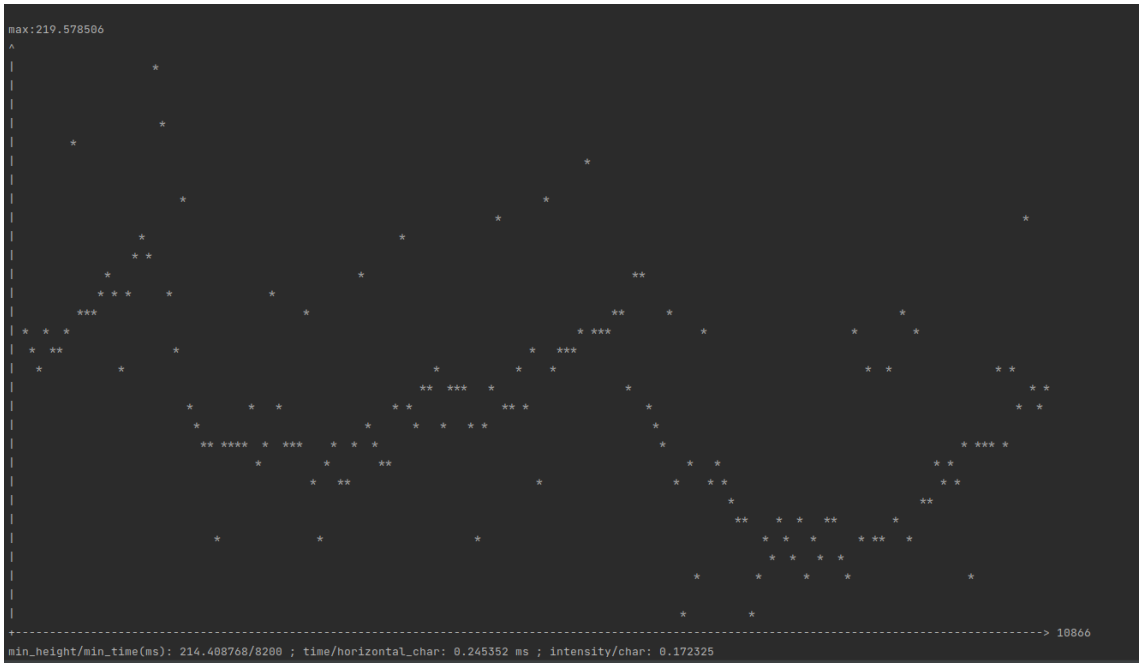


Figura 4.4: Gráfico com as intensidades ao longo do tempo.

### 4.7.2 Normalização do gráfico

Como se pode observar, o gráfico obtido era difícil de analisar. Então, decidiu-se fazer um simples algoritmo de normalização, em que cada ponto seria a média dos últimos 10 pontos. Desta forma conseguiu-se um gráfico melhorado, apresentado na figura 4.5.

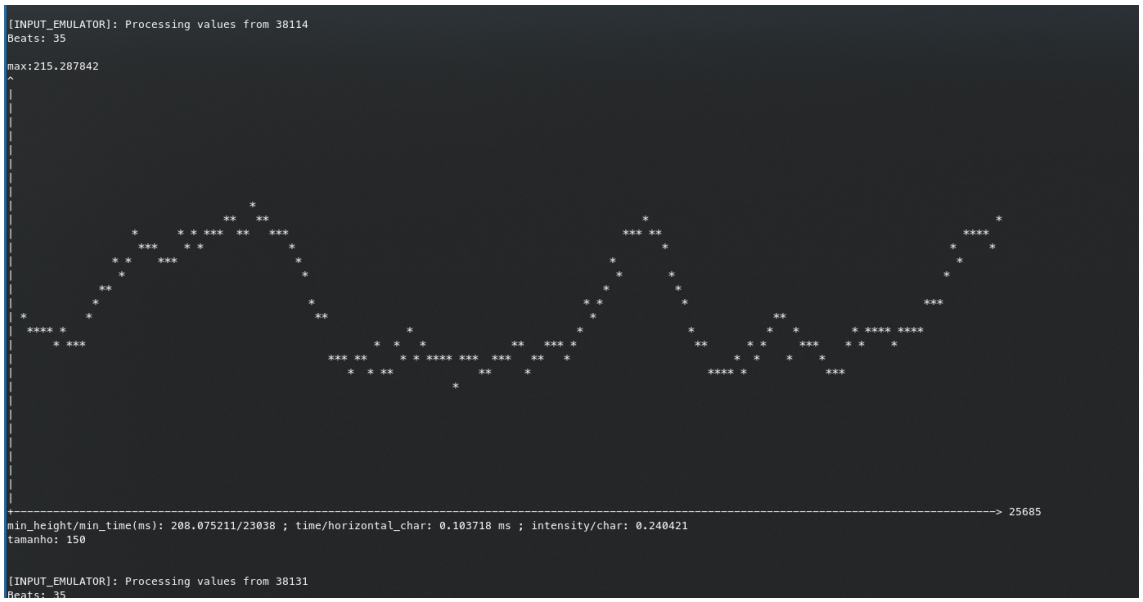


Figura 4.5: Gráfico com as intensidades normalizadas.

Com este simples método de normalização conseguem-se gráficos mais fáceis de analisar, cada vez maior for o número de pontos na media menos pon-

tos existem que saltam tanto mas mais tempo demora a ter se valores usáveis e cada vez mais o gráfico fica com picos mais suaves, o que torna mais difícil descobrir as zonas de picos.

### 4.7.3 Deteção de ciclos

Um ciclo neste contexto consiste na passagem por um mínimo no gráfico, passando por um máximo e finalizando com um mínimo novamente,isto é considerado em termos práticos um batimento.

Concluindo, para calcular os batimentos por minuto faz-se a média das durações dos últimos ciclos.

## 4.8 Testes

A biblioteca foi testada com uma pequena amostra de pessoas. A amostra era de 5 pessoas, 2 crianças e 3 adultos. O sensor conseguia detetar o batimento, pelo polegar da amostra completa.

Figura 4.6: *Smartwatch* com *IR* [32]

Comparou se os valores obtidos com o que o sensor de batimentos de um *smartwatch* (figura 4.6) em que se concluiu que o algoritmo desenvolvido obtinha valores inferiores ao do relógio ao redor de 10 a 20 batimentos por segundo.

A vantagem do algoritmo desenvolvido é ter um tempo de resposta a mudanças de batimento muito mais rápida o que é favorável para o jogo a ser desenvolvido.

## 4.9 Conclusão

A biblioteca não reporta números fiáveis para uma deteção precisa dos batimentos por minuto de uma pessoa que a usa. Porém, serve perfeitamente para saber quando o batimento sobe ou desce, o que pode ser muito bem utilizado num jogo, principalmente para o *DMS*.

Concluído este projeto, o estagiário ganhou experiência com a *Nintendo Switch* e ganhou conhecimentos sobre o seu funcionamento e regras de utilização.

# Capítulo 5

# Conclusão

Dando o estágio por terminado, é necessário fazer um balanço do que foi conseguido, tanto na perspetiva do estagiário como da empresa, e do que será necessário e prioritário desenvolver no futuro para que o projeto desenvolvido evolua. É este o objetivo do presente capítulo.

## 5.1 Objetivos Atingidos

Com a chegada das criptomoedas e a sua exploração e popularidade houve um crescente interesse por *blockchain* e suas áreas de aplicação.

A *Nerd Monkeys* é uma das várias empresas que tiveram interesse no assunto e fizeram a proposta de estágio para que o estagiário, João Gonçalves, estudasse a matéria em causa e chegasse as suas conclusões sobre ela.

O processo estava a correr bem mas estava a ser mais longo do que o esperado, impossibilitando o cumprimento do plano dentro de previsto. Assim, seria previsível que o estagiário acabasse o estágio sem qualquer implementação feita pelo que se decidiu mudar os planos e propor ao estagiário a realização de outras tarefas que mostrassem a sua capacidade de implementação e que aplicassem alguns conhecimentos que podiam interessar ao presente estudo.

O primeiro foi o *AC7ION*, este ajudou o estagiário a ganhar experiência com Rust e como fazer bibliotecas para motores de jogo. Houve um período em que se testava qual a forma correta de fazer *ABI* com *C++* e *Rust* para testar como alguns motores de jogos reagiam e desenvolvendo-se o *AC7ION*.

De seguida desenvolveu se a biblioteca para o *DMS* com bons resultados.

Concluindo, as investigações e desenvolvimentos feitos nesta área durante o estágio, do ponto de vista dos interesses da empresa nesta temática, obteve resultados positivos o que fez com que o estágio fosse um sucesso.

## 5.2 Desenvolvimentos Futuros

O *Game Chaining* é um projeto que tanto a *Nerd Monkeys* como o estagiário querem continuar por isso será algo que vai se retomar quando considerado oportuno. Enquanto isso o estagiário vai continuar a trabalhar com a *Nerd Monkeys* em outros projetos.

# Bibliografia


[1] Amanieu. *Mutex.txt*. URL: https://gist.github.com/Amanieu/. (acedido: 19/09/2021).

[2] *BigchainDB*. URL: https://www.bigchaindb.com/. (acedido: 23/08/2021).

[3] Blockchaingames.fun. *Blockchain Games List*. URL: https://blockchaingames.fun/blockchain-games-list/. (acedido: 01/09/2021).

[4] Evelyn Cheng. *Meet CryptoKitties, the $100,000 digital beanie babies epitomizing the cryptocurrency mania*. URL: https://www.cnbc.com/2017/12/06/meet-cryptokitties-the-new-digital-beanie-babies-selling-for-100k.html. (acedido: 15/09/2021).

[5] chromium.org. *Memory safety*. URL: https://www.chromium.org/Home/chromium-security/memory-safety. (acedido: 17/09/2021).

[6] Catalin Cimpanu. *Microsoft: 70 percent of all security bugs are memory safety issues*. URL: https://www.zdnet.com/article/microsoft-70-percent-of-all-security-bugs-are-memory-safety-issues/. (acedido: 17/09/2021).

[7] iMatix Corporation. *What is wrong with AMQP (and how to fix it)*. URL: https://web.archive.org/web/20190428175704/https://www.imatix.com/articles:whats-wrong-with-amqp. (acedido: 16/09/2021).

[8] egg-mode-rs. *egg-mode*. URL: https://github.com/egg-mode-rs/egg-mode. (acedido: 13/09/2021).

[9] Ethereum. *What is Ethereum?* URL: https://ethereum.org/en/what-is-ethereum/. (acedido: 16/09/2021).

[10] facebook. *zstd*. URL: https://github.com/facebook/zstd. (acedido: 09/09/2021).

[11] The Linux Foundation. *Hyperledger*. URL: https://www.hyperledger.org. (acedido: 20/08/2021).

[12] Jake Frankenfield. *Proof Of Stake (PoS)*. URL: https://www.investopedia.com/terms/p/proof-stake-pos.asp. (acedido: 19/09/2021).

[13] GeeksforGeeks. *NAT Hole Punching in Computer Network*. URL: https://www.geeksforgeeks.org/nat-hole-punching-in-computer-network/. (acedido: 19/09/2021).

[14] geeksforgeeks.org. *What is Sharding?* URL: https://www.geeksforgeeks.org/what-is-sharding/. (acedido: 10/09/2021).

[15] Alan T. Sherman; Farid Javani; Haibin Zhang; Enis Golaszewski. *On the Origins and Variations of Blockchain Technologies*. URL: https://ieeexplore.ieee.org/document/8674176. (acedido: 15/09/2021).

[16] Google. *MessagePack*. URL: https://msgpack.org/. (acedido: 16/09/2021).

[17] hashcash.org. *Hash*. URL: http://www.hashcash.org/. (acedido: 16/09/2021).

[18] Pieter Hintjens. *iMatix will end OpenAMQ support by 2011*. URL: https://web.archive.org/web/20160305155601/http://lists.openamq.org/pipermail/openamq-dev/2010-March/001598.html. (acedido: 16/09/2021).

[19] *Inspector Zé e Robot Palhaço em: Crime no Hotel Lisboa na eShop*. 2020. URL: https://www.nintendo.pt/Jogos/Aplicacoes-de-download-da-Nintendo-Switch/Inspector-Ze-e-Robot-Palhaco-em-Crime-no-Hotel-Lisboa-1842416.html. (acedido: 28/08/2021).

[20] ISEC. *Instituto Superior de Engenharia de Coimbra*. URL: https://www.isec.pt/pt/instituto/#lnkApresentacao. (acedido: 16/09/2021).

[21] *Jogos da Nerd Monkeys na Steam*. URL: https://store.steampowered.com/search/?developer=Nerd%20Monkeys. (acedido: 28/08/2021).

[22] libp2p.io. *Libp2p - NAT traversal status*. URL: https://libp2p.io/implementations/#nat-traversal. (acedido: 11/09/2021).

[23] libp2p.io. *What problems can libp2p solve?* URL: https://docs.libp2p.io/introduction/what-is-libp2p/#what-problems-can-libp2p-solve. (acedido: 02/09/2021).

[24] Jorge Loureiro. *Entrevista: Nerd Monkeys fala sobre a produção de jogos em Portugal, os efeitos de pandemia, e do futuro*. 2021. URL: https://www.eurogamer.pt/articles/2021-04-06-entrevista-nerd-monkeys-fala-sobre-a-producao-de-jogos-em-portugal-os-efeitos-de-pandemia-e-do-futuro. (acedido: 28/08/2021).

[25] Moumita. *Directed Acyclic Graph (DAG)*. URL: https://www.tutorialspoint.com/directed-acyclic-graph-dag. (acedido: 16/09/2021).

[26] Satoshi Nakamoto. ≪Bitcoin: A Peer-to-Peer Electronic Cash System≫. Em: (2008). URL: https://bitcoin.org/bitcoin.pdf.

[27] Nintendo. *Nintendo Switch*. URL: https://www.nintendo.com/switch/system/. (acedido: 13/09/2021).

[28] nlohmann. *json*. URL: https://github.com/nlohmann/json. (acedido: 16/09/2021).

[29] omkarchalke. *NAT Hole Punching in Computer Network*. URL: https://www.geeksforgeeks.org/nat-hole-punching-in-computer-network/. (acedido: 17/09/2021).

[30] Bitcoin Project. *Block Chain - Bitcoin*. URL: https://developer.bitcoin.org/devguide/block_chain.html. (acedido: 07/09/2021).

[31] Roshan Raj. *What is Blockchain Mining?* URL: https://intellipaat.com/blog/tutorial/blockchain-tutorial/what-is-bitcoin-mining/. (acedido: 19/09/2021).

[32] *RELOJ INTELIGENTE H1*. URL: https://satguruonline.cl/reloj-inteligente-h1. (acedido: 14/09/2021).

[33] Ameer Rosic. *What is Ethereum Casper Protocol? Crash Course*. URL: https://blockgeeks.com/guides/ethereum-casper/. (acedido: 09/09/2021).

[34] rust-lang.org. *Extensible Concurrency with the Sync and Send Traits*. URL: https://doc.rust-lang.org/book/ch16-04-extensible-concurrency-sync-and-send.html. (acedido: 17/09/2021).

[35] rust-lang.org. *The Cargo Book*. URL: https://doc.rust-lang.org/cargo/. (acedido: 10/09/2021).

[36] techinsights.com. *Nintendo Switch Teardown*. URL: https://www.techinsights.com/blog/nintendo-switch-teardown. (acedido: 13/09/2021).

[37] *Traffix na eShop*. 2020. URL: https://www.nintendo.com/games/detail/traffix-switch. (acedido: 19/09/2021).

[38] Aaron Turon. *Fearless Concurrency with Rust*. URL: https://blog.rust-lang.org/2015/04/10/Fearless-Concurrency.html. (acedido: 13/09/2021).

[39] *What is IPFS?* URL: https://docs.ipfs.io/concepts/what-is-ipfs/. (acedido: 23/08/2021).

[40] Wikipedia. *Blockchain*. URL: https://pt.wikipedia.org/wiki/Blockchain. (acedido: 19/08/2021).

[41] Wikipedia. *Consenso Distribuído*. URL: https://pt.wikipedia.org/wiki/Consenso_Distribu%C3%ADdo. (acedido: 07/09/2021).

[42] Wikipedia. *Fiat–Shamir heuristic*. URL: https://en.wikipedia.org/wiki/Fiat%E2%80%93Shamir_heuristic. (acedido: 09/09/2021).

[43] Wikipedia. *IOTA*. URL: https://en.wikipedia.org/wiki/IOTA_(technology). (acedido: 16/09/2021).

[44] Wikipedia. *NAT traversal*. URL: https://en.wikipedia.org/wiki/NAT_traversal. (acedido: 11/09/2021).

[45] Wikipedia. *Rust (linguagem de programação)*. URL: https://pt.wikipedia.org/wiki/Rust_(linguagem_de_programa%C3%A7%C3%A3o). (acedido: 10/09/2021).

[46] Wikipeida. *Bitcoin*. URL: https://pt.wikipedia.org/wiki/Blockchain. (acedido: 19/09/2021).

[47] ZeroMQ. *ØMQ - The Guide*. URL: https://zguide.zeromq.org/. (acedido: 16/09/2021).

# Capítulo 6

# Anexos

# Anexo A - Proposta de Estágio

Departamento de Engenharia Informática e de Sistemas

# Proposta de Projeto/Estágio

**Ano Letivo de 2020/2021**
**2º Semestre**

## Game Chaining

**SUMÁRIO**

A seguinte proposta é feita para o candidato a estágio: João Pedro Neves Gonçalves com o número interno da vossa instituição: 2018014306. O projecto a desenvolver na Nerd Monkeys envolve Blockchain (Comunicações e hashing), C++ (programação orientada a objetos) e Unreal Engine (computação gráfica).

Blockchain é uma tecnologia de registo distribuído que visa a descentralização como medida de segurança.
Unreal Engine é uma engine que pode ser usada para jogos ou para outras aplicações gráficas.
O objetivo deste projeto é investigar o que se pode ou não aplicar ao conceito de Blockchain no contexto de um jogo, ou desenvolvimento deste.

**RAMO:** Indicar o(s) ramo(s) em que se enquadra:
X Desenvolvimento de Aplicações
□ Redes e Administração de Sistemas
□ Sistemas de Informação.

## 1. ÂMBITO

O estágio proposto será desenvolvido no âmbito do programa de pesquisa de novos métodos e tecnologias da Nerd Monkeys®.

Aplicam-se as seguintes áreas estudadas durante o curso do estagiário:

POO e PA - Programação avançada em c++

PA e ED - Programação orientada a Blockchain que se baseia em comunicações e algoritmos para tratamento de dados.

FCG - Programação gráfica e orientada para jogos

## 2. OBJECTIVOS

O presente projeto/estágio pretende atingir os seguintes objetivos genéricos:

- Aferir as capacidades da tecnologia Blockchain para além da sua utilização comum atual.
- Criação de um prototipo funcional.

## 3. PROGRAMA DE TRABALHOS

O projecto/estágio consistirá nas seguintes actividades e respectivas tarefas:

- ***T1*** *– Pesquisa e experiência prática* – Durante este período o estagiário deverá realizar uma pesquisa extensiva sobre todas as componentes envolvidas no projecto e experimentação prática para validar os resultados da pesquisa.
- ***T2*** *– Criação de ferramentas* – tendo em vista as etapas T3 e T4.
- ***T3*** *– Criação de protótipo* – Criar um protótipo com base na pesquisa desenvolvida com um objetivo definido pelo orientador.
- ***T4*** *– Desenvolvimento do videojogo* – Com base no protótipo e ferramentas criadas, deverá ser desenvolvido um videojogo em, pelo menos, formato de *vertical slice* (demonstração jogáveis de todas as mecânicas).
- ***T5*** *– Elaboração de relatório de estágio.*

## 4. CALENDARIZAÇÃO DAS TAREFAS

| | | |
|---|---|---|
| INI | | Início dos trabalhos |
| M1 | (INI + 5 Semanas) | Tarefa T1 terminada |
| M2 | (INI + 9 Semanas) | Tarefa T2 terminada |
| M3 | (INI + 12 Semanas) | Tarefa T3 terminada |
| M4 | (INI + 22 Semanas) | Tarefa T4 terminada |
| M5 | (INI + 24 Semanas) | Tarefa T5 terminada |

## 5. LOCAL E HORÁRIO DE TRABALHO

O estágio decorrerá em formato de teletrabalho. Deverá ser desenvolvido entre as 10h e 19h de 2ªf a 6ªf com uma pausa de 1h para almoço.

## 6. TECNOLOGIAS ENVOLVIDAS

C++

Unreal Engine

Blockchain

## 7. METODOLOGIA

O estagiário terá reuniões de apresentação de trabalho, semanais ao orientador da empresa e será acompanhado por este diariamente.

Deverá manter um registo de tarefas realizadas para obter os resultados, sejam estas indicadas pelo orientador ou propostas pelo estagiário como meio de atingir os objetivos.

Dada a natureza prática do projeto, serão criadas várias versões, iteradas ao longo do tempo, que serão utilizadas como base de melhoria construtiva constante.

## 8. ORIENTAÇÃO

Empresa: Nerd Monkeys®

Diogo Resende diogo.vasconcelos@nerdmonkeys.pt
Diretor do estúdio (orientador de estágio)

DEIS-ISEC:

Nome <nome@isec.pt>
Categoria