UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
FACULDADE DE CIÊNCIAS - CAMPUS BAURU
DEPARTAMENTO DE COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

MAURÍCIO SCARELLI ARANTES

BACK-END DE JOGOS EDUCACIONAIS

BAURU
Fevereiro/2022

MAURÍCIO SCARELLI ARANTES

BACK-END DE JOGOS EDUCACIONAIS

Trabalho de Conclusão de Curso do Curso de Ciência da Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho", Faculdade de Ciências, Campus Bauru.
Orientador: Prof. Dr. Wilson Massashiro Yonezawa

BAURU
Fevereiro/2022

Arantes, Maurício Scarelli.
Back-end de jogos educacionais / Maurício Scarelli Arantes, 2022
40 f. : il.

Orientador: Prof. Dr. Wilson Massashiro Yonezawa

Monografia (Graduação)–Universidade Estadual Paulista. Faculdade de Ciências, Bauru, 2022
1. Ciência da computação. 2. Jogos multijogador. 3. API. I. Universidade Estadual Paulista. Faculdade de Ciências. II. Título.

Maurício Scarelli Arantes

Back-end de jogos educacionais

Trabalho de Conclusão de Curso do Curso de Ciência da Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho", Faculdade de Ciências, Campus Bauru.

Banca Examinadora

Prof. Dr. Wilson Massashiro Yonezawa
Orientador
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Departamento de Ciência da Computação

Profa. Dra. Simone Domingues Prado
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Departamento de Ciência da Computação

Prof. Dr. Renê Pegoraro
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Departamento de Ciência da Computação

Bauru, _____ de _____ de _____.

Dedico este trabalho a minha família e todos os amigos que me acompanharam nessa caminhada.

Agradecimentos

Agradeço primeiramente a minha família, que me apoiou e me forneceu toda a estrutura para que isso fosse possível, e em especial a minha mãe Lucinéia (*in memorian*) que sempre foi um grande exemplo de amor e compaixão para mim.

Agradeço ao meu orientador Prof. Dr. Wilson Massashiro Yonezawa, por me guiar durante o desenvolvimento e por todo o esforço empreendido nesse projeto.

Agradeço aos meus amigos, que sempre tornaram meus dias melhores nessa caminhada.

Agradeço ao Centro Acadêmico Ada Lovelace e todos os seus ex-integrantes, que criaram oportunidades para eu abraçar o curso com todo o amor possível.

Por fim, meus agradecimentos a todos os docentes e funcionários da Unesp que forneceram caminhos para eu me desenvolver como pessoa e como profissional.

Games shouldn't only be fun. They should teach or spark an interest in other things.
Hideo Kojima

Resumo

Com o aumento do acesso a meios digitais e tecnológicos, houve uma mudança de paradigmas em diversos campos de conhecimento, humano e sociais, que inevitavelmente atingem também o setor da educação, tornando evidente a necessidade de métodos de ensino que despertem maior interesse e interatividade do aluno. Um dos modelos de ensino explorado pelos educadores tem sido a aprendizagem através de jogos educacionais, fornecendo maior participação dos estudantes enquanto auxiliam na construção de conhecimento. Este projeto propõe a construção de uma aplicação para auxiliar o desenvolvimento de modo multijogador em jogos digitais, voltado principalmente a jogos educacionais, visando facilitar o desenvolvimento de jogos educacionais e incentivando novas formas de ensino.

Palavras-chave: API. Jogo digital. Jogo educacional. Multijogador.

Abstract

With the increase in access to digital and tecnological means, there has been a paradigm shift in several fields of knowledge, human and social, which inevitably affect the educational sector as well, making evident the need for teaching methods that arouse greater student interest and interactivity. One of the teaching models explored by educators has been learning through educational games, providing greater student participation while helping to build knowledge. This project proposes the construction of an application to help the development of multiplayer mode in digital games, aimed mainly at educational games, seeking to make easier the development of educational games and encouraging new ways of teaching.
Keywords: API. Digital game. Educational game. Multiplayer.

Lista de figuras

Lista de abreviaturas e siglas

API	Application Programming Interface
FEN	Forsyth-Edwards Notation
SAN	Standard Algebric Notation
WEB	World Wide Web
JSON	JavaScript Object Notation
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
XML	Extensible Markup Language
SOAP	Simple Object Access Protocol
REST	Representational State Transfer

Sumário

1 Introdução

"A Tecnologia Educacional e o uso das ferramentas digitais e virtuais no âmbito escolar tornaram-se uma necessidade indispensável" (SOUZA; INOCENTE; ARAUJO, 2016, p. 339). A tecnologia se torna cada vez mais presente no cotidiano, inevitavelmente essas mudanças também atingem o setor da educação e tornam evidente a necessidade dos métodos de ensino se modernizarem e gerar maior interatividade para o aluno.

De acordo com Tristão (2010, p.16-19), o desenvolvimento da criança e do adolescente acontece através do lúdico, desenvolvendo conceitos, estabelecendo relações lógicas, pensamento racional e cria relações cognitivas positivas e prazerosas atreladas a aprendizagem. Tendo isso em vista, é perceptível como jogos podem influenciar positivamente na absorção de conhecimento.

Videogames podem ser uma maneira de auxiliar no aprendizado, mesmo que não intencionalmente, exigindo que o jogador tome o papel de um personagem em um mundo complexo, resolva problemas ou gerencie entidades complexas como cidades, exércitos ou civilizações inteiras. Além dos pontos anteriores, qualquer jogo envolve um processo de aprendizagem para ser jogado, entender as regras, o papel do jogador dentro do ambiente, e como ele pode interagir com o mesmo, podendo durar horas e ser desafiador até mesmo para adultos (GEE, 2003).

A pandemia iniciada no ano de 2020 agravou o cenário da educação internacionalmente, ocasionando no aumento de evasão dos estudos e impactando o aprendizado de cerca de 1.6 bilhões de estudantes. Perante esse cenário, ferramentas que auxiliem o acesso de conhecimento podem apoiar professores no processo de ensino.

1.1 Justificativa

A principal motivação para o projeto em questão é a necessidade de mais ferramentas de ensino interessantes para o aluno, principalmente durante o período de pandemia, na qual exige que o professor realize maior acompanhamento do desempenho do aluno durante as aulas e atividades de ensino.

Algumas situações justificam a realização deste projeto, são elas: a dificuldade de implementar recursos de rede em jogos para mais de um jogador, sejam eles baseados em turnos ou em tempo real. Além da baixa disponibilidade de ferramentas em mercado capazes de auxiliar no processo de desenvolvimento de jogos educacionais em rede, apoiando de forma lúdica o ensino.

1.2 Objetivos

Este trabalho tem como objetivo projetar e implementar uma API que comporte a criação de uma plataforma que possa comportar múltiplos jogos educacionais e transformá-los em multijogador em rede de forma simples, armazenando dados das partidas dos alunos para que possam ser consultados e analisados posteriormente pelo professor. Busca-se, também, documentar o protótipo para facilitar o uso e construção de jogos para a plataforma por terceiros.

2 Fundamentação Teórica

2.1 Jogos

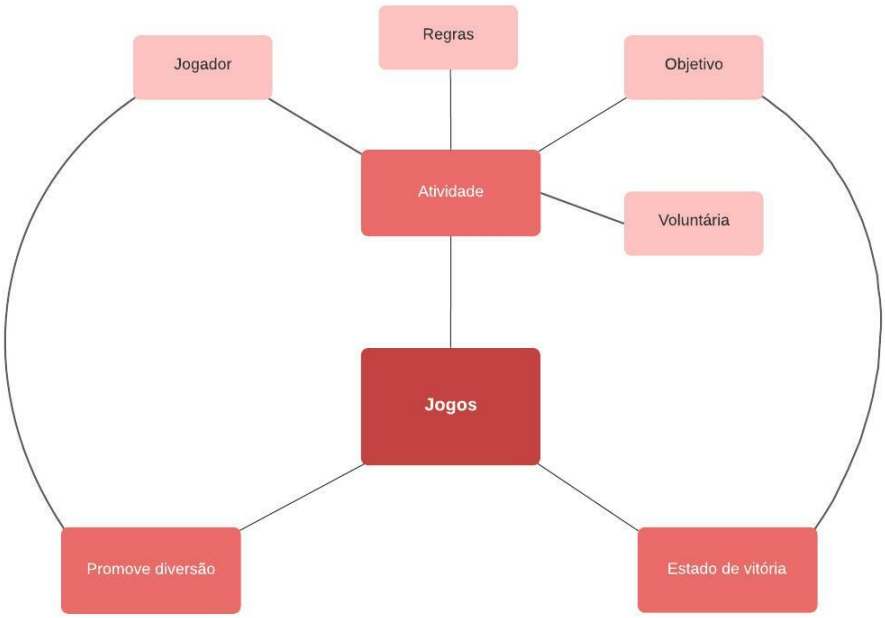
Para criar uma ferramenta auxiliar no desenvolvimento de jogos, é necessário primeiro entender o que são os jogos, como eles funcionam, quais características possuem em comum e como podem auxiliar no ensino.

Uma definição mais acadêmica do que é um jogo vem do filósofo Bernard Suits. Suits (1967) compreende que um jogo é uma tentativa voluntária de engajar em uma atividade utilizando apenas meios permitidos pelas regras, consideradas como barreiras a serem superadas pelo jogador.

Rogers (2012), *game designer* e especialista em jogos digitais, define de forma mais objetiva que um jogo é uma atividade que requer no mínimo um jogador, possua regras que definem e restringem o jogo, haja uma condição de vitória, um objetivo claro.

Tendo em vista as compreensões citadas anteriormente, este trabalho define que um jogo é uma atividade, deve promover diversão ao jogador, e possuir uma condição de vitória. A atividade tem de ser voluntária, requerer no mínimo um jogador, possuir regras e objetivo claro. O objetivo é associado a condição de vitória do jogo. A definição apresentada pode ser ilustrada pelo mapa mental da Figura 1.

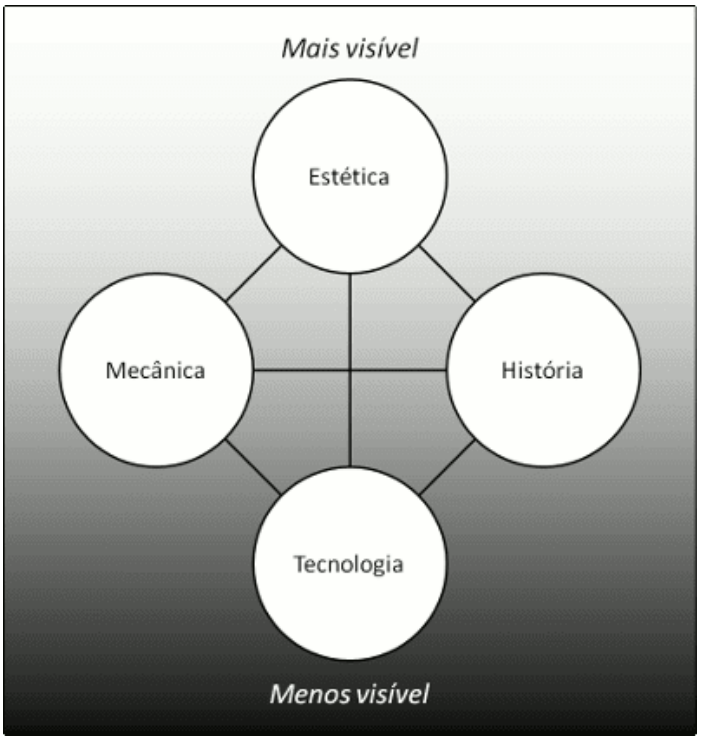
Figura 1 – Definição de um jogo.



Fonte: Elaborada pelo autor.

Definido o que é um jogo, resta saber quais suas características principais. Para Schell (2008), um jogo é formado por quatro elementos básicos que se relacionam entre si, sendo eles a mecânica, a estética, a história e a tecnologia, a chamada tétrade elementar apresentada na Figura 2.

Figura 2 – Tétrade elementar de Schell.



Fonte: Schell (2008)

a) Mecânica - São os procedimentos e regras do jogo, descrevendo objetivo e formas de alcançá-lo. Definido o conjunto de funções mecânicas fundamentais para o jogo, é necessário escolher uma tecnologia que possa suportá-las, a estética que gere destaque para essa mecânica e uma narrativa que traga coerência, mesmo que de maneira excêntrica, ao jogador;

b) Narrativa - Sequência de eventos que se desdobram no jogo. Podendo ser linear e previamente determinada ou ramificada e emergente. Ao querer contar uma história através de um jogo, é preciso escolher uma mecânica que realce a narrativa, além de uma estética que transmita as ideias principais e uma tecnologia apropriada;

c) Estética - Se refere as questões visuais, auditivas e sensações representadas no jogo. A experiência do jogador esta intimamente ligada a estética, dado que é a camada mais visível do jogo e traz o aspecto da imersão. Decidida uma estética, deve-se optar por uma tecnologia capaz de suportar a estética (e que a amplifique), uma mecânica que se comunique de forma a trazer imersão ao jogador e uma narrativa que faça a estética causar o impacto correto nos eventos;

d) Tecnologia - Não se refere apenas a meios eletrônicos, mas a quaisquer materiais e ações que tornem o jogo viável, desde lápis e papéis, peças de plástico ou computadores super potentes. A tecnologia escolhida dita diretamente o que é possível e impossível em um jogo, ela é o meio em que a estética ocorre, a mecânica existe e na qual a narrativa e contada.

A presença de uma mecânica é o que diferencia jogos de outras formas de entretenimento mais lineares, como livros ou filmes, visto que ambos possuem uma tecnologia, narrativa e uma estética.

2.1.1 Jogos digitais

No ponto de vista de Salen e Zimmerman (2004) não há fatores que diferenciam um jogo digital de um jogo que não seja. Para eles, todos os tipos de jogos são apenas sistemas, formados por objetos, atributos, relacionamentos e comportamentos, e o que os distinguem entre si são meros detalhes de *design*, sendo o *hardware* e o *software* somente materiais dos quais os jogos são compostos. Os autores utilizam como exemplo o jogo Tetris, criado por Alexey Pajitnov.

Realizando uma análise sobre o jogo Tetris como um sistema, é possível discernir que a lógica matemática para a construção do jogo existe em um ambiente separado do que é apresentado ao jogador, e as regras mesmo que presentes no *hardware* ou *software* do jogo se referem ao jogador. A importância é dada a interação entre o objeto (o jogo) e o jogador em si, sendo a tecnologia um dos meios de prover a interação.

O papel do *game designer* é desenvolver uma experiência para o jogador através da tecnologia como uma ferramenta, não possuindo a tarefa de criar essa ferramenta, e nem sendo o único fator para criar essa experiência. Salen e Zimmerman (2004) citam quatro pontos em que tecnologias podem tornar as ideias do *game designer* mais vívidas e alinhadas com seus pensamentos:

a) Interatividade imediata - Uma grande qualidade de um jogo digital é fornecer uma resposta (*feedback*) imediata as ações e comandos do jogador, porém, de forma reduzida, visto que as interações são realizadas através do pressionar de botões ou o movimento de um mouse, enquanto em um jogo real, principalmente em um esporte, onde todos os mínimos movimentos corporais são considerados *inputs* que resultam em uma ação;

b) Manipulação de informação - Armazenamento e manipulação de informações são tarefas simples para computadores, os jogos utilizam textos, sons, imagens e animações com frequência, os dados armazenados podem até alterar com tranquilidade as regras e funcionamento do jogo dinamicamente;

c) Sistemas complexos automatizados - Operações físicas de um jogo real que antes precisavam ser realizadas e gerenciadas podem facilmente ser automatizadas em um jogo digital, como por exemplo a movimentação de peças em tabuleiro após um lance de dados, ou o gerenciamento dos turnos de um jogo estratégico de cartas como *Magic: The Gathering* ou *Yu-Gi-Oh!*;

d) Comunicação via redes - Em um jogo real é necessário ter os jogadores fisicamente próximos uns dos outros, essa barreira pode ser facilmente quebrada em um jogo digital onde a comunicação ocorre via rede e os jogadores podem interagir entre si através de *chats* de texto ou até mesmo aúdio, unindo jogadores do mundo todo.

O jogo digital é frequentemente comparado a um sistema, onde o maior uso da tecnologia o diferencia de outros tipos de jogos. A tecnologia podendo ser a nível de *hardware*, como utilizando um computador ou um *console* como o Playstation®, quanto a nível de *software*, através de *game engines* e outros programas destinados ao desenvolvimento de jogos. Jogos digitais compartilham todas as características definidas anteriormente na Figura 1 sobre o que é um jogo.

2.1.2 Jogos digitais na educação

Durante seus estudos sobre a relação dos jogos com a aprendizagem, GEE (2003, 2005), professor e pesquisador em aprendizado baseado em jogos (GBL), conseguiu realizar várias conexões entre esses dois campos. Em seu trabalho intitulado de '*Good video games and good learning*', o autor descreve as características da aprendizagem em bons jogos.

O interesse de Gee (2003) pela relação entre os jogos e a aprendizagem iniciou ao notar seu filho mais novo jogando os jogos digitais. Percebeu que o jogo que o seu filho jogava era complexo, dificil de compreender, exigia muito tempo, possuía muitas tarefas, desafios e dificuldades.

Como educador, percebeu semelhança do jogo com o processo de ensino. O ensino também tem suas dificuldades, é longo e complexo. Porém, segundo ele, o ensino nas escolas não desperta nos estudantes o mesmo interesse quanto os jogos digitais. De acordo com o pesquisador, este é o problema que as escolas enfrentam, como fazer alguém aprender algo difícil, longo e complexo, e ainda assim se divertir em um ambiente motivador?

Analisando os jogos de computadores, o autor descreve que os bons jogos incorporam os bons princípios de aprendizagem, associados ao ensino escolar. Estes princípios, segundo ele, têm suporte nas pesquisas mais recentes das ciências cognitivas (GEE, 2003).

Desafio e aprendizagem são características responsáveis por criar bons jogos de computadores, ou seja, motivantes e divertidos. Os seres humanos por natureza gostam de aprender, porém nem sempre isso se reflete no ensino escolar (GEE, 2005).

O autor fornece como exemplo um paralelo em relação ao ensino de Biologia. De acordo com ele, muitas pessoas acreditam que o ensino de Biologia se resume em decorar fatos, com a finalidade de serem replicados em um teste. No entanto, segundo Gee (2005), décadas de pesquisas têm mostrado que os alunos ensinados desta maneira, através da memorização e reprodução de fatos, por mais que sejam capazes de passar nas avaliações, não são capazes de aplicar os conhecimentos na resolução de problemas. A Biologia não é uma ciência que se resume a um conjunto de fatos. Para o autor, a Biologia pode ser entendida como uma forma de jogo, que determinados tipos de pessoas jogam. Essas pessoas, se envolvem em tipos característicos de atividades, usam ferramentas e linguagem características, mantendo certos valores, ou seja, jogando por um certo conjunto de regras. Este tipo de pessoa que 'joga Biologia' é capaz de reter e usar muitos fatos, mas estes, são considerados menos importantes do que o 'fazer Biologia', que é uma atividade que não se restringe apenas nisso.

Para Gee (2005), aprender sobre um domínio, deve estar relacionado com a forma como jogamos com este domínio. O jogador deve assumir a identidade fornecida pelo jogo, seja ele herói, mago, biólogo ou matemático, sempre jogando pelas regras desse jogo. O jogador, aquele que está no processo de aprendizado sobre um domínio, deve descobrir quais as regras e como podem ser exploradas para realizar os objetivos do jogo.

2.2 API

A sigla API significa Interface de Programação de Aplicações (em inglês, *Application Programming Interface*), sendo um conjunto de rotinas e padrões que permitem a construção e a integração de aplicações garantindo a segurança de dados (MEDEIROS, 2014).

Uma API permite que uma solução ou serviço se comunique com outros de maneira ágil sem a necessidade de conhecer a linguagem ou implementação na qual ela foi feita, apenas as formas de realizar a troca de informações que atenda os requisitos. Ela fornece um conjunto de operações definidas por suas entradas e saídas, e necessariamente permite reimplementações sem comprometer seus usuários.

APIs são disponibilizadas quando uma empresa de *software* tem a intenção de que outros desenvolvedores criem novos produtos associados ao seu serviço ou permita desenvolver sistemas melhores a partir de código reutilizável. Instituições costumam disponibilizar seus códigos acompanhados de uma documentação com instruções de uso para que os usuários possam integrar de maneira simples.

2.2.1 De onde surgiram as APIs?

A ideia de API surgiu muito antes do termo ser cunhado. Para descrever a origem da ideia das APIs é necessário discutir quem inventou as bibliotecas de sub-rotinas, pois não é possível existir uma API sem uma biblioteca de sub-rotinas.

O termo biblioteca de sub-rotinas (em inglês, *subroutine library*) apareceu pela primeira vez na publicação '*Planning and Coding of Problems for an Electronic Computing Instrument-Part II, Volume III*' por Goldstine e Neumann (1948), antes de quaisquer computadores de propósito geral serem construídos, sendo o primeiro relato de um método para criar computadores capazes de armazenar um programa.

A publicação possuia como ideia principal que a maioria dos programas utilizariam operações em comum, e que uma biblioteca de sub-rotinas com essas operações diminuiria a quantidade de código necessário para escrever um programa e a quantidade de erros gerados no processo.

A ideia original de Goldstine e Neumann (1948) no entanto era muito difícil de se tornar algo prático, pois necessitaria de uma extensa intervenção do operador de computador para fazer uso dessas sub-rotinas.

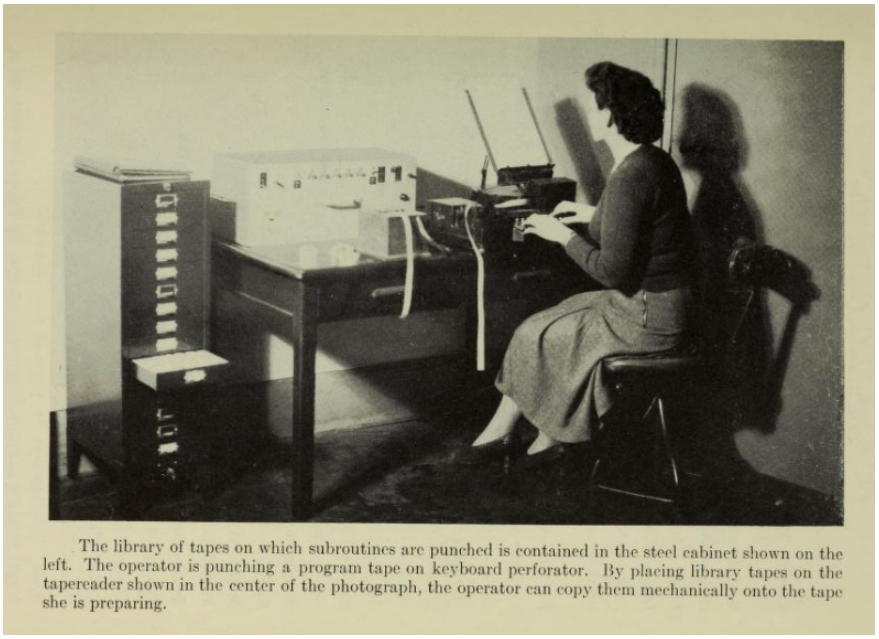
Já em 1949, Maurice Wilkes finalizou a construção do primeiro computador capaz de armazenar programas em memória da história, o EDSAC, anterior ao EDVAC, a máquina criada por Goldstine e Von Neumann em 1951. Os primeiros programas executados por ele eram meros testes e não necessitavam do uso de sub-rotinas, utilizavam uma arquitetura simples, onde as 30 primeiras palavras eram para inicialização (comparado a um *boot loader*) e o programa em si era escrito a partir da posição 30, o código era escrito em *assembly* com a ideia de afastar o usuário de lidar com o binário diretamente. O primeiro programa do EDSAC foi escrito por seu estudante de PhD David Wheeler, responsável também por criar o código de inicialização do EDSAC. (BLOCH, 2018)

Ao tentar criar o primeiro programa não trivial, tentando solucionar uma integral numérica da equação diferencial de Airy, Wilkes percebeu que grande parte do tempo gasto para escrever seus programas era buscando erros no seu próprio código, Wilkes contemplou como solução a implementação de bibliotecas de sub-rotinas, tarefa na qual repassou para Wheeler.

Wheeler finalizou a criação de sua arquitetura de sub-rotinas em setembro de 1949, onde ele criou pseudo-instruções, destinadas não ao computador, mas sim ao compilador para a realocação de sub-rotinas e passagem de parâmetros, sem necessitar da intervenção de um operador. Com isso, Wheeler foi capaz de reduzir a quantidade e instruções para inicialização. Sub-rotinas já eram capazes de chamar outras sub-rotinas dentro de si, mas sem permitir recursão. (BLOCH, 2018)

A biblioteca de sub-rotinas de David Wheeler consistia em um gaveteiro armazenando as fitas de papel perfurado que continham as sub-rotinas, cada gaveta era rotulada e possuía um conjunto de sub-rotinas do mesmo tipo, as quais eram copiadas mecânicamente para a fita do programa principal. A Figura 3 apresenta uma foto do gaveteiro com a biblioteca de sub-rotinas criada por Wheeler.

Figura 3 – Gaveteiro com as sub-rotinas do EDSAC.

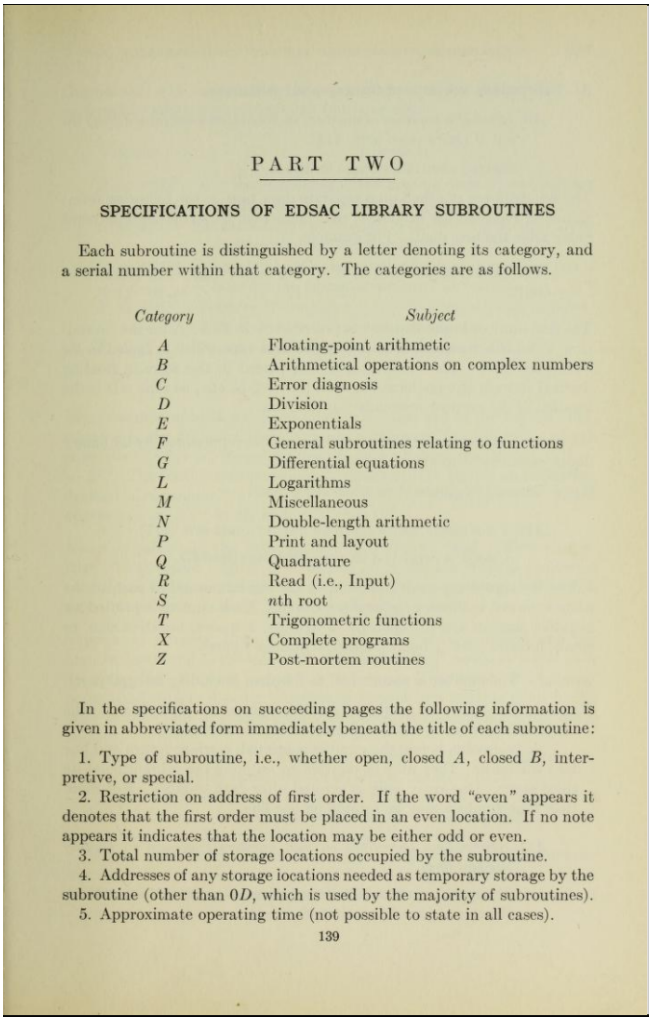


The library of tapes on which subroutines are punched is contained in the steel cabinet shown on the left. The operator is punching a program tape on keyboard perforator. By placing library tapes on the tapereader shown in the center of the photograph, the operator can copy them mechanically onto the tape she is preparing.

Fonte: Wilkes, Wheeler e Gill (1951)

A Figura 4 ilustra o documento que categorizava as sub-rotinas por gaveta, cada gaveta possui um rótulo com uma letra correspondente, e armazena sub-rotinas com o assunto do rótulo, a gaveta com a letra A armazenava sub-rotinas de aritmética com ponto flutuante, a letra L guardava sub-rotinas de funções logarítmicas, como pode ser visto na imagem.

Figura 4 – Categorização das sub-rotinas.



PART TWO

SPECIFICATIONS OF EDSAC LIBRARY SUBROUTINES

Each subroutine is distinguished by a letter denoting its category, and a serial number within that category. The categories are as follows.

Category	*Subject*
A	Floating-point arithmetic
B	Arithmetical operations on complex numbers
C	Error diagnosis
D	Division
E	Exponentials
F	General subroutines relating to functions
G	Differential equations
L	Logarithms
M	Miscellaneous
N	Double-length arithmetic
P	Print and layout
Q	Quadrature
R	Read (i.e., Input)
S	*n*th root
T	Trigonometric functions
X	Complete programs
Z	Post-mortem routines

In the specifications on succeeding pages the following information is given in abbreviated form immediately beneath the title of each subroutine:

1. Type of subroutine, i.e., whether open, closed *A*, closed *B*, interpretive, or special.
2. Restriction on address of first order. If the word "even" appears it denotes that the first order must be placed in an even location. If no note appears it indicates that the location may be either odd or even.
3. Total number of storage locations occupied by the subroutine.
4. Addresses of any storage locations needed as temporary storage by the subroutine (other than 0*D*, which is used by the majority of subroutines).
5. Approximate operating time (not possible to state in all cases).

139

Fonte: Wilkes, Wheeler e Gill (1951)

Foi então que Wilkes, Wheeler e Gill (1951), publicaram '*The Preparation of Programs for an Electronic Digital Computer*', o primeiro livro sobre programação de computadores do mundo, no qual apresentou ao mundo a primeira biblioteca de código reutilizável, a primeira API meticulosamente documentada e outros conceitos, se tornando uma das principais referências na escrita de programas até o surgimento de linguagens de alto nível.

Em 1952, WHEELER apresentou um artigo com ideias chaves sobre a criação de bibliotecas durante a conferência nacional da ACM:

> A preparação de uma biblioteca de sub-rotinas exige um trabalho considerável, é muito maior do que o esforço necessário para codificar a sub-rotina em sua forma mais simples. Normalmente é necessário codificá-la seguindo os padrões da biblioteca, podendo reduzir sua eficiência de tempo e espaço. Pode ser desejável codificá-la de tal maneira que a operação seja generalizada para algum propósito. Contudo, após ter sido codificada e testada, ainda resta a considerável tarefa de escrever uma descrição para que pessoas que não conhecem o interior do código possam utilizá-lo facilmente. Esta última tarefa pode ser a mais difícil. (WHEELER, 1952, tradução nossa)

O termo *Application Program Interface* só foi cunhado a partir do artigo '*Data structures and techniques for remote computer graphics*', quando os autores Cotton e Greatorex (1952), dissertavam sobre a interação entre uma aplicação e o computador em si. Uma interface de aplicação consistente poderia ser mantida mesmo se o computador em si fosse substituído por outro, um sistema flexível e independente de hardware garantiria que a tecnologia avançasse sem tornar o sistema obsoleto, visto que na época para cada novo computador criado era preciso reimplementar as bibliotecas pré existentes e, com novos algoritmos, melhorar a performance de aplicações criadas.

Posteriormente, APIs invadiram múltiplos campos da computação, podendo citar como alguns exemplos a biblioteca padrão utilizada pela linguagem C, chamadas de sistemas em sistemas operacionais, interfaces de linhas de comando, Java, até a criação de *web* APIs.

O conceito de *web* APIs moderno surgiu a partir do ano 2000, o termo API REST apareceu na tese de doutorado, '*Architectural Styles and the Design of Network-based Software Architectures* de Roy Fieldings descrevendo os primeiros conceitos sobre esse tipo de API.

A primeira *web* API a surgir foi criada pela *Salesforce*, publicada em 7 de fevereiro de 2000 no evento IDG Demo, oferecendo processos de automação para clientes seguindo um modelo de Internet como um serviço (*IaaS*). Em seguida, empresas como a *eBay* e *Amazon* disponibilizaram suas próprias *web* APIs (LANE, 2019).

Após 2004, uma nova seleção de APIs surgiram e começaram a se popularizar, sem um valor comercial diretamente associado, eram caracterizadas pelo compartilhamento de informações entre pessoas, fotos e outros dados mais sociais. Pode-se citar como principais APIs, o *Delicious*, *Flickr*, *Facebook* e *Twitter*.

Já em 2006, a *Amazon* disponibilizou ao mundo sua API focada em oferecer serviços de armazenamento de dados de baixo custo, seguindo o modelo de pagamento conforme o uso, o *Amazon Simple Storage* (S3). Seis meses após o lançamento do S3, a empresa disponibilizou um novo serviço chamada de *Amazon Elastic Compute* (EC2), provendo servidores nos quais desenvolvedores poderiam realizar a implantação da infraestrutura de novas aplicações e empresas de tecnologia.

Ambos os serviços integram a plataforma *Amazon Web Services* (AWS) e foram a primeira aparição comercial da computação em nuvem, popularizando o modelo e revolucionando o funcionamento de APIs e da computação como um todo.

2.2.2 Classificações de APIs

Ciente de que APIs existem de diversas formas e propósitos, é interessante definir classificações possíveis a elas.

A primeira forma de classificação a ser abordada é pelo mecanismo de transporte, se é

baseado em hipertexto, código-fonte ou operações binárias:

a) APIs de serviços *web* - APIs baseadas em hipertexto tipicamente utilizando padrões de comunicação como REST, SOAP, JSON-RPC ou XML-RPC. Costumam ser utilizadas na construção de aplicações *web* como parte da arquitetura orientada a serviços (SOA);

b) APIs de código-fonte - Oferecem bibliotecas de objetos, classes, etc. Utilizadas no desenvolvimento de projetos para criar aplicações, normalmente seguem padrões como J2EE ou .NET;

c) APIs legado - Conjunto de recursos ou protocolos com a finalidade de se comunicar com aplicações mais antigas.

Podem ser classificadas também pelo tipo de acesso:

a) APIs públicas - Também conhecidas como APIs abertas, estão disponíveis para desenvolvedores ou outros usuários de forma pública para consumo e com poucas restrições, podendo exigir registro, uma chave de API, ou outro tipo de autenticação, ou até mesmo ser completamente aberta. Tem foco principal no acesso por usuários externos, normalmente a empresa ou desenvolvedor responsável possui um site ou portal no qual disponibiliza a documentação para o uso;

b) APIs privadas - Também conhecidas como APIs internas, são omitidas de usuários externos e podem ser acessadas apenas por sistemas internos de uma empresa e as equipes responsáveis pelos mesmos;

c) APIs de parceiros - Um meio termo entre as anteriores, podem ser acessadas por pessoas de fora da organização desde que recebam permissões exclusivas para tal, o acesso é concedido por ou para os parceiros de negócio envolvidos;

d) APIs compostas - Combinam vários dados ou APIs de exposição diferentes. Desenvolvidas através de recursos de orquestração de APIs.

Pela finalidade:

a) APIs de dados - Fornecem acesso a ferramentas de CRUD (*create*, *read*, *update*, *delete*), conjuntos de dados provenientes de bancos de dados ou provedores de nuvem;

b) APIs de serviços internos - Expõem serviços ou processos internos ou algumas ações complexas;

c) APIs de serviços externos - Serviços de terceiros que podem ser integrados ao seu sistema;

d) APIs de experiência de usuário - Fornecem dados necessários para serem consumidos por seu público-alvo, sejam aplicativos móveis, sites ou portais internos.

E pela arquitetura escolhida, seja REST, SOAP, RPC, baseada em eventos/streaming, entre outras possíveis arquiteturas.

2.2.3 Vantagens e desvantagens

Pode-se destacar algumas vantagens no uso de APIs:

a) A troca de informações é feita de maneira segura, visto que os dados podem ser acessados após realizar uma sequência de requisições por um protocolo de autorização padrão;

b) Possibilidade de monitorar os acessos a API, conhecendo quem, quando e onde foram feitas as requisições e quais as informações foram solicitadas, trazendo auditabilidade a aplicação;

c) Restrição da quantidade de informações trafegadas de acordo com a necessidade ou autorização do sistema;

d) Possibilidade de reimplentar algoritmos e realizar refatorações sem impactar a forma de comunicação do usuário;

e) No caso de substituição de uma máquina, seja por quebra ou por melhoria tecnológica, não é necessário recriar todas as funcionalidades para a máquina nova;

f) Traz maior facilidade para o uso dos recursos por um terceiro que desconheça as minúcias do código-fonte;

g) Mais agilidade na manutenção da aplicação.

E citar como desvantagens:

a) Requer tempo e custo para a implementação;

b) Em caso de quedas de sistema, pode ocasionar a quebra do funcionamento de outros sistemas que dependem dessa API;

c) Necessita de manutenções periódicas para garantir a saúde do ecossitema;

d) Dificuldade na padronização da interface e na escrita de uma documentação clara e eficiente.

2.2.4 Exemplos de APIs

Alguns exemplos de APIs populares podem ser citados, como por exemplo a API do *Facebook*, utilizada para efetuar login em diversos sistemas e trazer informações pessoais de cadastro, ou a API do Google Maps que possibilita ao desenvolvedor e ao usuário executar operações como exibir locais no mapa ou traçar rotas entre pontos (FERNANDES, 2018).

2.2.5 APIs em jogos

Naturalmente, jogos também utilizam APIs em seu desenvolvimento, principalmente para operações básicas como procedimentos de renderização gráfica, desde os próprios *drivers* gráficos, de empresas como NVIDIA, AMD ou Intel, como APIs de mais alto nível, como DirectX, OpenGL e Vulkan. APIs de áudio como a OpenAL ou XAudio, *inputs* de controles como XInput, entre outros.

Softwares de desenvolvimento de jogos, *game engines* como Unity, Unreal, Godot e *GameMaker*, além de fazerem uso de APIs como as citadas anteriormente, por si só também disponibilizam APIs para auxiliar os desenvolvedores na criação de seus jogos, implementado funções como geração de imagens, movimento e fisicas básicas.

Além das APIs utilizadas na construção dos jogos, é possível também disponibilizar APIs que expõem dados dos jogos ou das partidas, como por exemplo a *Riot Games API*, a qual pode fornecer dados sobre jogadores de *League of Legends* (ou outros jogos da empresa), como nível, dano em uma determinada partida, ranque, etc.

Em suma, as possibilidades de uso de APIs em jogos são extensas e possívelmente infinitas.

2.3 Representação digitais de jogadas

Ao iniciar um jogo, cada jogador é apresentado às possíveis formas de interação estratégica que ele possui e como impactam o ambiente ou outros jogadores, de forma que ele tenha autonomia de tomar suas decisões e obter vantagem de acordo com seus movimentos (FIANI, 2009, p.41-50).

Ações podem ser desde atos mais simples, como a rolagem de dados, movimentar uma peça para um lado ou para outro, até operações mais complexas atreladas a inúmeras regras em um jogo de cartas por exemplo.

Para tornar possível a criação de um jogo digital, é necessário pensar em uma forma de recriar e representar o estado atual do jogo ou alcançá-lo a partir de uma configuração inicial e a descrição das ações tomadas por cada jogador (e pelo próprio jogo, caso necessário), principalmente multijogador, onde o ambiente de ambos devem possuir as mesmas informações para que não haja uma dissonância entre o que foi realizado por um jogador e o que foi revelado para o outro.

Pensando dessa forma, percebe-se então que existem duas formas mais comuns de se representar o estado do jogo e as jogadas feitas, enviar a cada jogada o estado completo do jogo, ou registrar apenas os movimentos feitos.

2.3.1 Registrando o estado completo

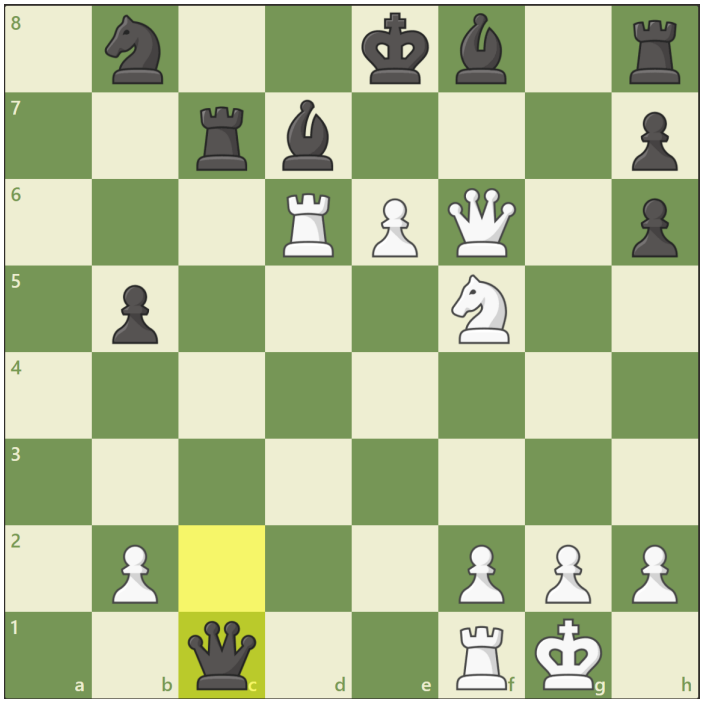
É possível registrar o estado completo de um jogo a todo passo dado. Utilizando o xadrez como exemplo, pode-se gravar todos os posicionamentos realizado no tabuleiro, por meio de algum código de referência as peças e casas presentes no jogo, a representação mais comum em motores de xadrez é a Notação Forsyth-Edwards (conhecida como FEN), onde é representado o tabuleiro, qual o próximo jogador, e outras regras adicionais (CHESS.COM, 2020).

Na representação FEN, cada tipo de peça possui uma letra correspondente, se for minúscula é uma peça preta, se for maiúscula é uma peça branca. 'p' representa o peão, 'r' representa a torre, 'n' o cavaleiro, 'b' o bispo, 'q' a rainha, 'k' o rei.

Além disso são utilizados números de 1 a 8 entre as peças para indicar a quantidade de espaços vazios entre elas (ou do início da linha do tabuleiro). As peças e espaços são registrados em cada linha, a separação das linhas é feita com o caractere '/'.

As linhas são contadas seguindo a orientação do tabuleiro, das peças pretas para as brancas (*rank* 8 a 1). Após o registro de todas as peças, informa-se qual o próximo jogador utilizando uma letra, sendo 'w' para a vez das brancas ou 'b' para a vez das pretas.

Figura 5 – Exemplo de posicionamento de peças.



Fonte: Elaborada pelo autor.

"1n2kb1r/2rb3p/3RPQ1p/1p3N2/8/8/1P3PPP/2q2RK1 w" é a sequência de FEN que representa o tabuleiro apresentado na Figura 5, onde o trecho "1n2kb1r"da sequência descrita corresponde a linha 8 com o seguinte posicionamento: um espaço vazio (1), um cavalo preto (n), dois espaços vazios (2), um rei preto (k), um bispo preto (b), um espaço vazio (1) e uma torre preta (r).

2.3.2 Registrando as jogadas

Novamente aplicando o xadrez como exemplo, é possível também gravar cada movimento na rodada, é comum para descrever o decorrer das partidas e ensinar jogadas clássicas, a Notação Algébrica Padrão (SAN, Standard Algebric Notation) sendo a mais comum e adotada pela FIDE (sigla da Federação Internacional de Xadrez).

A representação das peças como letras continua da mesma forma apresentada anteriormente, porém não há diferenciação entre as peças pretas e brancas (todas as peças são maiúsculas), visto que pode ser discernido pela ordem das jogadas, além de que a letra representando o peão é ocultada.

A notação padrão informa primeiro a peça utilizada, e em seguida a casa de destino. Caso haja uma captura no movimento é colocado um 'x' entre a letra da peça e o destino. Caso o movimento ocasione em um xeque, no final da sentença é colocado um símbolo '+', e o xeque-mate '#'.

O movimento da rainha preta em C2 para C1, realizado na Figura 5, pode ser representado em SAN com a mensagem "Qc1".

3 Ferramentas

Nesta seção serão tratadas algumas ferramentas utilizadas no processo de desenvolvimento do projeto.

3.1 JavaScript

JavaScript é uma linguagem de programação frequentemente utilizada no desenvolvimento *frontend* de páginas *web*, sendo aplicada a documentos HTML, permitindo interações dinâmicas e flexíveis entre sites. (GUININ, 2019)

Ela permite desde a realização de tarefas básicas, como por exemplo ajustar *layouts* até realizar atividades que demandam mais conhecimento como criar jogos ou animações 3D.

Por ser uma linguagem compacta e flexível, se tornou comum o desenvolvimento de novas ferramentas e APIs utilizando essa tecnologia.

3.2 Node.js

Node.js pode ser resumido como uma plataforma de execução JavaScript *server-side*, tornando possível a criação de aplicações JavaScript executando como uma aplicação *standalone*, desvinculando a necessidade de um navegador para a utilização. A plataforma é baseada no V8, o motor e interpretador de código aberto do Google Chrome. (DUARTE, 2021)

De acordo com Souza (2020), a característica que diferencia a plataforma das demais é de que sua execução é *single-threaded*, ou seja, uma única linha de execução de código. Unindo este fator a leveza do ambiente, é possível criar milhões de conexões simultâneas para lidar com requisições, sendo altamente escalável.

Algumas das vantagens oferecidas pelo Node.js são sua leveza, comunidade ativa, e alta produtividade. A ferramenta é ideal para criação de APIs, sistemas de *chat*, *IoT*, e *backend* de jogos.

3.3 Typescript

Typescript é uma linguagem de código aberto desenvolvido pela Microsoft, construído em cima do Javascript para adicionar recursos de tipagem estáticas a linguagem base, além de acrescentar suporte ao paradigma da Programação Orientada a Objetos (POO) e elevando o nível de aplicações Javascript. (CAVALCANTE, 2021)

A linguagem surgiu para suprir deficiências existentes no Javascript devido a sua tipagem por inferência, dificultando projetos cujos parâmetros são bem definidos e possuem tipagem forte, além da necessidade de criar interfaces e relações de herança como em linguagens tradicionais, possibilitando a criação de arquiteturas mais eficientes e de acordo com princípios de engenharia de *software*, como SOLID[1].

A coerção de tipo de variável com o operador de igualdade '==' e a comparação de valor dentro de um intervalo, como mostra o Código 3.1, são exemplos de erros que o Typescript se propõe a corrigir.

```
1 if ("" == 0) {
2   // Verdadeiro, o comparador '==' coage ambos os valores a
3   // terem o mesmo tipo antes da comparacao.
4 }
5 if (1 < x < 3) {
6   // Verdadeiro para qualquer valor de x.
7 }
```

Código 3.1 – Exemplo de código em Javascript.

Embora seja uma linguagem a parte, na hora de compilar o código, todo Typescript é também transpilado para Javascript, trazendo compatibilidade com todos os ambientes que rodam Javascript nativamente. (TYPESCRIPT, 2012)

3.4 Socket.IO

Socket.IO é uma biblioteca que permite a comunicação em tempo real, bi-direcional e baseada em eventos entre cliente e servidor. Consiste em um servidor Node.js e uma biblioteca cliente disponível para diversas linguagens (Javascript, Java, C++, etc).

O cliente tenta através da biblioteca estabelecer a conexão de um *WebSocket*[2] e caso ele seja impossibilitado, irá abrir uma conexão via HTTP *long polling* (Servidor segura a requisição até ser capaz de enviar novos dados, e assim que recebido, o cliente abre uma nova requisição).

Este tipo de comunicação permite que a comunicação em um jogo ocorra de maneira eficiente e em tempo real.

[1] SOLID são cinco princípios de engenharia de *software* para escrever códigos mais limpos, diminuindo acoplamentos, facilitando refatoração e reaproveitamento de código (PAIXAO, 2019).

[2] *WebSocket* é uma API que possibilita abrir uma sessão de comunicação interativa entre cliente e servidor com respostas orientadas a eventos, trazendo o conceito de soquetes para o ambiente *web* (MOZILLA, 2020).

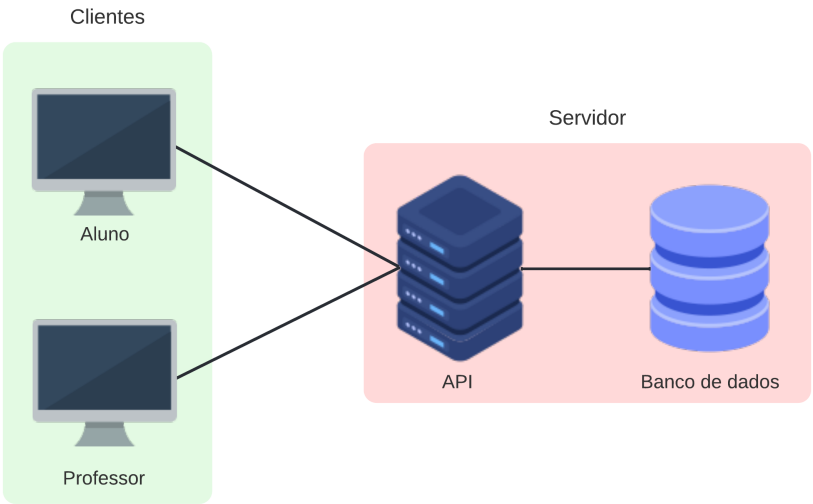
4 Desenvolvimento

Nesta seção serão apresentadas as etapas do projeto e como foram desenvolvidas. Todo o código desenvolvido pode ser encontrado no *Github*[1] do projeto.

4.1 *Design* da aplicação

A aplicação proposta possui dois tipos de usuário, aluno e professor, ambos se comunicam com a API através de um programa cliente, podendo ser um jogo desenvolvido para a plataforma, ou um *software* de gerenciamento destinado ao professor no acompanhamento dos alunos. A API possui diversas rotas de comunicação, os endereços URI nos quais serão chamados pelo programa cliente para realizar operações entre cliente-servidor. O servidor possui um banco de dados *Postgres* no qual será acessado dependendo da operação feita pela API, inserindo ou retornando dados de usuário, jogo ou sala de aula. A relação descrita entre cliente e servidor é demonstrada pela Figura 6.

Figura 6 – Definição de cliente e servidor



Fonte: Elaborada pelo autor.

A funcionalidades planejadas como conteúdo da API são:

a) Criação de usuário - Para criar uma partida em um jogo é necessário possuir uma conta cadastrada, sendo utilizada para armazenamento dos dados das partidas do aluno, e criação de vínculo com a turma de um professor;

[1] https://github.com/mauscarelli/tcc-backend

b) Login - Autenticação do usuário previamente criado para realizar as demais operações no sistema;

c) Criar uma sala de aula - O professor tem a possibilidade de criar uma entidade apelidada de sala de aula, tendo acesso aos dados de partidas dos alunos que fizerem parte dessa entidade;

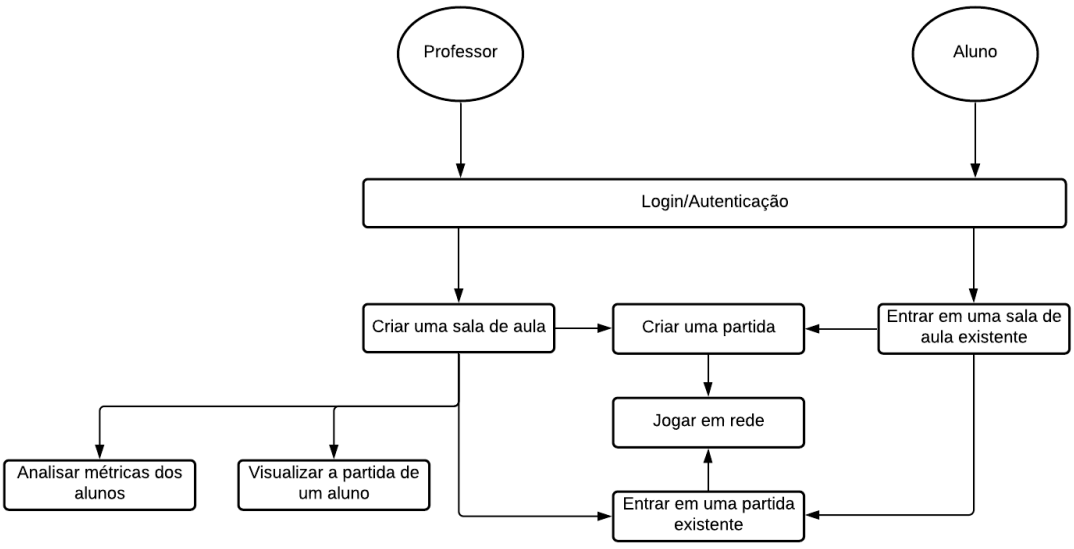
d) Entrar em uma sala de aula - O aluno consegue entrar em uma sala de aula criada pelo professor a partir de um código fornecido pelo professor, permitindo o educador visualizar seu registro de partidas;

e) Criar uma partida - O aluno ou professor poderá criar uma sala de um jogo educacional desenvolvido para a plataforma, fornecendo acesso para que outros usuários possam participar do jogo;

f) Entrar em uma partida - O aluno ou professor é capaz de participar de jogo em uma sala criada por outro usuário.

O fluxo das funcionalidades descritas acima pode ser analisado na Figura 7.

Figura 7 – Diagrama de fluxo de uso.



Fonte: Elaborada pelo autor.

4.2 Criação do protótipo

As atividades chave no desenvolvimento da API foram a sua integração com o banco de dados Postgres, a criação dos módulos de comunicação baseado em eventos em paralelo com as comunicações por solicitação-reposta.

Foi utilizada a biblioteca *pg* para comunicação com o banco de dados. Para a criação das rotas da aplicação e gerenciamento das requisições HTTP foi utilizado o *framework* Express, enquanto as comunicações via *WebSockets* a biblioteca Socket.IO.

Parte do processo de iniciação do servidor para recepção e manipulação de chamadas pode ser observado pela Figura 8, onde são realizadas as configurações, iniciando uma aplicação *web* com o Express, definem-se as rotas de comunicação da aplicação a partir do objeto *router* importado do arquivo routes.ts, cria-se um servidor HTTP para a aplicação com a função *createServer*, e por fim, abre conexão com a base de dados. Após as configurações iniciais, a função *start* é chamada pelo arquivo que instânciou a classe *Server*, fazendo com que o servidor HTTP ouça as chamadas recebidas na porta definida, e criando o servidor de comunicações baseadas em eventos do Socket.IO, com os caminhos de comunicação definidos no arquivo socket.ts.

Figura 8 – Código-fonte da iniciação do servidor

```
1  import 'dotenv/config'
2  import express, { Application } from 'express'
3  import 'reflect-metadata'
4  import pool from './configs/dbconfig'
5  import { createServer, Server as HttpServer } from 'http'
6  import { Server as SocketServer } from 'socket.io'
7  import { socketInit } from './routes/socket'
8  import router from './routes/routes'
9
10 class Server {
11     private app: Application;
12     private httpServer: HttpServer;
13     private io: SocketServer;
14
15     constructor () {
16       this.config()
17       this.dbConnect()
18     }
19
20     private config () {
21       this.app = express()
22       this.app.use(express.urlencoded({ extended: true }))
23       this.app.use(express.json({ limit: '1mb' }))
24
25       this.app.use(router)
26       this.httpServer = createServer(this.app)
27     }
28
29     private dbConnect () {
30       pool.connect((err: Error) => {
31         if (err) throw err
32         console.log('Connected')
33       })
34     }
35
36     public start = (port: number) => {
37       return new Promise((resolve, reject) => {
38         this.httpServer.listen(port, () => {
39           this.io = socketInit(this.httpServer)
40           resolve(port)
41         }).on('error', (err: Object) => reject(err))
42       })
43     }
44 }
45
46 export default Server
```

Fonte: Elaborada pelo autor.

4.3 Rotas

As rotas são os pontos de acesso disponibilizados de uma aplicação para que o cliente possa consumir os serviços providos por uma API REST. Uma rota pode ser acessada a partir

da URL base do projeto complementada com a operação desejada e eventuais parâmetros necessários, além do método HTTP correspondente. O protótipo utiliza objetos JSON como corpo de entrada e saída padrão das requisições. As principais rotas REST que constituem a aplicação são:

- '/user/createAccount' - realizado através do método POST. Cria um usuário na base de dados da aplicação para ocorrer as demais interações com o servidor. Recebe como entrada um objeto JSON com os dados *username*, *password*, *firstName*, *lastName* e *age*. Em caso de sucesso, retorna um código HTTP 200 ("OK"), juntamente de um objeto contendo um booleano *success* verdadeiro e *string* id sendo o id do usuário. De forma alternativa, pode retornar um erro 400 ("*Bad Request*") indicando a falta de uma informação na entrada ou usuário já cadastrado na base.

- '/user/login' - realizado através do método POST. Autentica um usuário para realizar novas operações. Recebe como entrada um objeto JSON com os dados *username* e *password*. Em caso de sucesso, retorna um código HTTP 200 ("OK"), juntamente de um objeto contendo um booleano *success* verdadeiro e um objeto *data* com id, *username*, *firstName*, *lastName* e *age*. De forma alternativa, pode retornar um erro 400 ("*Bad Request*") indicando a falta de uma informação na entrada ou usuário não encontrado, pode também emitir erro 401 ("*Unauthorized*") caso a senha informada seja conflitante com o registro.

- '/classroom/create' - realizado através do método POST. Cria uma entidade sala de aula para estabelecer relação entre aluno e professor. Recebe como entrada um objeto JSON com os dados *schoolName*, *className*, *professorId*, sendo este último o id de usuário obtido anteriormente na autenticação. Em caso de sucesso, retorna um código HTTP 200 ("OK"), juntamente de um objeto contendo um booleano *success* verdadeiro e *string* id sendo o id da sala de aula. De forma alternativa, pode retornar um erro 400 ("*Bad Request*") indicando a falta de uma informação na entrada ou erro 401 ("*Unauthorized*") caso o usuário não tenha sido encontrado.

- '/classroom/join' - realizado através do método POST. Associa o aluno a sala de aula, criando relação entre aluno e professor. Recebe como entrada um objeto JSON com os dados *userId*, obtido durante a autenticação, e *classroomId*, fornecido pelo professor. Em caso de sucesso, retorna um código HTTP 200 ("OK"), juntamente de um objeto contendo um booleano *success* verdadeiro. De forma alternativa, pode retornar um erro 400 ("*Bad Request*") indicando a falta de uma informação na entrada, classe não encontrada, ou aluno previamente cadastrado na sala, ou erro 401 ("*Unauthorized*") caso o usuário não tenha sido encontrado.

- '/classroom/:classroomId' - realizado através do método GET. Lista os estudantes cadastrados na sala de aula. Recebe como entrada id da classe via parâmetro descrito

na rota substituindo ':classroomId'. Em caso de sucesso, retorna um código HTTP 200 ("OK"), juntamente de um objeto contendo um booleano *success* verdadeiro e um *array data* listando objetos comid, *firstName* e *lastName*. De forma alternativa, pode retornar um erro 400 ("*Bad Request*") indicando a falta de uma informação na entrada.

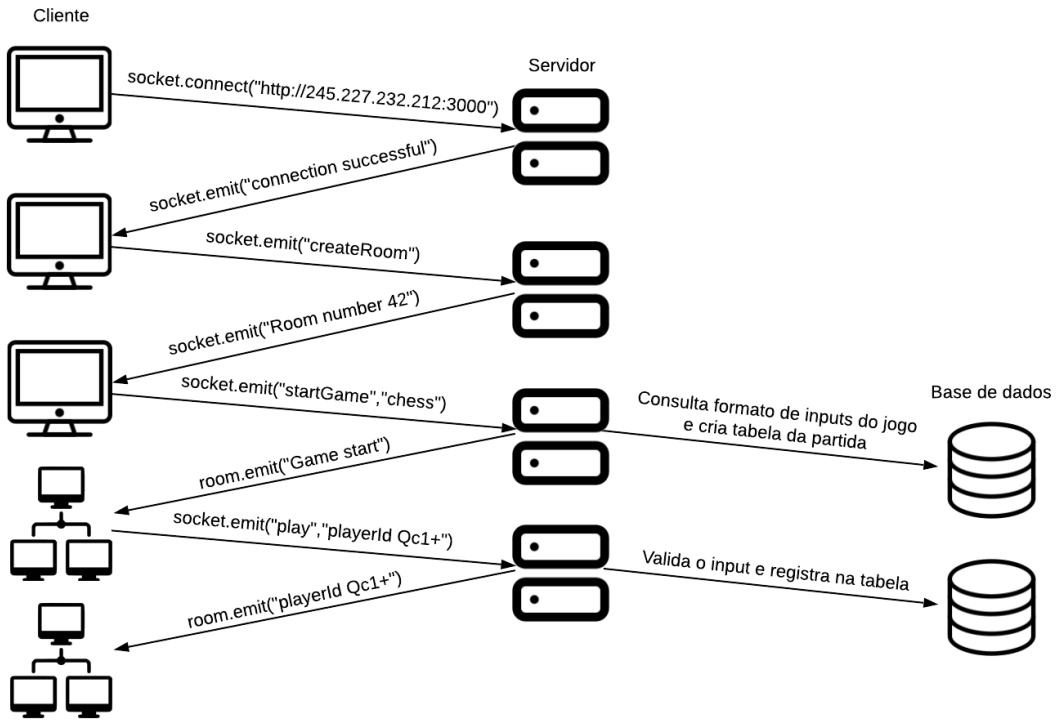
Após a operação de login através de comunicação solicitação-reposta (*Request-response*), o cliente deve realizar uma série de solicitações (chamados de eventos), na rota '/games', por meio de um *WebSocket* ou HTTP *long polling*, para jogar uma partida em rede. Os principais eventos da rota '/games' que constituem a aplicação são:

- 'connection' - Ocorre naturalmente ao abrir uma conexão com a rota, informa o servidor que um novo usuário se conectou. Em caso de sucesso, retorna para o cliente um evento '*serverToClient*' com uma mensagem de sucesso.
- 'create-room' - Cria uma sala para iniciar um jogo, e carrega as informações necessárias para a sala, como número mínimo e máximo de jogadores. Recebe como entrada um objeto usuário, possuindo id, *username*, *firstName* e *lastName*, e uma *string* com o id do jogo (previamente cadastrada na plataforma e repassada ao criador do jogo). Em caso de sucesso, retorna um evento '*success*' com o id da sala. Caso não tenha salas disponíveis ou houve uma falha ao criar o jogo, é emitido um evento 'err' com uma mensagem de erro.
- 'join-room' - Entra na sala informada para participar do jogo. Recebe como entrada uma *string* com o id da sala e um objeto usuário, possuindo id, *username*, *firstName* e *lastName*. Em caso de sucesso, dispara um evento '*newPlayer*' para todos os jogadores da sala com a mensagem de que um jogador entrou, e retorna um evento '*joined-room*' para o cliente com a lista de todos os jogadores da sala e uma mensagem de sucesso. Caso falhe ao entrar na sala é emitido um evento 'err' com uma mensagem de erro.
- 'leave-room' - Sai da sala informada. Recebe como entrada uma *string* com o id da sala. Não possui retorno.
- 'start-game' - Inicia a partida na sala para todos os jogadores, criando dinamicamente uma tabela para armazenar os dados da partida. Recebe como entrada uma *string* com o id da sala, um objeto usuário, possuindo id, *username*, *firstName* e *lastName*, e um objeto modelo com todas as propriedades que o jogo deseja registrar em cada jogada. Em caso de sucesso, dispara um evento '*game-started*' para todos os jogadores da sala com a mensagem de que o jogo foi iniciado. Caso falhe ao iniciar a partida é emitido um evento 'err' com uma mensagem de erro.
- 'play' - Realiza uma jogada, registra o movimento na tabela criada no evento anterior e retransmite para os outros jogadores. Recebe como entrada uma *string* com o id da sala,

um objeto usuário, possuindo id, *username*, *firstName* e *lastName*, e um objeto com os dados da jogada em propriedades descritas no objeto modelo. Em caso de sucesso, dispara um evento '*played-move*' para todos os jogadores da sala com um objeto contendo dados do jogador e do movimento. Caso falhe ao realizar a jogada é emitido um evento 'err' com uma mensagem de erro.

O fluxo de comunicações pretendido para o jogo ser viável é similar ao demonstrado pela Figura 9, onde o usuário realiza requisições ao servidor para criar uma sala, iniciar uma partida, e fazer uma jogada, equanto o servidor faz processamentos internos, operações no banco de dados, e emite respostas ao solicitante ou a sala na qual o solicitante participa.

Figura 9 – Fluxo de comunicação das partidas.



Fonte: Elaborada pelo autor.

As Figuras 10, 11 e 12, exibem testes de criação de sala e realização de jogadas em diferentes pontos de vista. A Figura 10 apresenta o ponto de vista do servidor, onde é iniciada a aplicação, dois usuários se conectam, o primeiro cria uma sala, o segundo entra na sala criada e então solicita para iniciar o jogo, cada jogador realiza um movimento.

Figura 10 – Partida teste observada pelo servidor.

```
C:\Users\mausc\OneDrive\Documentos\projetos\tcc-backend>npm run local

> tcc-backend@1.0.0 local C:\Users\mausc\OneDrive\Documentos\projetos\tcc-backend
> ts-node-dev -r tsconfig-paths/register --respawn --transpile-only --ignore-watch node_modules --no-notify src/app.ts

[INFO] 04:27:00 ts-node-dev ver. 1.1.8 (using ts-node ver. 9.1.1, typescript ver. 4.3.5)
Socket started
Running on port 3000
Connected
socketId XQk94n800KksetGGAAAB connected
Room number 1 created
socketId 3PzF6lPogA1xoH0iAAAD connected
joining room 1
starting game on room 1
move received on room 1
move received on room 1
```

Fonte: Elaborada pelo autor.

A Figura 11 apresenta o ponto do primeiro usuário, abrindo conexão com o servidor, emite um evento para criar uma sala, recebe uma mensagem de que o segundo jogador entrou na sala e então solicita para iniciar o jogo, cada jogador realiza um movimento.

Figura 11 – Partida teste observada pelo primeiro cliente.

```
C:\Users\mausc\OneDrive\Documentos\projetos\tcc-backend\src\__tests__>node client1.js
Connection successful
Room number 1
teste entrou na sala
Jogo iniciado
player: {"id":"38c5e172-67f7-b4dd-2e1e-04a089166601","username":"teste"} move: {"piece":"Nf3","turn":"b"}
player: {"id":"cc646ddf-828f-2442-ad8d-f37184f189c5","username":"scarelli"} move: {"piece":"Qc1","turn":"w"}
```

Fonte: Elaborada pelo autor.

A Figura 12 apresenta o ponto do segundo usuário, abrindo conexão com o servidor, entra na sala criada pelo primeiro usuário, recebe uma lista de jogadores presentes na sala, recebe a mensagem de que o jogo foi iniciado, cada jogador realiza um movimento.

Figura 12 – Partida teste observada pelo segundo cliente.

```
C:\Users\mausc\OneDrive\Documentos\projetos\tcc-backend\src\__tests__>node client2.js
Connection successful
teste entrou na sala
Successfuly joined the room
Players connected: [{"id":"cc646ddf-828f-2442-ad8d-f37184f189c5","username":"scarelli"},{"id":"38c5e172-67f7-b4
dd-2e1e-04a089166601","username":"teste"}]
Jogo iniciado
player: {"id":"38c5e172-67f7-b4dd-2e1e-04a089166601","username":"teste"} move: {"piece":"Nf3","turn":"b"}
player: {"id":"cc646ddf-828f-2442-ad8d-f37184f189c5","username":"scarelli"} move: {"piece":"Qc1","turn":"w"}
```

Fonte: Elaborada pelo autor.

5 Conclusão

A partir dos dados levantados, foi possível definir os moldes da aplicação e construir um protótipo funcional em Typescript, documentá-lo, e realizar testes usando xadrez como exemplo. A API foi capaz de simular uma partida entre dois jogadores, armazenar os dados das jogadas em uma tabela criada dinamicamente.

O evento de realizar uma jogada obteve aproximadamente 30 milisegundos de tempo de resposta durante os testes, sendo considerado aceitável para um jogo em rede. O protótipo provou ser possível criar uma ferramenta para apoiar outros desenvolvedores na criação de jogos multijogador, ou participar de um jogo executando em cima do ambiente, com uma documentação descritiva a respeito do código desenvolvido, visando facilitar o uso para pessoas que desconhecem seu interior.

5.1 Trabalhos futuros

Para trabalhos futuros propõe-se a adição de *cache* em algumas consultas realizadas ao banco de dados, trazendo maior agilidade a aplicação, e a busca por formas eficientes de se gerar métricas sobre os dados coletados das partidas, com a finalidade de facilitar o processo de avaliação do professor a respeito do aprendizado. Em adição, pode-se refatorar partes do código para reduzir ainda mais o tempo de resposta de uma jogada a partir de otimizações.

Propõe-se também a criação de um jogo com interface gráfica utilizando uma *game engine* para exemplificar o funcionamento da API, demonstrando de forma mais prática as possibilidades e limitações da aplicação.

Referências

BLOCH, J. 2018. Disponível em: <https://www.infoq.com/presentations/history-api/>. Acesso em 10 de Dezembro de 2021.

CAVALCANTE, P. H. A. *Introdução a Typescript: o que é e como começar?* 2021. Disponível em: <https://blog.geekhunter.com.br/introducao-a-typescript/>. Acesso em 12 de Setembro de 2021.

CHESS.COM. 2020. Disponível em: <https://www.chess.com/terms/fen-chess>. Acesso em 5 de Outubro de 2021.

COTTON, I.; GREATOREX, F. Data structures and techniques for remote computer graphics. *AFIPS: American Federation of Information Processing Societies*, Association for Computing Machinery, p. 533—544, 1952.

DUARTE, L. *O que é Node.js e outras 5 dúvidas fundamentais*. 2021. Disponível em: <https://www.luiztools.com.br/post/o-que-e-nodejs-e-outras-5-duvidas-fundamentais/>. Acesso em 4 de Setembro de 2021.

FERNANDES, A. *O que é API? Entenda de uma maneira simples*. 2018. Disponível em: <https://vertigo.com.br/o-que-e-api-entenda-de-uma-maneira-simples/>. Acesso em 10 de Agosto de 2021.

FIANI, R. *Teoria dos Jogos*. [S.l.]: Elsevier, 2009. 41–50 p.

GEE, J. P. *What video games have to teach us about learning and literacy*. New York: Palgrave Macmillan, 2003. 1–71 p. Disponível em: <https://blog.ufes.br/kyriafinardi/files/2017/10/What-Video-Games-Have-to-Teach-us-About-Learning-and-Literacy-2003.-ilovepdf-compressed.pdf>. Acesso em: 5 de maio de 2021.

GEE, J. P. Good video games and good learning. *Phi Kappa Phi Forum*, p. 33–37, 2005.

GOLDSTINE, H. H.; NEUMANN, J. von. *Planning and Coding of Problems for an Electronic Computing Instrument. Part II, Volume III*. Princeton: Institute for Advanced Study, 1948.

GUININ, D. *Saiba tudo sobre a linguagem JavaScript*. 2019. Disponível em: <https://www.segredosdatecnologia.com/tudo-sobre-o-javascript/>. Acesso em 2 de Setembro de 2021.

LANE, K. *Intro to APIs: History of APIs*. 2019. Disponível em: <https://blog.postman.com/intro-to-apis-history-of-apis/>. Acesso em 5 de Fevereiro de 2022.

MEDEIROS, H. *Application Programming Interface: Desenvolvendo APIs de Software*. 2014. Disponível em: <https://www.devmedia.com.br/application-programming-interface-desenvolvendo-apis-de-software/30548>. Acesso em 10 de Agosto de 2021.

MOZILLA. 2020. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/API/WebSocket>. Acesso em 08 de Março de 2022.

PAIXAO, J. R. da. *O que é SOLID: O guia completo para você entender os 5 princípios da POO*. 2019. Disponível em: <https://medium.com/desenvolvendo-com-paixao/o-que-é-solid-o-guia-completo-para-você-entender-os-5-princípios-da-poo-2b937b3fc530>. Acesso em 08 de Março de 2022.

ROGERS, S. *Level Up!: Um guia para o design de grandes jogos*. [S.l.]: Blucher, 2012.

SALEN, K.; ZIMMERMAN, E. *Rules of play: Game design fundamentals*. [S.l.]: MIT press, 2004.

SCHELL, J. *The art of game design*. [S.l.]: A K Peters/CRC Press, 2008. 23–46 p.

SOUZA, I. de. *Saiba o que é Node.js, como ele funciona e como usá-lo no seu site*. 2020. Disponível em: <https://rockcontent.com/br/blog/node-js/>. Acesso em 2 de Setembro de 2021.

SOUZA, S. L. P. de A. C.; INOCENTE, N. J.; ARAUJO, E. A. S. de. Autoeficácia no trabalho docente: o uso de tecnologia digital e virtual no processo de ensino e aprendizagem. *Revista Brasileira de Gestão e Desenvolvimento Regional*, v. 12, n. 2, p. 328–348, 2016.

SUITS, B. What is a game? *Philosophy of Science*, The University of Chicago Press, v. 34, n. 2, p. 148–156, 1967.

TRISTÃO, M. B. *O lúdico na prática docente*. 39 p. Licenciado em Pedagogia — Faculdade de Educação, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2010.

TYPESCRIPT. 2012. Disponível em: <https://www.typescriptlang.org>. Acesso em 15 de Setembro de 2021.

WHEELER, D. The use of sub-routines in programmes. *ACM national meeting*, Association for Computing Machinery, p. 235–236, 1952.

WILKES, M.; WHEELER, D.; GILL, S. *The Preparation of Programs for an Electronic Digital Computer*: With special reference to the edsac and the use of a library of subroutines. Madison: Addison-Wesley Press, 1951.