INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO CEARÁ
IFCE CAMPUS ARACATI
COORDENADORIA DE CIÊNCIA DA COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

EDGAR DOS SANTOS OLIVEIRA

eJET - Uma Plataforma Web para Jogos Educativos via Templates

ARACATI-CE
2021

EDGAR DOS SANTOS OLIVEIRA

eJET - Uma Plataforma Web para Jogos Educativos via Templates

Trabalho de Conclusão de Curso (TCC) apresentado ao curso de Bacharelado em Ciência da Computação do Instituto Federal de Educação, Ciência e Tecnologia do Ceará - IFCE - Campus Aracati, como requisito parcial para obtenção do Título de Bacharel em Ciência da Computação.

Orientador: Prof. Ricardo Lenz

Aracati-CE
2021

Dados Internacionais de Catalogação na Publicação
Instituto Federal do Ceará - IFCE
Sistema de Bibliotecas - SIBI
Ficha catalográfica elaborada pelo SIBI/IFCE, com os dados fornecidos pelo(a) autor(a)

O48e Oliveira, Edgar.
eJET : Uma Plataforma Web para Jogos Educativos via Templates / Edgar Oliveira. - 2021.
71 f. : il. color.

Trabalho de Conclusão de Curso (graduação) - Instituto Federal do Ceará, Bacharelado em Ciência da Computação, Campus Aracati, 2021.
Orientação: Prof. Me. Ricardo Lenz.

1. Framework de jogos educativos. 2. Templates. 3. Educação Remota. I. Titulo.

EDGAR DOS SANTOS OLIVEIRA

eJET: Uma Plataforma Web para Jogos Educativos via Templates

Trabalho de Conclusão de Curso (TCC) apresentado ao curso de Bacharelado em Ciência da Computação do Instituto Federal de Educação, Ciência e Tecnologia do Ceará - IFCE - Campus Aracati, como requisito parcial para obtenção do Título de Bacharel em Ciência da Computação.

Aprovada em <data>

BANCA EXAMINADORA

Prof. Me. Ricardo Lenz (Orientador)
Instituto Federal de Educação, Ciência e Tecnologia do Ceará

Profa. Me. Érica de Lima Gallindo
Instituto Federal de Educação, Ciência e Tecnologia do Ceará

Prof. Me. Diego Rocha Lima
Instituto Federal de Educação, Ciência e Tecnologia do Ceará

DEDICATÓRIA

Aos meus pais.
À minha família.
Aos professores.
O orientador desse trabalho.
Aos meus colegas e amigos.
A todos que me apoiaram nessa jornada.

AGRADECIMENTOS

À minha maravilhosa mãe, esta que sempre será meu exemplo de perseverança e determinação, que me criou, educou singularmente e tem me abençoado com uma índole digna de uma pessoa respeitável.

Ao meu pai, por quem possuo profundo respeito, que tem me provido de todas as coisas físicas e não físicas, me guiando, ajudando e mostrando um bom exemplo de conduta diante das adversidades que esse mundo nos impõe.

Ao meu irmão, que me inspira com sua determinação de buscar aquilo que quer, me faz rir com sua personalidade irônica e me ensina sobre mim mesmo, mesmo sem querer ou perceber.

A toda a minha família, que não são apenas responsáveis pela minha personalidade como também minha existência, a eles devo tudo.

À minha namorada, que me apoiou durante os momentos mais difíceis e me proporcionou o suporte psicológico durante toda essa longa jornada.

Ao meu orientador, Ricardo, que me ensinou muito sobre diversos assuntos (até mesmo sobre filosofia), me guiou tal qual Virgílio guiou Dante pelo inferno e me permitiu recuperar a fé quando estava prestes a desistir.

À professora Érica, que me ensinou que muitas vezes o valor do nosso trabalho está em resolver problemas; exatamente como isso é feito, em certos aspectos, talvez nem seja tão importante; e também ensinou a não subestimar nosso próprio esforço.

À professora Carina e o professor Reinaldo, me mostraram que mais importante do que como se chega a algum lugar é continuar se movendo em sua direção. A determinação é essencial para atingir qualquer objetivo e é mais valiosa que o talento.

Ao professor Diego, que inspira inúmeros alunos como eu a fazermos nosso melhor, sempre exigente e detalhista, nunca apelando para uma pressão gratuita, que alguns professos costumam usar para "incentivar os alunos".

Ao professor Joel, sempre humilde, nos fazia sentir que estávamos diante de outro

aluno; constantemente apresentava novos níveis de complexidade na programação, nos fazendo sentir ainda bastante inexperientes, a cada nova aula, o que me enchia de determinação e ânimo.

A todos os meus colegas alunos, que renovaram as minhas esperanças, todos passando por dificuldades muito piores que as minhas e nunca demostraram outra reação que não fosse gargalhadas, como bons nordestinos que são, e graças a eles eu aprendi a fazer o mesmo.

Ao meu cachorro, por não ter comido meu trabalho.

RESUMO

Dentro do contexto atual complexo com uma educação remota, e considerando o fenômeno da *gamificação*, o presente trabalho propõe um sistema para jogos educativos acessíveis via *Web* que permita uma rápida produção de conteúdo por parte de professores através de uma alimentação de dados para *templates* de jogos pré-prontos mediante a utilização de um *framework* especializado para jogos digitais 2D voltados para a educação. O sistema implementado demonstra o *framework* proposto com um *template* de jogo de tabuleiro, permitindo o cadastro de usuários – alunos e professores – e a consequente produção e consumo da atividade jogada pelos alunos. Ao final, um relatório é gerado para o professor com os resultados individuais de cada aluno que participou da atividade do jogo. O trabalho faz comparativos com diversos outros *frameworks* e contribui com uma implementação de Software Livre tanto para o *framework* proposto como para o sistema Web já em funcionamento.

Palavras-chaves: Framework de jogos educativos. Templates. Educação Remota.

ABSTRACT

Within the current complex context with remote education, and considering gamification trends, the present work proposes a system for educational Web-based games that allows a fast production of content by teachers through input data for existing game templates by using a specialized framework for 2D digital educational games. The implemented system shows the proposed framework with a board game template, allowing the registration of users – students and teachers – and the consequent production and consumption of the activity played by the students. At the end, a report is generated for the teacher with the individual results of each student who participated in the game activity. The work makes comparisons with several other frameworks and contributes with a Free Software implementation both for the proposed framework and for the Web system already in operation.

Keywords: Educational game framework. Templates. Remote Education

LISTA DE ILUSTRAÇÕES

Figura 1 – Pesquisas sobre Gamificação no *Google* entre 2014 e 2020 16

Figura 2 – Alguns Conceitos Básicos em Jogos Digitais 24
Figura 3 – Tipo de Imagem de Bitmap e Vetorial 26
Figura 4 – Tecnologias Gráficas Fundamentais na Web 30

Figura 5 – Trabalhos Relacionados 36

Figura 6 – Visão Geral da Plataforma eJET 44
Figura 7 – Descrição textual alimentando um Template de Jogo 57
Figura 8 – Interação dos alunos com um jogo, com resultados coletados para o professor 58
Figura 9 – Professores alimentam *templates* com descrições de conteúdo, gerando jogos disponibilizados para alunos por meio de link único . . . 59

Figura 10 –Tela de Login 61
Figura 11 –Tela de Registro 62
Figura 12 –Tela de Atividades 62
Figura 13 –Caixa de Diálogo para Criar Nova Atividade 63
Figura 14 –Exemplo de Arquivo JSON 63
Figura 15 –Exemplo de Atividade Criada 64
Figura 16 –Melhores Pontuações 65
Figura 17 –Tela de Resultados 65

LISTA DE TABELAS

Tabela 1 – Itens Considerados no Escopo 21

Tabela 2 – *Frameworks* relacionados e Estrelas no Github 43

Tabela 3 – Comparativo entre os *Frameworks* 66

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
CSS	Cascading Style Sheets
CSV	Comma-Separated Values
ECS	Entity Component System
FPS	Frames Per Second
GPU	Graphics Processing Unit
HTML	HyperText Markup Language
IFCE	Instituto Federal de Educação, Ciência e Tecnologia do Ceará
IOS	iPhone Operating System
IDH	Índice de Desenvolvimento Humano
JSON	JavaScript Object Notation
OCDE	Organização para a Cooperação e Desenvolvimento Econômico
PISA	Programme for International Student Assessment
SGML	Standard Generalized Markup Language
SVG	Scalable Vector Graphic
TCC	Trabalho de Conclusão de Curso
W3C	World Wide Web Consortium
XHR	XMLHttpRequest
XML	Extensible Markup Language
YAML	YAML Ain’t Markup Language

SUMÁRIO

1 Introdução 14
1.1 Objetivos 20
1.1.1 Objetivo Geral 20
1.1.2 Objetivos Específicos 20
1.2 Metodologia 21
1.3 Organização do Trabalho 23

2 Fundamentação Teórica 24
2.1 Conceitos Básicos de Jogos Digitais 24
2.2 Conceitos Básicos de Computação Gráfica 25
2.2.1 Gráficos de *Bitmap* e Vetorial 26
2.2.2 Transformações Geométricas 27
2.2.3 Animações e Transições 28
2.3 Tecnologias *Web* 29
2.3.1 Tecnologias Fundamentais: HTML, CSS, *JavaScript* 29
2.3.2 Tecnologias Gráficas: SVG, Canvas, WebGL 30
2.3.3 Descrição de Dados: XML, JSON, YAML 32
2.4 Padrões de Projeto e Código 32

3 Trabalhos Relacionados 34
3.1 Visão Geral 34
3.2 Frameworks 35
3.2.1 Boardgame.io 35
3.2.2 *Construct* 35
3.2.3 CreateJS 37
3.2.4 Crafty JS 37
3.2.5 EnchantJS 38
3.2.6 Frozen 38
3.2.7 GDevelop 38
3.2.8 iLearnTest 39
3.2.9 *ImpactJS* 39
3.2.10 *Kiwi.js* 39
3.2.11 *LimeJS* 39
3.2.12 *MelonJS* 40
3.2.13 *Panda Engine* 40
3.2.14 *Phaser* 40

3.2.15 *PixiJS* 41
3.2.16 Turbulenz 41
3.3 Análise 41

4 Proposta 44
4.1 Framework eJET 45
4.1.1 Módulo *WebGL* 45
4.1.2 Módulo ECS 50
4.1.3 Módulo Input 54
4.1.4 Módulo HTML 55
4.2 Sistema *Web* eJET 57

5 Resultados 61

6 Considerações Finais 67

REFERÊNCIAS 69

1 Introdução

"A educação é a arma mais poderosa que você pode usar para mudar o mundo"

— Nelson Mandela

"O que aprendemos com prazer nunca esquecemos."

— Alfred Mercier

A educação é um fator essencial para o crescimento da sociedade como um todo, fazendo transbordar mais criatividade, cidadania, inovação, produtividade, cultura e pensamento crítico. É um componente fundamental para o desenvolvimento humano, sendo, inclusive, elemento importante na fórmula do IDH (Índice de Desenvolvimento Humano) de um país. Conforme o *Global Partnership for Education*[1], em um *ranking* educacional de 65 países o Brasil está em 54º lugar[2]. No PISA[3], o Brasil ocupa o 57º lugar[4]. O país possui 12 milhões de analfabetos e mais que 50% dos adultos entre 25 e 64 anos não concluíram o Ensino Médio no ano de 2019[5]. De fato, dois terços dos brasileiros com 15 anos têm conhecimentos abaixo do básico em matemática, havendo ainda uma estagnação em leitura ao longo da década[6]. Diante de um quadro difícil da educação no país, e considerando o quão valiosa ela é, torna-se motivador lançar-se nos esforços em prol de instrumentos que possam de alguma forma agregar valor às atividades educacionais.

Muitas vezes, melhorar a educação no Brasil envolve um trabalho em múltiplas frentes; além da necessidade de um maior investimento nesse âmbito, não só no fortalecimento estrutural como também na capacitação contínua e valorização da carreira de profissionais diversos da área de educação, também é valioso ajudar a compor meios mais dinâmicos de aprendizado, usando a tecnologia para somar mais ao ensino através de recursos digitais que possam despertar o interesse e promover uma proveitosa interação por parte dos alunos. A educação, explorando mais espaços

[1] https://worldpopulationreview.com/country-rankings/education-rankings-by-country

[2] O Brasil conta com 1203 pontos; o primeiro lugar tem 1731 pontos; Peru, a penúltima posição, tem 1104 pontos.

[3] PISA: Programa Internacional de Avaliação de Alunos (em inglês: *Programme for International Student Assessment*). O programa, coordenado pela OCDE (Organização para a Cooperação e Desenvolvimento Econômico), que vem sendo realizado desde 2000, avalia o desempenho em vários países quanto à leitura, matemática e ciências, entre outros.

[4] 57º numa lista da ordem de 70 países; para maiores informações, consultar as reportagens em: https://www.universia.net/br/actualidad/orientacao-academica/o-brasil-ranking-mundial-educaco-1166599.html e https://www1.folha.uol.com.br/educacao/2019/12/brasil-e-57o-do-mundo-em-ranking-de-educacao-veja-evolucao-no-pisa-desde-2000.shtml

[5] https://g1.globo.com/educacao/noticia/2019/06/19/mais-da-metade-dos-brasileiros-de-25-anos-ou-mais-ainda-nao-concluiu-a-educacao-basica-aponta-ibge.ghtml

[6] https://g1.globo.com/educacao/noticia/2019/12/03/brasil-cai-em-ranking-mundial-de-educacao-em-matematica-e-ciencias-e-fica-estagnado-em-leitura.ghtml

e tecnologias potenciais, chegando a mais pessoas e comunidades e com maior receptividade, multiplica seus benefícios, permite mais ascensão social, melhora produtividade e gestão, e enriquecer intelectualmente, tornando-se chave para um amanhã mais duradouro e sustentável.

No Brasil, mesmo certos grupos menos favorecidos e com várias deficiências em educação têm acesso a celular e Internet[7], ainda que com uma banda mais limitada[8]. Isso significa que algum tipo de conteúdo educativo poderia chegar em muitos através de meios digitais, em particular por meio de tecnologias *Web*. Assim, salta à vista a ideia de buscar caminhos para o ensino usando algo que possa ser produzido num computador e consumido via Internet por vários usuários, especialmente em algum molde moderno, atrativo e dinâmico.

É de comum saber que muitos estudantes têm dificuldades no processo educacional com certos métodos mais tradicionais de ensino. Mais recentemente, a tendência do uso de jogos como ferramentas para o aprendizado tem proporcionado diversas vezes bons resultados (GIRARD; ECALLE; MAGNAN, 2013). Trata-se da chamada *gamificação*, o uso de elementos de jogos em contextos não relacionados a jogos (DETERDING et al., 2011). A gamificação pode ser aplicada a várias áreas em geral; aqui, o foco se dá na gamificação na educação, usando mecânicas e dinâmicas de jogos para engajar pessoas no aprendizado. Através de exercícios dispostos num formato de jogo simples e acessível para o público alvo, a experiência de educação é enriquecida. Os elementos diversos da estrutura básica de um jogo digital (ZICHERMANN; CUNNINGHAM, 2011), como o elemento da pontuação, do desafio do tempo, de níveis diversos, etc. podem ser apresentados num pacote gráfico interativo com o propósito de motivar o jogador, culminando num aprendizado que tem despertado a atenção e interesse de muitos. Os potenciais de aplicação disso se dão não somente para um determinado nível de ensino, mas fluem para diversos segmentos, envolvendo não só a educação básica, mas também o ensino superior.

O gráfico da Figura 1 mostra o nível de interesse em gamificação ao longo da última década por meio de buscas realizadas no *Google Trends*[9], inspirado no trabalho de Basten (2017). O gráfico apresenta a porcentagem de buscas, em cada momento do tempo, diante do total de buscas realizadas. Nota-se um aumento significativo especialmente entre os anos de 2010 e 2014, assim sendo destacados na Figura 1.

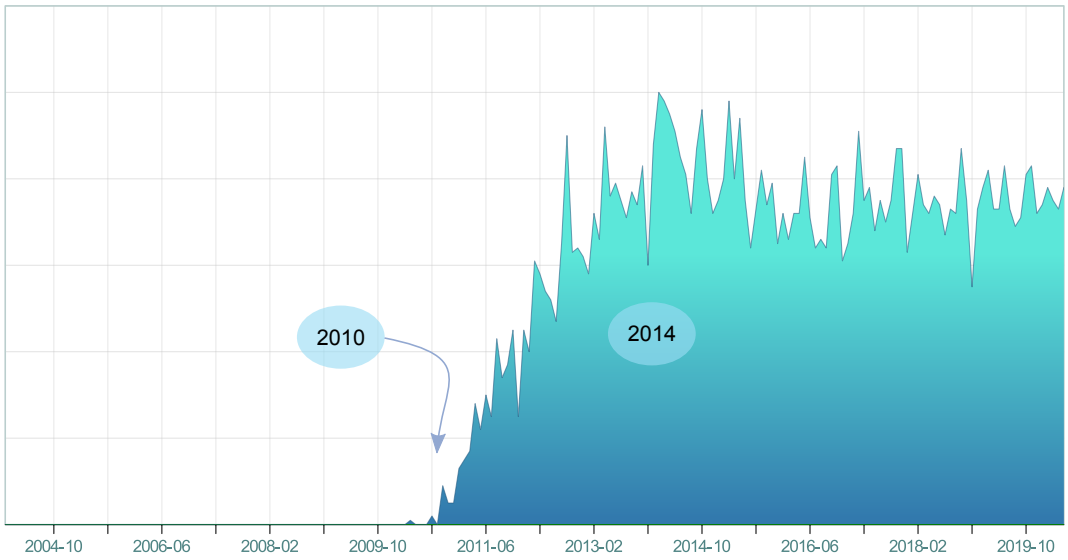
De fato, Su (2016) sugere inclusive que várias instituições de ensino deveriam

[7] https://www.zdnet.com/article/internet-use-increases-among-poor-brazilians/

[8] Uma banda mais limitada pode implicar num contexto de maior economia nos dados, preferindo-se um uso moderado de imagens menores a um uso intenso de imagens, sons ou até mesmo vídeos. Para vários jogos tradicionais, isso pode ter um impacto maior; muitos jogos educativos, porém, já são mais econômicos por natureza em relação a isso.

[9] Google Trends é uma ferramenta do Google que mostra os mais populares termos buscados em um passado recente.

Figura 1 – Pesquisas sobre Gamificação no *Google* entre 2014 e 2020



Fonte: Dos autores

encorajar seus professores para pesquisar e desenvolver materiais inovadores de ensino. Ademais, o uso de tecnologias digitais formatadas de um modo interessante para o aluno podem ainda ser valiosas ferramentas para pessoas com deficiências ou transtornos (NETO; BLANCO; SILVA, 2017).

No contexto atual da pandemia do Covid-19, com grande ampliação de modalidades de ensino remoto, o interesse por soluções educacionais ligadas à gamificação torna-se ainda mais relevante na busca por manter a atenção e engajamento dos alunos e evitar maiores índices de evasão[10] (PERNELLE et al., 2021). De fato, além de gerar um estímulo para o estudante explorar mais conteúdos, um aprendizado estruturado de forma mais lúdica com jogos educativos pode ajudar inclusive a reduzir a *ansiedade* (SU, 2016), fenômeno este que tem sido tão discutido nesse período de crise.

Ainda na pandemia já se tem relatos de uma recepção e percepção positivas dos alunos sobre o uso de gamificação na educação como sendo uma estratégia inovadora, eficiente e com atrativos maiores para engajamento (NIETO-ESCAMEZ; ROLDÁN-TAPIA, 2021). Isso é especialmente interessante diante de condições físicas e psicológicas ruins de muitos alunos diante do confinamento e pandemia em geral, pois muitos não tem um ambiente físico adequado para o estudo. Assim, a ideia de gamificação na educação pode ser implementada como complemento às aulas e pode

[10] Pode-se lembrar ainda que o Brasil, entre os 100 países com maior IDH, é um dos que têm as maiores taxas de abandono escolar.

ser um instrumento valioso inclusive em tempos pós-Covid, ainda segundo o trabalho citado.

Em todo caso, segundo os autores do livro *Animal Play*, o próprio ato de jogar ou brincar em si é atividade natural e prazerosa durante a vida de inúmeros animais, incluindo o ser humano, sobretudo na infância. Essa atividade atrai muito naturalmente os seres humanos e é fruto de um fecundo processo evolutivo. Os animais que jogam ou simulam atividades antes de realizá-las acabam em geral tendo um desempenho melhor. Aprender brincando constitui-se, assim, num método bastante natural de adquirir e reforçar o conhecimento (BEKOFF; DIMOTTA, 2008).

O trabalho de Liu e Chen (2013), por exemplo, apresenta um estudo mostrando resultados superiores no aprendizado de crianças com o uso de jogos educativos. Por outro lado, jogos educativos podem ser projetados para atender tanto um segmento mais jovem como um mais adulto também; são também chamados às vezes de *serious games* (DJAOUTI; ALVAREZ; JESSEL, 2011).

Entretanto, o uso de jogos para a educação pode, às vezes, ter certas dificuldades de adoção. Alguns professores, por exemplo, podem interpretar esses jogos de natureza educativa como um certo concorrente às suas aulas. Mas não precisa ser assim; ao invés disso, tais jogos se encaixam melhor numa categoria de conteúdo complementar, até mesmo um aliado na abordagem de diversas matérias, podendo ser efetivos no aprendizado de adultos e de crianças conforme o ritmo de cada um (GIRARD; ECALLE; MAGNAN, 2013).

Por outro lado, mesmo que professores considerem vantajoso e até tenham interesse em usar jogos educativos, vários profissionais da educação deparam-se com o seguinte problema: muitas vezes, precisam encontrar jogos com conteúdos já prontos para uso, pois não têm os meios para criar, eles mesmos, seus próprios conteúdos. A liberdade do docente poder criar seus próprios conteúdos é primordial, inclusive para atender demandas regionais por certos saberes que não necessariamente estejam contemplados em determinadas estruturas curriculares gerais padronizadas. Com a possibilidade de criar seus próprios conteúdos, o docente pode lidar melhor com os contextos e demandas dos seus estudantes e elaborar algo mais apropriado. Assim, ‘poder criar’, no contexto de jogos educativos, é imprescindível.

Existem, é verdade, diversas soluções para a criação de jogos, porém muitas vezes são focadas em jogos gerais, ao invés de jogos educativos, ou ainda exigem profundos conhecimentos de programação, o que as inviabiliza para muitos. Além disso, muitas vezes não são projetados de forma expansível ou adaptável, isto é, não têm como base um *framework* ou algo do tipo, o que possibilitaria um maior reuso e adaptação para cenários diversos; outros, ainda, nem são *abertos*.

À respeito de ser 'aberto', convém lembrar que, muitas vezes, uma ferramenta computacional, por mais completa que seja para diversos usuários, dificilmente atende as necessidades de todos, ou não atende da melhor forma. Caso um usuário tenha necessidade (ou mesmo vontade) de agregar novas funcionalidades, ou até contribuir para o programa, e distribuir para outros que necessitam das mesmas melhorias, esse tipo de ação pode não ser possível em função da *licença do software*. Para garantir que os usuários possam modificar, melhorar, executar sem restrições arbitrárias em diversas plataformas, compartilhar e redistribuir o programa para a comunidade pode-se optar pela adoção de uma licença de *Software Livre*, assegurando todas essas liberdades fundamentais (STALLMAN, 2002). O *Software Livre*[11], inclusive por ser aberto, pode ser inspecionado quanto ao seu conteúdo, o que transmite mais transparência e confiabilidade (o *software* que não é livre, isto é, o *software* proprietário, deixa os usuários sem algumas (ou todas) essas liberdades; os usuários, muitas vezes, não podem sequer ter certeza do que o programa realmente está fazendo). Além disso, frequentemente (embora não necessariamente) o Software Livre é disponibilizado gratuitamente. Para muitos países e diversos contextos sócio-econômicos, isso torna-se uma característica extremamente desejável, particularmente quando se visa a educação pública universal gratuita, sendo, portanto, também algo que será levado em consideração neste trabalho.

Outra consideração importante a se fazer sobre a natureza dos jogos educativos sendo visada aqui é da adoção da plataforma *Web*, que pode ser acessado de todas as plataformas, ao invés de serem jogos implementados para uma plataforma específica, como uma aplicação para *tablets* de alguma marca ou restrições de sistemas operacionais (como Windows, iOS ou Android). Mesmo que, tradicionalmente, a plataforma *Web* se apresente em geral com uma capacidade gráfica mais limitada diante de implementações nativas para certos sistemas específicos, tem a vantagem quanto ao quesito de distribuição[12]. Potencialmente qualquer usuário pode ter acesso usando o dispositivo/sistema que tiver, sendo um fator importante na busca de difundir mais o acesso à educação. Assim, para o propósito do presente trabalho de jogos voltados para educação, essa característica de adotar a plataforma *Web* é importante. Além disso, pode-se mais facilmente trabalhar com as atualizações do programa, sem precisar demandar esforço algum por parte da rede de usuários, como a necessidade de eles terem que instalar módulos ou dependências de bibliotecas (CHARLAND; LEROUX, 2011). Por outro lado, há de se comentar, para fins de esclarecimento, que mesmo a questão da limitação de desempenho na *Web* já tem sido mitigada em anos recentes, especialmente com a *WebGL* com suporte à GPU (ANGEL; SHREINER, 2014) e ainda a possibilidade de *WebAssembly* (HAAS et al., 2017), que permite um

[11] https://www.gnu.org/philosophy/free-sw.en.html
[12] https://developer.mozilla.org/en-US/docs/Games/Introduction

desempenho superior. Com isso, a plataforma *Web*, que começou como uma simples rede de troca de documentos, firma-se ainda mais como uma ubíqua plataforma de aplicações madura e aberta a demandas sofisticadas de implementações, mesmo que tenham conteúdos gráficos mais elaborados de vídeo e jogos, assegurando assim um caminho mais do que suficiente para quem deseja desenvolver soluções de jogos educativos para *Web*.

Existem diversos jogos educativos; comum a todos eles são os elementos da dinâmica de jogos, como imagens em movimento e pontuação, entre outros já destacados anteriormente. Particularmente, um modelo bastante popular para jogos educativos é o baseado em algum tipo de tabuleiro, com peças, cartas ou, num sentido genérico, imagens (com capacidade de animação e movimento) que ficam dispostas sobre o mesmo e apresentam o atual estado do jogo. Muitos jogos educativos podem ser trabalhados dentro desse modelo. Uma modalidade que se encaixa aqui é a de um jogo que trabalha com cartas, umas contendo conceitos e outras contendo as respectivas explicações; o jogador deve, interagindo com as cartas, fazer as correspondências certas e assim vencer o jogo. A dinâmica completa dessa modalidade pode ser modelada num *template*, isto é, uma implementação pré-pronta de jogo, podendo incluir inclusive figuras, sons e outros ativos e mídias já prontos para uso. Dessa forma, basta que sejam fornecidos os conceitos e as respectivas explicações e um jogo completo poderá ser disponibilizado num ambiente adequado.

Dessa maneira, uma solução para jogos educativos poderia implantar um esquema que trabalhasse com *templates* de jogos, visando assim a automação desses processos. Por outro lado, não bastaria que um sistema assim apenas disponibilizasse certo *template* para jogos; por mirar especificamente em jogos educativos, seria mais interessante ainda que o sistema disponibilizasse todo o ambiente adequado para isso, isto é, um ambiente que permitisse que alunos pudessem se conectar, jogar e terem suas partidas registradas para o devido acompanhamento por parte de um professor. Isso significa que o sistema teria que suportar ainda o cadastro de alunos e as estruturas adequadas para isso.

Além disso, se o propósito é que os professores possam mais facilmente e rapidamente criar conteúdos próprios de jogos educativos, o sistema não só deve suportar a ideia em si de *template* de jogo, como também poderia permitir que professores pudessem criar os conteúdos dos jogos pela própria *Web*, isto é, cadastrando no mesmo ambiente dos alunos os seus conteúdos de jogos criados. Assim, casa-se o ato de criar conteúdo do jogo educativo com o ato de consumir o conteúdo do jogo educativo num mesmo ambiente unificado e que funciona inteiramente via *Web*. Ademais, tal ambiente deve, para fins de acompanhamento por parte do professor, apresentar como cada aluno se saiu no jogo. Assim, diante do exposto, é possível

agora definir com maior precisão quais os objetivos propostos pelo trabalho.

1.1 Objetivos

1.1.1 Objetivo Geral

O trabalho objetiva fornecer um sistema para jogos educativos acessíveis via *Web* e com conteúdos que podem ser criados rapidamente por meio da alimentação de dados encaminhada para um *template* de jogos disponibilizado; o sistema deve ter como base um *framework* aberto e expansível, também proposto aqui para o propósito de desenvolvimento de jogos educativos, e deve suportar um cadastro fácil e rápido de alunos e professores, tendo ou não conhecimento em programação, além de permitir que alunos possam jogar um jogo modelado pelo professor e que o professor possa acompanhar os resultados. Para propósitos de validação da proposta, o sistema deverá vir com um *template* pronto para uso, o da modalidade de um tabuleiro com cartas. Deverá, ainda, ser implementado de forma aberta, sendo oferecido na forma de *Software Livre*, gratuito e sem depender de nenhum equipamento ou sistema operacional em particular.

1.1.2 Objetivos Específicos

1. Criar um *framework* para desenvolvimentos de jogos gráficos com tecnologias Web, apropriados para a linha de jogos educativos;
2. Criar um sistema Web para uso de alunos e professores, com telas para login, cadastro de usuários, criação de um jogo por parte do docente segundo um *template*, execução do jogo por parte dos alunos e visualização dos resultados dos alunos para acompanhamento pelo professor;
3. Fazer um levantamento das principais tecnologias e *frameworks* para trabalho gráfico na Web no contexto de jogos digitais 2D;
4. Apresentar os conceitos básicos subjacentes ao trabalho, incluindo certos conceitos de jogos digitais 2D usados no *framework*;
5. Apresentar uma documentação básica do *framework* e do sistema Web propostos, mostrando, no caso desse último, como alunos e professores podem usá-lo;
6. Implementar um *template* de jogo pronto para uso, demonstrando assim o *framework* no sistema Web e validando a proposta.

1.2 Metodologia

Considerando o tema proposto de *softwares* educativos na *Web* e a tendência de gamificação, foi conduzida uma revisão bibliográfica para apropriação dos conceitos básicos relacionados e exploração de soluções e *frameworks* existentes passíveis de serem utilizados para isso. Com essa revisão bibliográfica, foi-se delineando um objetivo mais claro para o trabalho, seu nicho específico de atuação e seu escopo.

Projetos com tecnologia *Web* frequentemente trabalham em cima de *frameworks*. Assim sendo, foram investigados diversos *frameworks* relacionados com o tema. Para fins de delimitação do escopo, uma série de características, algumas obrigatórias e outras opcionais, foram demarcadas. Essas características aparecem listadas brevemente na Tabela 1 e são descritas com maiores detalhes abaixo:

Tabela 1 – Itens Considerados no Escopo

Item	Descrição
R1	Gráficos 2D
R2	Plataforma Web
R3	Software Livre
R4	Independência de SO específico
CD1	Sistema / ferramenta vinculada
CD2	Modalidade clássica de tabuleiro disponível
CD3	Independência de uma teia de bibliotecas

- Por se optar por jogos educativos projetados especificamente na *Web*, optou-se por eliminar soluções existentes que não fossem implementadas com tecnologia *Web*. Isso gerou o primeiro requisito para delimitação do escopo (R1).

- Além disso, jogos educativos na *Web* são frequentemente projetados num paradigma de gráficos 2D. Em geral, jogos com gráficos 3D exigem um investimento maior e, para fins educacionais, muitas vezes uma abordagem em 2D já é bastante adequada. Assim, optou-se por focar em soluções *Web* que trabalhassem especificamente com gráficos 2D, gerando assim um segundo requisito (R2).

- Tendo-se optado pelo trabalho com algo livre e aberto, foram examinadas soluções disponíveis dentro desse escopo, deixando de fora soluções não livres / proprietárias, o que define o terceiro requisito (R3).

- A ideia de trabalhar com tecnologia *Web* não podia ser apenas um certo substrato do projeto, de modo que o mesmo dependesse de elementos exteriores e que fossem dependentes de algum sistema operacional ou dispositivo computacional específico. Ao contrário, deve-se frisar que a solução deve explicitamente ser independente de sistemas específicos, sendo assim verdadeiramente multiplataforma. Isso gerou o requisito (R4).

Os quatro requisitos supracitados (R1, R2, R3 e R4) foram considerados essenciais, delimitando o escopo dos trabalhos que seriam analisados. Porém, algumas outras características desejáveis foram estipuladas; tais características seriam atributos muito interessantes de se encontrar num trabalho, mas não seriam estritamente obrigatórias. São elas:

- É possível que um determinado framework possa ter um *software*, uma ferramenta ou sistema à disposição, de alguma forma vinculado por sua própria natureza ao framework, que permita atuar no fluxo de trabalho deste último. Por exemplo, o framework pode ter um sistema que implementa e/ou complementa suas funcionalidades de produção de conteúdo, automação na geração de conteúdos diversos relacionados, etc. Nem todo framework terá isso, mas não deixa de ser uma característica interessante, gerando assim a primeira Característica Desejável (CD1).

- Uma vez considerando que a modalidade clássica de jogo de tabuleiro pode servir de fundamento para muitos jogos educativos, torna-se também atrativo o framework que já possuir, de alguma forma, algo semelhante a isso já previamente implementado, relativamente pronta para uso, sem prejuízo de outras modalidades. Assim, a característica de disponibilizar essa modalidade clássica de jogo firmou-se como a segunda Característica Desejável (CD2).

- Por fim, se o objetivo do trabalho envolve uma abertura para a contribuição de outros desenvolvedores da comunidade, pode ser mais acessível e convidativo

para outros se o projeto for mais descomplicado e livre de dependências diversas de bibliotecas de terceiros. Isso não é estritamente obrigatório, mas em todo caso, constitui-se num atrativo também (CD3).

Assim, todos os trabalhos analisados terão as características obrigatórias R1, R2, R3 e R4 e poderão ou não ter as características opcionais CD1, CD2 e CD3. Por outro lado, a solução proposta pelo presente trabalho deve também alinhar-se com os requisitos obrigatórios e, na medida do possível, suprir também com as características desejáveis.

1.3 Organização do Trabalho

O restante desse trabalho é organizado da seguinte forma: no Capítulo 2, os conceitos teóricos e tecnologias de base utilizadas são apresentados; no Capítulo 3, uma investigação sobre os principais *frameworks* da área é apresentada; no Capítulo 4, a proposta de *framework* é descrita com mais detalhes, junto do Sistema *Web* também proposto; no Capítulo 5, são apresentados os resultados do trabalho, demonstrando a ideia proposta, as telas do Sistema *Web* e os resultados da comparação com os trabalhos relacionados. Por fim, o Capítulo 6 apresenta as conclusões e opções de trabalhos futuros.

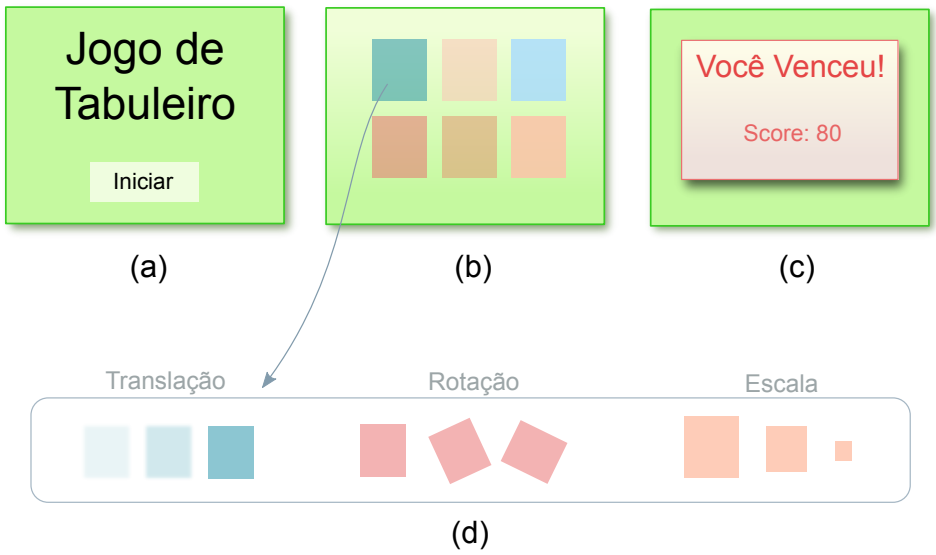
2 Fundamentação Teórica

Nesse capítulo veremos uma breve explicação de conceitos preliminares e tecnologias utilizadas. Isso será necessário para uma melhor compreensão dos Trabalhos Relacionados, seção seguinte.

2.1 Conceitos Básicos de Jogos Digitais

De uma forma simples, pode-se organizar a apresentação visual de um jogo digital 2D em uma hierarquia de telas (*screens*). Assim, ao entrar num jogo, pode-se apresentar em determinado momento uma tela inicial com um menu de opções (*menu screen*). O usuário poderá fazer algumas escolhas e configurações nesse momento inicial do jogo e depois escolher uma opção para iniciar o jogo, quando então se parte para a tela principal (*main screen*). Visualmente, pode-se realizar uma transição de uma tela para a outra (*screen transition*); existem diferentes animações gráficas para realizar essa categoria de transição, que normalmente envolve um movimento de saída da tela atual e o movimento de entrada da tela seguinte. Os itens (a) e (b) na Figura 2 demonstram essas diferentes telas.

Figura 2 – Alguns Conceitos Básicos em Jogos Digitais



Fonte: Dos autores

Uma vez na tela principal do jogo (no caso do jogo de tabuleiro, conforme a Figura 2), geralmente há uma composição visual para o ambiente de fundo formado

por unidades gráficas menores chamadas de *tiles*. Os jogadores e demais personagens animados que possam haver no jogo são representados por objetos gráficos que se movem no plano de frente do cenário, sendo chamados de *sprites*. Frequentemente *tiles* fazem parte de um cenário estático e *sprites* são os elementos dinâmicos, animados, que se movem pela tela. Seus movimentos podem ser controlados pelo jogador ou podem seguir trajetórias determinadas por algoritmos de busca ou caminhos pré-definidos.

Às vezes, o tabuleiro é grande demais para caber inteiramente de uma vez na tela. Nesse caso, pode-se focar somente uma certa área parcial do tabuleiro na tela, havendo nesse caso a necessidade de ter algum tipo de mecanismo de *scroll*. Às vezes, a natureza de um jogo é tal que é preferível que todo o tabuleiro esteja visível de uma vez na tela. Nesse caso, pode-se diminuir o tamanho dos elementos gráficos envolvidos ou simplesmente diminuir o número de elementos presentes no tabuleiro.

Durante o jogo, a *interface* gráfica contará principalmente com movimentos dos *sprites* sobre o tabuleiro e eventuais caixas de diálogo (*dialogs*), conforme apresentado no item (c) da Figura 2. No caso de jogos educativos de tabuleiro, tais caixas de diálogo podem trazer uma pergunta, possibilidades de resposta do jogador, etc. Em muitos jogos, há o uso de alguma categoria de fator aleatório para guiar o movimento do jogador, tipicamente realizado com a jogada de um dado. Isso também pode ser apresentado na forma de uma caixa de diálogo.

Por fim, quando o jogo termina, uma tela final de resultados pode ser apresentada (*score screen*). Nessa tela, tipicamente são apresentados a pontuação do(s) jogador(es) e algumas estatísticas sobre a partida, como tempo ou número de acertos, erros, movimentos de um tipo ou outro, ou qualquer outra possibilidade de informação conforme admita o tipo do jogo. Um elemento adicional ainda em jogos é de um *Ranking*; nesse caso, uma *ranking screen* traria os nomes dos jogadores com melhores pontuações, sendo isso um elemento estimulante para outros.

Ao longo da partida, os objetos gráficos poderão ser animados de diferentes formas. Poderão variar de imagens ou mudar de posição, rotação ou tamanho, isto é, poderão passar pelas chamadas Transformações Geométricas, que serão descritas à seguir junto dos conceitos básicos de Computação Gráfica utilizados.

2.2 Conceitos Básicos de Computação Gráfica

Parte fundamental dos jogos digitais sendo considerados envolve a aplicação de conceitos da área de Computação Gráfica. Telas e cenários são projetadas com desenhos, e vários elementos gráficos animados são modificados por meio de Trans-

formações Geométricas. Assim, entende-se ser útil cobrir brevemente alguns desses conceitos fundamentais da área.

2.2.1 Gráficos de Bitmap e Vetorial

Jogos digitais irão empregar imagens no formato de *Bitmap* ou Vetorial. O *Bitmap* é um formato padrão de imagem que contém uma matriz de *pixels*, fornecendo a cor de cada *pixel*. É também chamado de imagem matricial por muitos autores. Não contém informações geométricas sobre os objetos que estariam compondo a imagem (o que seria encontrado em uma imagem vetorial), mas apenas a cor em cada *pixel*. Dessa forma, o formato de *Bitmap* não garante que ao esticar a imagem as formas geométricas dos objetos desenhos serão preservados. É usado para fotos e outras categorias de desenho.

Figura 3 – Tipo de Imagem de Bitmap e Vetorial

Fonte: Dos autores

O formato vetorial, por outro lado, trabalha com a especificação precisa dos elementos de natureza mais geométrica que o compõem. Tipicamente, na hora de ser apresentado na tela passa por um processo de conversão para pixels chamado de *rasterização*. Por sua própria natureza de especificação precisa, admite ser rasterizado para imagens de dimensões arbitrárias, suportando assim ser “esticado” sem perda de qualidade. É um formato especialmente valorizado para a geração de documentos visando uma visualização gráfica ou impressão de qualidade superior.

Apesar disso, em anos recentes surgiu a chamada *Pixel Art*, uma categoria de arte digital que remete, num estilo retrô, a sistemas gráficos mais antigos, empregando uma composição baseada em pixels e, por vezes, até exaltando esse recurso para

fins estilísticos. Podem ser comparadas a mosaicos feitos a partir de pixels. Este estilo vem desde arquiteturas de vídeo games da chamada era de 8 ou 16 bits (nome dado em função da palavra das suas CPUs na época). Normalmente tais imagens costumam ter resolução bem pequena propositalmente. Em jogos digitais, tem sido bastante usada também, sendo o seu uso interessante também para jogos educativos. Na implementação proposta, há uma opção especial para suportar essa modalidade também.

Em termos de correlação com as tecnologias gráficas *Web* mencionadas mais adiante, o formato vetorial é utilizado pela tecnologia SVG e o formato de textitBitmap pelo textitCanvas.

2.2.2 Transformações Geométricas

Transformações Geométricas como Translação, Escala e Rotação, amplamente usadas na Computação Gráfica (HEARN; BAKER; CARITHERS, 2014), podem ser representadas através de matrizes. No caso de jogos gráficos 2D, essas matrizes tipicamente são trabalhadas com coordenadas homogêneas, tendo dimensão 3x3. De forma correspondente, pontos são representados como uma matriz-coluna 3x1. Assim, as matrizes são aplicadas sobre os pontos e causam neles a devida transformação geométrica intencionada.

Essas matrizes de transformação são bastante usadas porque permitem um tratamento homogêneo, onde facilmente se acumulam rotações, translações, etc. numa só matriz. Essa matriz então é aplicada sobre todos os pontos que compõem um determinado objeto, obtendo-se assim o objeto transformado como um todo.

Dessa forma, um ponto 2D pode ser representado pela seguinte matriz:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

É muito comum a necessidade de mover objetos para diferentes posições durante a execução do jogo. Caso deseje-se mover um conjunto de pontos de um objeto em uma direção sem alterar a distância entre as partes que compõem o objeto, pode-se usar uma mesma matriz de translação para todos os pontos. A matriz que aplica uma translação (+x, +y) sobre um ponto pode ser definida da seguinte forma:

$$\begin{bmatrix} 1 & 0 & x \\ 0 & 1 & y \\ 0 & 0 & 1 \end{bmatrix}$$

Nem sempre as imagens ou objetos que compõem a tela possuem o tamanho correto de antemão, sendo às vezes necessário torná-los maiores ou menores. Mudar a escala de um componente pode se mostrar útil também em aumentar o reuso de recursos visuais, como texturas, permitido que esses sejam reencaixados em diferentes contextos. Para aplicar a transformação de escala (x, y) sobre um objeto, a seguinte matriz de escala pode ser usada:

$$\begin{bmatrix} x & 0 & 0 \\ 0 & y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Por fim, diversas imagens e objetos possuem sentido ligado diretamente a direção, sendo difícil fazer uso eficiente desta imagem sem alterá-lo. Por exemplo, uma seta que aponta para baixo pode ser reposicionada para qualquer outra direção apenas sendo rotacionada, evitando assim a necessidade de criar uma seta para cada novo ângulo desejado. Assim, para aplicar uma rotação de θ graus pode-se usar a seguinte matriz de rotação:

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

A implementação da proposta faz uso de todas essas matrizes de transformações para fins de manipulação dos objetos gráficos.

2.2.3 Animações e Transições

Em jogos digitais é muito comum a presença de animações que modificam o estado visual dos objetos na tela, comunicando uma mudança de estado do objeto ou criando sentido através da troca contante entre estado em uma espécie de *loop*. Por exemplo, no sistema proposto é possível modificar a textura de um objeto, seguindo uma sequência de enquadramentos na imagem do objeto, apenas uma vez ou através de *loop*; ou apresentar o texto de uma entidade de forma gradual; ou ainda a transição entre telas, com a imagem de uma tela se movendo para fora e outra entrando na exibição. Animações assim podem ser feitas com um *loop* que vai gradualmente mudando a posição das imagens seguindo uma interpolação. Contudo, deve-se ter o cuidado de não se executar as iterações do *loop* de forma rápida (ou lenta) demais; para isso, é conveniente ter certo controle dos intervalos de tempo. Para isso, pode-se medir o tempo entre uma iteração e outra e verificar se a frequência de atualização está em

conformidade com a taxa de quadros desejada, isto é, com o chamado *Frames Per Second* (FPS). A implementação da proposta também adota isso.

2.3 Tecnologias *Web*

A atmosfera da *Web* é muito vasta, podendo apresentar muitas soluções diversas para certos problemas; algumas trazem uma novidade mas ainda de forma imatura; outras vêm e exploram melhor aquela novidade já num formato mais maduro, mas logo são superadas por outras. Nesse oceano tecnológico, porém, algumas tecnologias se mantêm estáveis, sendo a base sobre a qual as demais se desenvolvem. Assim, na criação de jogos educativos para o ecossistema *Web* torna-se importante delinear as tecnologias fundamentais que suas implementações estão usando e referenciando rotineiramente, sendo este o material a ser explorado à seguir.

2.3.1 Tecnologias Fundamentais: HTML, CSS, JavaScript

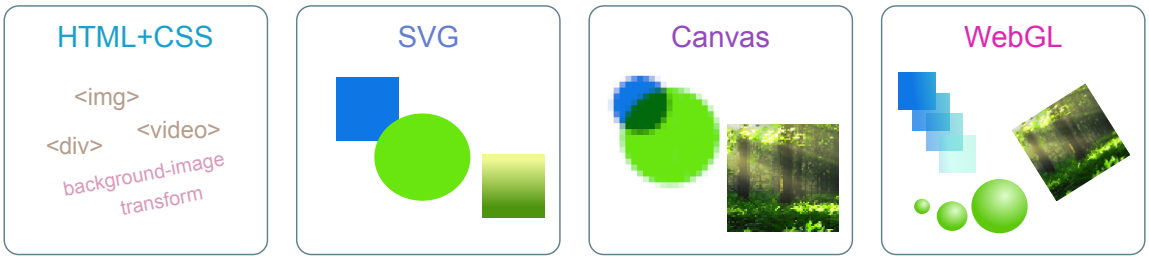
As tecnologias *Web* fundamentais são a tríade HTML, CSS e *JavaScript*. O *Hypertext Markup Language* (HTML) é a linguagem de marcação padrão de documentos que foram criados para serem exibidos em navegadores na *Web*. Ela é formada por blocos delimitados por *tags*. Esses blocos dão significado semântico ao conteúdo do documento, como, por exemplo, a *tag* <p> dá o significado semântico de parágrafo ao conteúdo dentro dela. O interior das *tags* pode variar de texto, imagem e vídeo (MUSCIANO; KENNEDY et al., 1996).

A tecnologia do *Cascading Style Sheets* (CSS), por sua vez, pode ser usada em conjunto com o HTML para descrever como as informações contidas nas *tags* serão apresentadas (MEYER, 2017). Ela foi criada para separar o conteúdo do estilo, cuidando de informações como cores, posicionamento do conteúdo na página e quais fontes usar para cada texto. Essa separação pode ajudar na personalização, flexibilidade e acessibilidade das páginas *Web*.

Por fim, *JavaScript* é a linguagem de programação padrão da *Web*, com elementos de programação orientada a objeto, criada com o objetivo de tornar a página *Web* dinâmica, definindo seu comportamento e possibilitando a interatividade com usuário (FLANAGAN, 2006). Por ser a linguagem padrão para desenvolvimento na *Web*, várias outras linguagens têm tido mecanismos que possibilitam compilar seus códigos para *JavaScript*. Certas linguagens já são propostas abertamente levando em consideração esse processo de transcompilação, como por exemplo TypeScript (BIERMAN; ABADI; TORGERSEN, 2014). Alguns frameworks mencionados no presente trabalho podem fazer uso de TypeScript também.

2.3.2 Tecnologias Gráficas: SVG, Canvas, WebGL

Figura 4 – Tecnologias Gráficas Fundamentais na Web



Fonte: Dos autores

É possível estruturar certos tipos de gráficos apenas com HTML e CSS, sem um uso específico de outras tecnologias. Por exemplo, podem ser usadas tags como <img>, , etc. para criar objetos gráficos, personalizando-os com as devidas propriedades em CSS. Com a adição de JavaScript, esses elementos poderão ganhar um movimento e reagir a eventos.

Contudo, trabalhar com HTML e CSS de "forma pura" para gráficos no contexto de um jogo não é a forma mais adequada. Para esse e outros fins que demandam um tratamento mais adequado pensando-se em gráficos, soluções mais específicas foram projetadas, como o padrão SVG, Canvas e a *WebGL* (ver Figura 4).

Durante muito tempo, desenvolvedores tentavam usar *plug-ins* para funcionalidades mais próximas de jogos na Web. Contudo, muitos desses plug-ins eram proprietários e cheios de *bugs*, sendo gradualmente substituídos por tecnologias gráficas nativas e padronizadas dos navegadores. Dessa forma, muitos passaram a adotar gradualmente SVG, Canvas e *WebGL* para fins que antes eram feitos, às vezes, por meio de plug-ins. Hoje, contudo, os navegadores modernos mais populares já suportam trabalhar com as três tecnologias.

O *Scalable Vector Graphics* (SVG) é um formato de imagem vetorial para gráficos 2D baseado em XML (FERRAIOLO; JUN; JACKSON, 2000). Ele possui suporte a interatividade e animações. O formato foi criado pela *World Wide Web Consortium* (W3C), tendo uma história desde 1999. O SVG pode se integrar às páginas *Web* através de código HTML, com uso da *tag* <svg>, oficialmente adicionada no HTML5. Os atributos podem ser estilizados através de CSS, como qualquer outro elemento de HTML, e com o uso do JavaScript é possível até criar animação e eventos interativos.

Já o *Canvas* é um dos elementos do HTML 5 que permite a renderização dinâmica de formas e imagens *bitmap* (FULTON; FULTON, 2013). O elemento *Canvas* é configurado para uso com um determinado *contexto gráfico*. Através desse contexto, pode-se usar a chamada *Canvas* API, que contém comandos *JavaScript* para desenho 2D no *Canvas*; ou pode-se usar um outro contexto gráfico para ativar o uso da *WebGL* (SHAPPIR, 2012). Convém lembrar que o *Canvas* trabalha especificamente com pixels, isto é, o seu objeto é armazenado como uma imagem de *Bitmap* na tela.

A *WebGL* é uma API baseada na *OpenGL* ES para JavaScript implementada nos modernos e mais populares navegadores (ANGEL; SHREINER, 2014). A *WebGL* 1 é baseada na *OpenGL* ES 2 e a *WebGL* 2 na *OpenGL* ES 3. A *OpenGL*, por sua vez, é um padrão amplamente usado em sistemas gráficos, presente em diversos computadores e sistemas há décadas. A *WebGL* é como uma "versão" da *OpenGL* que executa dentro do navegador Web.

Por meio da *WebGL*, pode-se alcançar um uso mais efetivo das *Graphics Processing Units* (GPU), ou Unidades de Processamento Gráfico), um *hardware* especial maciçamente paralelo para processamento gráfico presente em diversos computadores (OWENS et al., 2008). As modernas GPUs programáveis permitem que o programador possa fornecer códigos que serão executados pelas mesmas de forma paralela, permitindo uma execução de desempenho muito superior do que um código serial nas CPUs tradicionais. A natureza de projeto das GPUs, voltadas para explorar um alto nível de paralelismo, podem ser usadas por outras tarefas para além da computação gráfica (NICKOLLS; DALLY, 2010), como tarefas relacionadas à cálculos diversos de álgebra linear e inteligência artificial, entre outros. Porém, seu mais popular uso se dá na aplicação para jogos, contemplando assim também *game engines* e *frameworks*. De fato, o popular framework Three.js[1], projetado para aplicações gerais de computação gráfica 3D, implementa muitos códigos em cima da *WebGL*.

Vale lembrar que a forma de acessar a *WebGL* se dá por meio do elemento Canvas também, mas fornecendo um outro tipo de contexto na hora de criar o mesmo, assim estabelecendo o mecanismo de acesso à API da *WebGL*. Apesar de a *WebGL* ser considerada uma API mais de "baixo nível" diante de outras soluções gráficas, é a que dá mais poder para o programador; este, por sua vez, terá que escrever mais códigos e manipular os dados de forma mais manual, mas os resultados poderão ter um desempenho superior e, além disso, a implementação já fica numa infra-estrutura que permite muitas futuras personalizações. O presente trabalho usará primariamente a *WebGL*.

[1] https://threejs.org

2.3.3 Descrição de Dados: XML, JSON, YAML

A forma como dados são descritos é um tópico amplo na computação. No ambiente da *Web*, também várias formas já foram abordadas. Dois formatos de base tiveram ampla adesão na *Web*: XML e JSON. Esses formatos tiveram usos diversos para armazenar, enviar e representar dados diversos.

O padrão *Extensible Markup Language* (XML) faz parte das *Standard Generalized Markup Languages* (SGMLs), sendo usado para documentos de linguagens genéricas de marcação. Foi criada em 1998 pela XML *Working Group*, com o objetivo de ser usado diretamente na *internet*. Tecnicamente um documento XML é formado por entidades que podem se referir a outras entidades, sendo uma dessas a entidade raiz; assim sendo, é possível fazer uma representação de informações em um esquema similar a um grafo (BRAY et al., 2000). Alguns *frameworks* permitem trabalhar com dados no formato XML. Contudo, tem havido maior adoção em anos recentes do formato JSON.

O *JavaScript Object Notation* (JSON)[2] é um formato padrão que usa texto legível para seres humanos para transmitir e armazenar dados consistindo em pares de atributo e valor, e listas. O formato é frequentemente usado no contexto *Web*, uma vez que é baseado na própria linguagem *JavaScript*.

O JSON é um formato mais leve que o XML para muitas coisas. Além disso, caso um desenvolvedor optasse por outro formato à parte do JSON tipicamente teria que encontrar outras bibliotecas que implementassem um devido suporte ou então implementar ele mesmo tal suporte. Assim, devido à sua forte presença universal nos sistemas atuais e não necessidade de bibliotecas extras, a proposta do presente trabalho adota o formato JSON.

Outro formato ainda foi considerado na pesquisa: o YAML *Ain't Markup Language* (YAML), uma linguagem descritiva simples de ser lido por humanos e com algumas inspirações em *Python* (BEN-KIKI; EVANS; INGERSON, 2009). Apesar de seu apelo de facilidade de uso, o formato ainda não é tão usado como o JSON, nem é nativo de várias linguagens, sendo necessário o suporte de bibliotecas extras.

2.4 Padrões de Projeto e Código

Em jogos digitais, assim como em muitas áreas da computação, há vários padrões de projeto, de programação e organização de estruturas diversas que atendem a necessidades frequentes das aplicações desse domínio. Por meio de um exame de

[2] https://www.json.org/json-en.html

implementações diversas, dois padrões de códigos frequentemente usados (e também usados na implementação da proposta) são apresentados à seguir.

O Entity Component System (ECS) é um padrão arquitetural de código focado em *cache hits*, isto é, mantendo os dados de tal forma que sua leitura e escrita sejam otimizadas, ainda que se utilize mais memória do que um programa tradicional consumiria. Para que o padrão seja seguido, os dados devem estar separados dos comportamentos, sendo os dados chamado componentes (*Component*), listas tipadas de objetos, classes e abstrações para elementos do programa, e os comportamentos sistemas (*System*) que consomem os objetos indiretamente através de entidades (*Entity*). Se os componentes estão armazenados num mesmo endereço, eles pertencem a uma mesma entidade. Este padrão tem sido bastante utilizado no desenvolvimento de certos jogos, sobretudo em linguagens com *Garbage Collector*, como o *JavaScript*, contribuindo para a otimização.

Além do ECS, o trabalho usa em sua implementação as chamadas Listas Tipadas ou *typed arrays*, que são uma categoria específica de lista que armazena apenas objetos do mesmo tipo. A vantagem dessa categoria é que todos os objetos ocupam exatamente o mesmo tamanho na memória, estando todos juntos um após o outro em sequência. Assim, os itens podem ser otimizados na hora do acesso e também o gerenciamento de memória da lista é simplificado.

3 Trabalhos Relacionados

3.1 Visão Geral

Seguindo as considerações metodológicas estabelecidas no início deste documento, um conjunto de trabalhos foi selecionado. Para fins de visão geral dos trabalhos relacionados, os *frameworks* com fontes foram agrupados no diagrama da Figura 5, que inclui os nomes de cada trabalho com o respectivo endereço de seu projeto, e já inclui também a presente proposta em sua lista.

Diante de um ecossistema notável e sempre crescente de *frameworks* no contexto de desenvolvimento *Web*, convém mencionar de forma breve determinados *frameworks* (alguns populares) que ficaram fora do escopo do trabalho, uma vez que não contemplam os requisitos mínimos delineados, além de comentar alguns casos particulares:

- *Frameworks* voltados para visualização científica de dados, como D3.js[1] e eChartJS[2].
- O popular *framework* ThreeJS[3], voltado para trabalhar com gráficos 3D, assim como PlayCanvas[4], BabylonJS[5] e Voxel.js[6].
- O *framework* iLearnTest[7] é voltado para jogos educativos, contudo não trabalha com gráficos, ficando um pouco à parte dos demais trabalhos considerados. Apesar disso, é brevemente mencionado por sua proximidade na área de educação.
- O *framework* Boardgame.io[8] é voltado para jogos baseados em jogadas de turnos, podendo até eventualmente ser usado num jogo educativo — contudo, também não trabalha diretamente com gráficos, ficando também um pouco mais deslocado dos demais trabalhos. Em todo caso, será brevemente comentado.
- Tecnologias proprietárias, isto é, que não são *software* livre, como GameMaker[9], também foram cortados. O Construct[10], entretanto, torna-se um caso particular

[1] https://github.com/d3/d3
[2] https://github.com/apache/echarts
[3] https://github.com/mrdoob/three.js/
[4] https://github.com/playcanvas/engine
[5] https://github.com/BabylonJS/Babylon.js
[6] https://github.com/maxogden/voxel-engine
[7] https://github.com/vemu/iLearn
[8] https://github.com/boardgameio/boardgame.io
[9] https://www.yoyogames.com/en/gamemaker
[10] https://www.construct.net/en

que ainda será comentado entre os trabalhos relacionados.

- *Frameworks* voltados para simulação física também estão fora do escopo pretendido, como Matter.js[11] e Planck.js[12].
- *Frameworks* voltados para um sistema operacional específico como iOS ou Android foram também prontamente descartados.

A seguir, os trabalhos relacionados são então brevemente comentados.

3.2 Frameworks

3.2.1 Boardgame.io

Boardgame.io[13] é um *game engine open source*, que usa *JavaScript*, baseada em turnos. Toda a lógica do jogo é baseada em funções que descrevem como a lógica do jogo muda à partir de um determinado movimento. Este *fremework* possui suporte a criação de jogos múltiplos jogadores online, inteligência artificial para jogar automaticamente e extensibilidade através de *plug-ins*. Além da *JavaScript engine*, o projeto também usa *Python* para criar o *bot* que joga o jogo automaticamente e React e Socket.io para comunicação entre os módulos de cliente e servidor. Também tem documentação disponível, exemplos e sítio para a comunidade no repositório do Github.

3.2.2 Construct

Construct é uma *game engine* para criação de jogos 2D em HTML5 criado pela *Scirra*. Essa biblioteca é de propósito geral, isto é, não foca em uma modalidade de jogo específica. Permite a criação de jogos para não-programadores através de uma interface gráfica; porém, como ocorre em outros sistemas semelhantes, os recursos oferecidos pela interface gráfica muitas vezes são inferiores ao que a programação direta pode oferecer. A empresa por trás do projeto passou a dar suporte à linguagem *JavaScript* também (desde a versão 3 do *Construct*), o que aumentou a acessibilidade proporcionada pela *engine*. Além disso, o *Construct* tem uma comunidade ativa, tornando-o numa das ferramentas mais populares para desenvolvedores de jogos desse perfil. Entretanto, a partir da segunda versão o programa deixou de ter

[11] https://github.com/liabru/matter-js
[12] https://github.com/shakiba/planck.js/
[13] https://boardgame.io/

Figura 5 – Trabalhos Relacionados

Framework	Fonte
Boardgame.io	https://boardgame.io/
Crafty	https://craftyjs.com/
Enchant.JS	https://github.com/wise9/enchant.js
FrozenJS	https://github.com/iceddev/frozen
GDevelop	https://gdevelop-app.com/
iLearnTest	https://github.com/vemu/iLearn
ImpactJS	https://impactjs.com/
Kiwi.js	https://github.com/gamelab/kiwi.js
LimeJS	https://www.limejs.com/
MelonJS	https://www.melonjs.org/
Panda Engine	https://github.com/ekelokorpi/panda-engine
Phaser	https://phaser.io
PixiJS	https://www.pixijs.com
Turbulenz	https://github.com/turbulenz/turbulenz_engine
EJET	https://github.com/edgarsay/EJET

Não atendem aos requisitos obrigatórios:

Gráficos de Visualização Científica:

- D3.js
- eChartJS

Simulação Física:

- Matter.js
- Planck.js

Gráficos 3D:

- ThreeJS
- PlayCanvas
- BabylonJS
- Voxel.js

Não Software Livre:

- GameMaker
- Construct

Fonte: Dos autores

uma licença *open source*, e a primeira versão só pode gerar jogos nativos, utilizando a linguagem *Python*.

3.2.3 CreateJS

CreateJS (*Create JavaScript*) é uma coleção de bibliotecas *JavaScript* e ferramentas que funcionam juntas para criar conteúdo interativo na internet, podendo ser usada para a criação de jogos. Ela é composta por 4 módulos: EaselJS, TweenJS, SoundJS, PreloadJS (MANDERSCHEID, 2014).

EaselJS (*Ease JavaScript*) O EaselJS fornece soluções para trabalhar com gráficos e interatividade com o HTML5 Canvas. Ele fornece uma API que é familiar aos desenvolvedores do *Flash*, incluindo uma lista de exibição hierárquica (MANDERSCHEID, 2014).

TweenJS (*Tween JavaScript*) TweenJS é uma biblioteca de interpolação para uso em *JavaScript*. Foi desenvolvida para integrar-se à biblioteca EaselJS, mas não depende dela. Ele suporta interpolação de propriedades de objetos numéricos e propriedades de estilo CSS (MANDERSCHEID, 2014).

SoundJS (Sound *JavaScript*) O SoundJS (*Sound JavaScript*) trata da reprodução de áudio via HTML5, *WebAudio* e *Flash* usando um modelo de *plug-in* que consulta recursos e seleciona um *plugin* adequado para funcionar em várias plataformas na maioria dos navegadores (MANDERSCHEID, 2014).

PreloadJS (Preload *JavaScript*) PreloadJS (*Preload JavaScript*) é uma biblioteca para pre-carregar ativos, incluindo imagens (usando um modelo de plug-in e SoundJS), sons, *JavaScript*, dados de texto etc. Ele usa XHR (XMLHttpRequest) sempre que possível e volta ao carregamento baseado em *tags* se necessário. Permite várias filas e várias conexões (MANDERSCHEID, 2014).

3.2.4 Crafty JS

Crafty[14] é uma biblioteca totalmente escrita em *JavaScript* para auxiliar na criação de jogos de maneira estruturada. Implementa o padrão de arquitetura ECS, utiliza a API *WebGL* para renderizar gráficos e possui um sistema de eventos e manipulação de elementos HTML, além de rotinas de desenho na tela. Além do *software*, disponibiliza também um fórum, documentação e módulos criados pela comunidade.

[14] https://craftyjs.com/

3.2.5 EnchantJS

EnchantJS[15] é um *framework* simples para a criação de jogos e aplicativos. Não possui dependências, usando apenas a *JavaScript Engine*. É orientado a eventos e possui uma arquitetura de criação de gráficos baseada em árvores de objetos, o que é comum em vários sistemas gráficos tradicionais. Para além do código, apresenta documentação em inglês, alemão e japonês. É notável o abandono deste *framework* pelos autores, uma vez que o próprio sítio oficial, indicado no repositório do *Github*, não mais está disponível e a última atualização foi realizada no ano de 2015.

3.2.6 Frozen

Frozen[16] é uma *engine open source* para criação de jogos HTML5 através de ferramentas e modularização. Essa *engine* utiliza uma variedade de ferramentas, além da linguagem *JavaScript engine* do navegador, como *Node, Grunt, Lo-Dash, Hammer.js, Dcl, Box2d, Dojo, JSDoc* (para gerar documentação), *Jasmine* (testes automatizados) e AMD.

Em comparação às outras *engines* citadas neste trabalho, com *Frozen* é possível perceber a elevada quantidade de dependências vinculadas à sua biblioteca, além de não apresentar documentação, pois o sítio indicado no repositório não é encontrado, e sua última atualização foi disponibilizada no ano de 2013.

3.2.7 GDevelop

GDevelop[17] é uma *game engine* de propósito geral com *interface* gráfica e *open source* para o desenvolvimento de *software*, permitido criar jogos 2D *Web* e nativos sem nenhum conhecimento em uma linguagem de programação específica. Toda a lógica do jogo é criada através de um sistema baseado em eventos, de modo tal que é possível criar um jogo sem uma única linha de código, o que torna a plataforma amigável para iniciantes, porém bastante limitada para a criação dos jogos próprios jogos em si, restritos nesse caminho. A *game engine* é baseada na *WebGL* e possui dependências como textitPixi.js e textitReact.js além de algumas partes escritas em C++, com *WebAssembly*. Para além do código, a plataforma possui documentação, fórum e *wiki*.

[15] https://github.com/wise9/enchant.js
[16] https://github.com/iceddev/frozen
[17] https://gdevelop-app.com/

3.2.8 iLearnTest

O *iLearnTest* é um *framework* para a criação de jogos educativos, baseado em *templates* (PAIVA et al., 2016). Durante o desenvolvimento do jogo, ele separa o conteúdo educativo das mecânicas do jogo incorporando o conteúdo, que é recebido através de um XML. Durante a partida permite ao usuário visualizar a pontuação conquistada através de respostas corretas. O *iLearnTest* foi criado em cima do *Construct*2, tendo como objetivo principal minimizar o tempo gasto e o conhecimento necessário para criar um jogo educativo.

3.2.9 ImpactJS

O *ImpactJS*[18] é uma *game engine* voltada para criação de jogos HTML com gráficos 2D através de um conjunto de ferramentas para renderização gráfica, gerenciamento de música e compatibilidade com outra biblioteca, *Box2D Physics*, para suporte de colisões físicas entre objetos. Para além do código, é notável a existência de documentação, fórum e tutoriais para auxiliar no entendimento e uso do *framework*.

3.2.10 Kiwi.js

O *Kiwi.js*[19] se trata de um *framework open-source* para a criação de jogos HTML5 para *desktop* e *mobile*, com o uso do *framework CocoonJS*, que permite a construção de aplicativos *mobile* à partir de HTML5, CSS3 e *JavaScript*. Ao invés de usar linguagens nativas como *Java* ou *Swift, CocoonJS* usa um categoria de compressão de uma página Web normal. O maior problema encontrado no *Kiwi.js* é o aparente abandono do projeto pelos desenvolvedores: o repositório do *Github* mostra que o último *commit* realizado foi feito em 2015 (RICHARDS, 2015). Além do código, o *Kiwi.js* possui um sítio oficial, indicado em seu repositório, e documentação, porém pode ser bastante difícil acessar essa informação devido ao estado atual do sítio, cheio de *bugs*.

3.2.11 LimeJS

LimeJS[20] trata-se de uma *game engine* para criação de jogos como experiência nativa para todos os modelos de telas com sensibilidade ao toque e navegadores *desktop*. O *framework* possui uma ferramente de linha de comando escrita em Python

[18] https://impactjs.com/
[19] https://github.com/gamelab/kiwi.js
[20] https://www.limejs.com/

que serve para instalar dependências, como *Box2D* e *Closure Library*, além de preparar o programa para distribuição e outras funcionalidades para desenvolvedores da *engine*. Porém não fornecem ferramentas direcionadas para o desenvolvimento do jogo em si. Além do código, a ferramente possui um mini tutorial em seu sítio, porém nenhuma documentação foi fornecida, apenas um link para um livro sobre *Closure* que é um forte dependência dessa *engine*.

3.2.12 MelonJS

MelonJS[21] é uma *engine open source* para criação de jogos HTML5 de propósito geral. Utiliza WebGL para renderizar gráficos 2D, suporta áudio, permite simulações físicas e conexões com *plug-ins*, como o PhysicsEditor. Não possui nenhuma dependência além da *JavaScript engine*. Para além do *software* básico apresenta também documentação e exemplos. O código é estruturado principalmente sobre o paradigma de programação orientada a objetos, o que o torna mais intuitivo, porém é menos eficiente e comum que o padrão de arquitetura ECS no contexto considerado de jogos.

3.2.13 Panda Engine

A *Panda Engine*[22] é *open source* e tem como foco jogos em HTML5 através de módulos diversos, que gerenciam arquivos, áudio e colisões físicas, entre outros. Internamente renderiza gráficos utilizando a biblioteca *PixiJS*. Além do núcleo é possível acessar a documentação, exemplos de código usando o *framework*, templates de jogos genéricos e utilizar uma plataforma, PANDA 2[23], específica para criação de jogos como a *Panda Engine*. Porém há de se ressaltar que essa plataforma em si não é gratuita e nem é *open source*.

3.2.14 Phaser

Desenvolvido pela *Photon Storm, Phaser* é um *framework* gratuito e *open source* para o desenvolvimento de jogos 2D HTML5. Ele usa renderização com *Canvas* ou *WebGL* e pode trocar entre eles automaticamente dependendo do suporte do *browser*. Pode ser compilado para iOS, *Android* e aplicações nativas com o uso de *softwares* de terceiros. O desenvolvedor pode usar tanto *JavaScript* como também *Typescript* (FAAS, 2017).

[21] https://www.melonjs.org/
[22] https://github.com/ekelokorpi/panda-engine
[23] https://www.panda2.io/

O *Phaser*[24] possui um *design* elogiado, com a devida documentação. Foi criado com base no *PixiJS*. Alguns problemas enfrentados no uso desse *framework* foram frequentemente associados à linguagem *JavaScript* para a qual o mesmo foi desenvolvido, o que fez com que vários desenvolvedores passassem a usar *TypeScript*.

3.2.15 PixiJS

O *PixiJS*[25] é uma biblioteca com estrutura de código muito similar ao *ActionScript*. Permite criar gráficos ricos e interativos, aplicações para múltiplas plataformas e jogos sem a necessidade de um conhecimento profundo na *WebGL* ou de lidar com questões de compatibilidade de navegadores e dispositivos. Caso a *WebGL* não seja suportado por alguma razão, o *PixilJS* pode trocar para o uso do *Canvas*. Ele se torna uma biblioteca atrativa para usuários de *ActionScript* por sua similaridade de estrutura. Segundo a *Goodboy*, empresa que o criou (SPUY, 2015), o *PixiJs* tem como principal vantagem a velocidade de renderização que faz com que muitas vezes ela seja utilizada por outros *frameworks* como *Phaser* (SPUY, 2015). Contudo, o *PixiJS* é uma biblioteca com foco mais na renderização gráfica em si, não possuindo funcionalidades mais específicas para o desenvolvimento de jogos.

3.2.16 Turbulenz

Turbulenz[26] é um *game engine* com API Node em *JavaScript* e *TypeSript* para criar e distribuir jogos 2D e 3D que podem rodar em qualquer plataforma que suporta HTML5. Na sua API de baixo nível este *framework* possui ferramentas para gráficos, matemática, física (2D e 3D), áudio, *networking*, *input*. E na sua API de alto nível apresenta abstração como telas, animações, gerenciamento e arquivos, *server requests*, *Canvas*, etc. Além do código, a *engine* também fornece uma ampla documentação e tutoriais.

3.3 Análise

A Tabela 2 apresenta os *frameworks* dos trabalhos relacionados que têm fontes no *Github* relacionados com uma variável à respeito do número de estrelas, que são votos positivos de usuário do GitHub, que têm em seus respectivos repositórios nessa plataforma.

[24] https://phaser.io
[25] https://www.pixijs.com
[26] https://github.com/turbulenz/turbulenz_engine

Como se pode notar das apresentações anteriores dos trabalhos relacionados, as implementações ou não suportam jogos gráficos, ou são *frameworks* de propósito geral, isto é, sem foco ou facilidades específicas para jogos educativos, ou não são *open-source*. Para criar jogos educativos muitos deles vão exigir um investimento maior do desenvolvedor para ter que lidar com várias estruturas que não são fornecidas, tornando-se menos amigáveis e também menos produtivos para este fim.

Por outro lado, um sistema para jogos educativos gráficos com suporte a templates e criação rápida de conteúdo pela *Web*, com toda a infraestrutura para participação de alunos e professores, abrange várias outras possibilidades não previstas por esses trabalhos.

Jogos educativos, frequentemente, empregam alguma forma de estratégia. De fato, estratégia é, muito frequentemente, um gênero usado por jogos educativos pela sua inerente versatilidade de conteúdos que podem ser abordados e têm a predisposição de fazer o jogador pensar (MILGROM; ROBERTS, 1990). Alguns modelos clássicos de jogos abrangidos pelo gênero estratégia envolvem jogos de tabuleiro com peças, jogo da memória, jogo de cartas, caça palavras, quebra-cabeças, etc. Estes modelos clássicos são moldáveis para diversos contextos e usos educacionais. Com o uso de um *framework* aberto e expansível focado em jogos educativos por meio da alimentação de templates prontos, criar uma experiência de ensino baseada nessas modalidades clássicas de jogo tem o potencial de permitir um uso ampliado e prático desse tipo de conteúdo complementar para aulas por parte de profissionais da educação, incentivando uma liberdade de criação de materiais dinâmicos devidamente adaptados às necessidades específicas de cada um.

Tabela 2 – *Frameworks* relacionados e Estrelas no Github

Projeto	Estrelas no GitHub
Boardgame.io	8.700
Contruct	657
CreateJS	178
Crafty	2.989
Enchant.JS	1.665
FrozenJS	1
Gdevelop	2.804
iLearnTest	0
ImpactJS	1.624
Kiwi.js	215
LimeJS	1.412
MelonJS	3.375
Panda Engine	671
Phaser	27.015
PixiJS	29.093
Turbulenz	3.052

Fonte: Dos Autores.

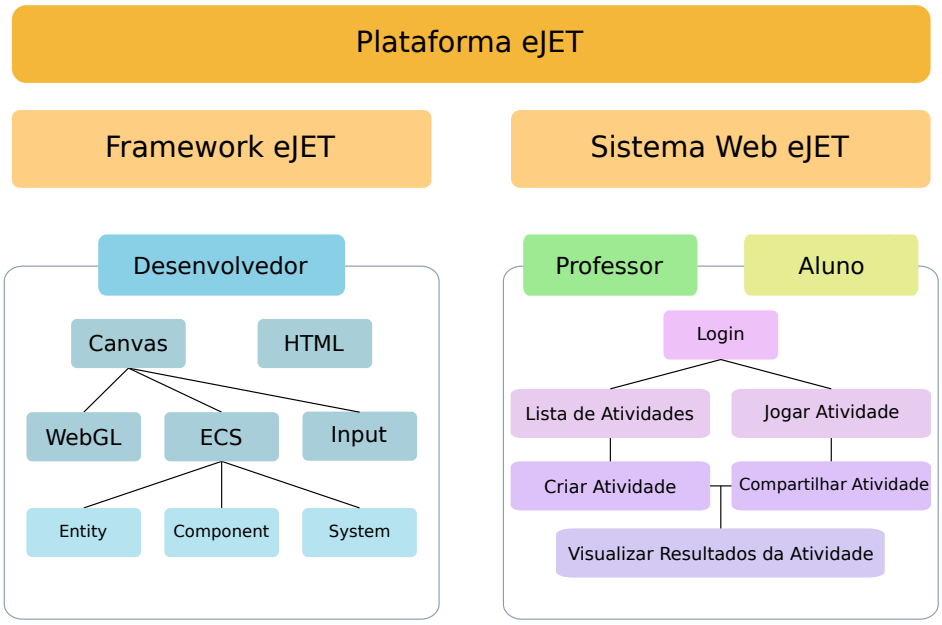
4 Proposta

“Cada problema que eu resolvia tornava-se uma regra que me servia para resolver outros problemas”

— René Descartes

Uma visão geral da arquitetura da plataforma eJET, incluindo o *framework* e o sistema propostos, é disponibilizada no diagrama da Figura 6.

Figura 6 – Visão Geral da Plataforma eJET



Fonte: Dos autores

Conforme se observa na Figura 6, a Plataforma eJET compreende duas partes: o *Framework* e o Sistema *Web*. O *framework* é a base usada para implementação dos jogos gráficos que permanecem imersos no sistema maior, a plataforma *Web*. Pelo sistema, o gerenciamento de contas e a mediação dos conteúdos são também oferecidos. Na Figura 6, são apresentados diferentes módulos do *framework* de potencial interesse para o desenvolvedor. Em relação ao sistema, são apresentadas diferentes telas de interesse para os usuários que vão utilizá-lo. Maiores detalhes seguem adiante.

4.1 Framework eJET

O *framework* proposto para desenvolvimento de jogos educativos é agora apresentado com mais detalhes. Devido à ênfase original em jogos de tabuleiro, adotou-se o nome de *framework* Edgar de Jogos Educativos de Tabuleiro (eJET). Por se tratar de um artefato para ser consumido por desenvolvedores, também a descrição desse artefato é feita para o público-alvo de programadores. Nesse caso, a exposição é justamente da arquitetura geral do código com uma breve documentação dos principais módulos do mesmo. O repositório oficial também está disponível para consulta, em `https://github.com/edgarsay/eJET`.

4.1.1 Módulo WebGL

Esse módulo trata de um conjunto de funções para acessar a *WebGL*[1]. Por meio da *WebGL* pode-se obter um desempenho gráfico superior na implementação. O seu uso pode ser mediado por uma forma simplificada de acesso através da função **ejetWebglInit** que inicializa os recursos necessários.

ejetWebglInit:

Descrição:

Inicia o modulo WebGL do *framework*.

Declaração:

```
function ejetWebglInit(canvasQuerySelector)
```

Parâmetros:

canvasQuerySelector (String) Caminho CSS para o contêiner do jogo.

Exemplo:

```
ejetWebglInit("#game");
```

[1] Foram usados vários nomes de comandos baseados na seguinte fonte: https://webglfundamentals.org/webgl/lessons/webgl-fundamentals.html

loadImageAndCreateTextureInfo:

Descrição:

Retorna um objeto contendo as informações de uma imagem, como *width* e *height*, e ajusta caso seja uma *pixelart*.

Declaração:

```
loadImageAndCreateTextureInfo(path, pixelart)
```

Parâmetros:

path (String)	Localiza a imagem.
pixelart (Boolean)	É uma *pixelart*?

Exemplo:

```
var sprite = loadImageAndCreateTextureInfo(
"./images/sprite.png", true) ;
```

drawImage:

Descrição:

Desenha uma textura no canvas.

Declaração:

```
drawImage(tex, texWidth, texHeight, srcX, srcY,
srcWidth, srcHeight, dstX, dstY, dstWidth, dstHeight)
```

Parâmetros:

tex (loadImageAndCreateTextureInfo)	Informações da textura.
texWidth (Number)	Largura da textura original.
texHeight (Number)	Altura da textura original.
srcX (Number)	Coordenada X do pixel onde se deseja começar o recorte na textura original.
srcY (Number)	Coordenada Y do pixel onde se deseja começar o recorte na textura original.
srcWidth (Number)	Largura em pixels do recorte, na textura original, para desenhar, opcional.
srcHeight (Number)	Altura em pixels do recorte, na textura original, para desenhar, opcional.
dstX (Number)	Posição, no eixo X, onde sera desenhada a textura no Canvas, opcional.
dstY Number)	Posição, no eixo Y, onde sera desenhada a textura no Canvas, opcional.
dstWidth (Number)	Largura da textura, ou recorte, desenhada no Canvas, opcional.
dstHeight (Number)	Altura da textura, ou recorte, desenhada no Canvas, opcional.

Exemplo:

```
drawImage(sprite, 32, 32, 0, 0);
```

drawShape:

Descrição:

Desenha uma forma geométrica, triângulo ou quadrado, no canvas.

Declaração:

```
drawImage[shape](x, y, width, height, rotation, color)
```

Parâmetros:

x (Number)	Posição, no eixo X, onde sera desenhada no Canvas.
y (Number)	Posição, no eixo Y, onde sera desenhada no Canvas.
width (Number)	Largura do desenho, em pixels.
height (Number)	Altura do desenho, em pixels.
rotation (Number)	Rotação do desenho, em radianos.
color (String)	Cor do desenho, no formato hexadecimal.

Exemplo:

```
drawImage['triangle'](0, 0, 50, 50, 0, '#FF0000');
```

createTextTextureInfo:

Descrição:

Retorna um objeto contendo as informações de uma textura, que por sua vez contém o texto especificado.

Declaração:

```
createTextTextureInfo(text, width, height, pixelart)
```

Parâmetros:

text (String)	Texto que será desejado dentro da textura.
width (Number)	Largura da textura.
height (Number)	Altura da textura.
pixelart (Boolean)	Caso se deseje pixelizar a fonte.

Exemplo:

```
var textura = createTextTextureInfo("Hellor World",
100, 25, false);
```

drawText:

Descrição:

Desenha uma textura com texto, **createTextTextureInfo**, no canvas.

Declaração:

```
drawText(textTexture, x, y, texColor)
```

Parâmetros:

textTexture (createTextTextureInfo)	Informações da textura.
x (Number)	Posição, no eixo x, onde sera desenhada no Canvas.
y (Number)	Posição, no eixo y, onde sera desenhada no Canvas.
texColor (String)	Cor do desenho, no formato hexadecimal.

Exemplo:

```
drawText(texture, 0, 0, "#000000");
```

4.1.2 Módulo ECS

Esse módulo trata da implementação do ECS (Entity Component System), um padrão de arquitetura usado em vários jogos, conforme discutido na fundamentação teórica do trabalho, visando fornecer flexibilidade e desempenho com base em velocidade de leitura e *cache hits*.

Classe: Entity

Descrição:

Classe principal que serve como um endereço para um conjunto de componentes.

Atributos:

id (Number)	Posição em que se encontram os componentes relacionados a entidade em um *Array*.
children (Array<Entity>)	Entidades que herdam características, como posição.
add (Function(component: Component))	Vincula um novo componente à Entidade.
get (Function(componetType: String))	Retorna um componente da Entidade.
set (Function(component: Component)	Redefine um componente da Entidade.
remove (Function(componetType: String)	Desvincula um novo componente da Entidade.

Exemplo:

```
var newEntity = new Entity().add(Component.transform());
newEntity.add(Component.shape('square', 700, 150,
'#655057'));
System.renderShape(newEntity);
```

Objeto: Component

Descrição:

Objeto que armazena um conjunto de funções que retornam objetos válidos como componentes.

Atributos:

transform (Function(x: Number, y: Number, scaleX: Number, scaleY: Number, rotation: Number))	Armazena a posição, escala e rotação da entidade.
motion (Function(speed: Array<Number>, acceleration Array<Number>))	Armazena a velocidade e aceleração da entidade.
sprite (Function(textureInfo: createTextTextureInfo, width: Number, height: Number, sourceX: Number, sourceY: Number, x: Number, y Number))	Armazena uma textura.
shape (Function(typeOfShape: String, width: Number, height: Number, color: Number, offSetX: Number, offSetY: Number))	Armazena uma forma geométrica.
text (Function(text: String, width: Number, height: Number, offSetX: Number, offSetY: Number, color: String, pixelart: Boolean))	Armazena um texto, e uma textura contendo o texto.
spriteAnimation (Function(sourceFrame: Array, fps: Number, loop: Boolean, stop: Boolean))	Armazena a informação de uma animação, baseada em recortes de um *sprite*.
textAnimation (Function(finalText: String, fps: Number, loop: Boolean, stop: Boolean))	Armazena informações de um animação, com base em um texto final.
hitBox (Function(x: Number, y: Number, width: Number, height: Number))	Armazena informações de um quadrado.

Exemplo:

```
var redSquare = Component.shape("square", 150, 150,
"#FF0000");
```

Objeto: System

Descrição:

Objeto que contém funções que consomem e modificam a informação de Componentes através de Entidades.

Atributos:

renderSprite (entity: Entity)	Desenha na tela a textura de uma entidade, caso ela possua os componentes *transform* e *sprite*.
renderShape(entity: Entity)	Desenha a forma geométrica que esta ligada à entidade, caso a entidade possua os componentes *transform* e *shape*.
renderText (entity: Entity)	Renderiza o componente de texto da entidade, caso possua um componente *transform* e *text*.
movement (entity: Entity, delta: Number, parentMotion)	Modifica a posição da entidade e seu filhos, com base nos componente de *transform* e *motion*.
animateSprite (entity: Entity, delta: Number)	Modifica a textura da entidade como base nas informações dos seus componentes *sprite* e *spriteAnimation*.
animateSprite (entity: Entity, delta: Number)	Modifica o texto, atualizando a textura, da entidade como base nas informações dos seus componentes *text* e *textAnimation*.

Exemplo:

```
System.movement(entity);
```

4.1.3 Módulo Input

Esse módulo contém um conjunto de mecanismos para lidar com entrada de dados através do *joystick*, *teclado* e *mouse*, contemplando assim algumas das principais formas de interatividade com um jogo digital.

ejetInputInit:

Descrição:

Inicia o módulo Input do *framework*.

Declaração:

```
ejetInputInit()
```

Parâmetros:

Exemplo:

```
ejetWebglInit("#game");
ejetInputInit();
```

Objeto: ejetInput

Descrição:

Objeto principal gerado pela função inicializadora **ejetInputInit**.

Atributos:

update (Function())	Atualiza as informações, como posição do *mouse*, estado das teclas e botões.
isDown (Function(combo: String))	Retorna **true** caso o **combo**, *String* representando um conjunto de teclas, tenha sido pressionado.
isPressed (Function(combo: String))	Retorna **true**, caso o **combo**, *String* representando um conjunto de teclas, tenha sendo pressionado.
onMouseOver (Function(entity: Entity, action: Function))	Chama a função **action**, caso o mouse esteja sobre o componente **hitBox** da entidade.
onClick (Function(entity: Entity, action: Function))	Chama a função **action**, caso o usuário clique no componente **hitBox** da entidade.

Exemplo:

```
ejetInput.update();
ejetInput.onClick(entity, function(){alert("click");};
```

4.1.4 Módulo HTML

Construtor createElement:

Descrição:

Construtor que consome uma *string* com o tipo informando o tipo de elemento que deve ser criado ou um objeto do tipo **Element** e retorna um objeto do tipo **Element** com novos comportamentos.

Atributos:

- ◇ setClassName (Function(className: String)) redefine o atributo **className** e retorna o próprio elemento.
- ◇ addClass (Function(className: String)) concatena um novo valor a o **className** já existente e retorna o próprio elemento.
- ◇ setClassName (Function(className: String)) redefine o atributo **className** e retorna o próprio elemento.
- ◇ addClass (Function(className: String)) concatena um novo valor a o **className** já existente e retorna o próprio elemento.
- ◇ setStyle (Function(styleObject: Object)) redefine as regras CSS, em formato de objeto, que serão aplicadas no elemento e retorna o próprio elemento.
- ◇ setInnerHTML (Function(innerHTML: any)) redefine o atributo **innerHTML** e retorna o próprio elemento.
- ◇ addAtTheEnd (Function(parent: HTMLElement)) Coloca o elemento como último filho do elemento **parent** e retorna o próprio elemento.
- ◇ addAtTheStart (Function(parent: HTMLElement)) Coloca o elemento como primeiro filho do elemento **parent** e retorna o próprio elemento.
- ◇ addElement (Function(element: HTMLElement)) adiciona um elemento, **element**, como ultimo filho e retorna o próprio elemento.
- ◇ clone (Function()) Retorna um clone do próprio elemento.

Exemplo:

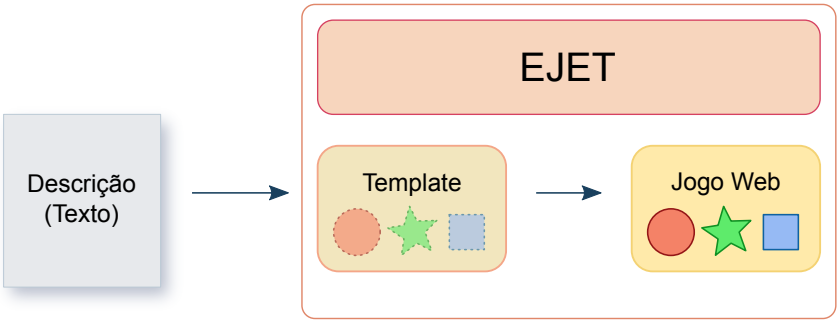
```
      var container = document.querySelector(
conteinerSelector);
      var background = createElement('div');
      background.setClassName(
'ejet-background-gray ejet-flex-center');
      background.setStyle( width:  '960px', height:  '470px' );
      background.addAtTheStart(container);
```

4.2 Sistema *Web* eJET

A proposta inclui o Sistema *Web* eJET, que utiliza o *framework* delineado anteriormente e constrói uma implementação *Web* completa com cadastro de usuários (alunos e professores) e as devidas operações relacionadas e jogos. Nesse sentido, o Sistema *Web* eJET pode ser visto como uma aplicação do *framework* ou ainda uma ferramenta ligada ao mesmo.

No sistema, alunos e professores podem ambos se registrar e logar. Uma vez logados, poderão acessar os recursos disponíveis. No caso, o professor poderá criar jogos a partir de um *template* existente, alimentando com uma descrição textual como será o jogo (Figura 7).

Figura 7 – Descrição textual alimentando um Template de Jogo



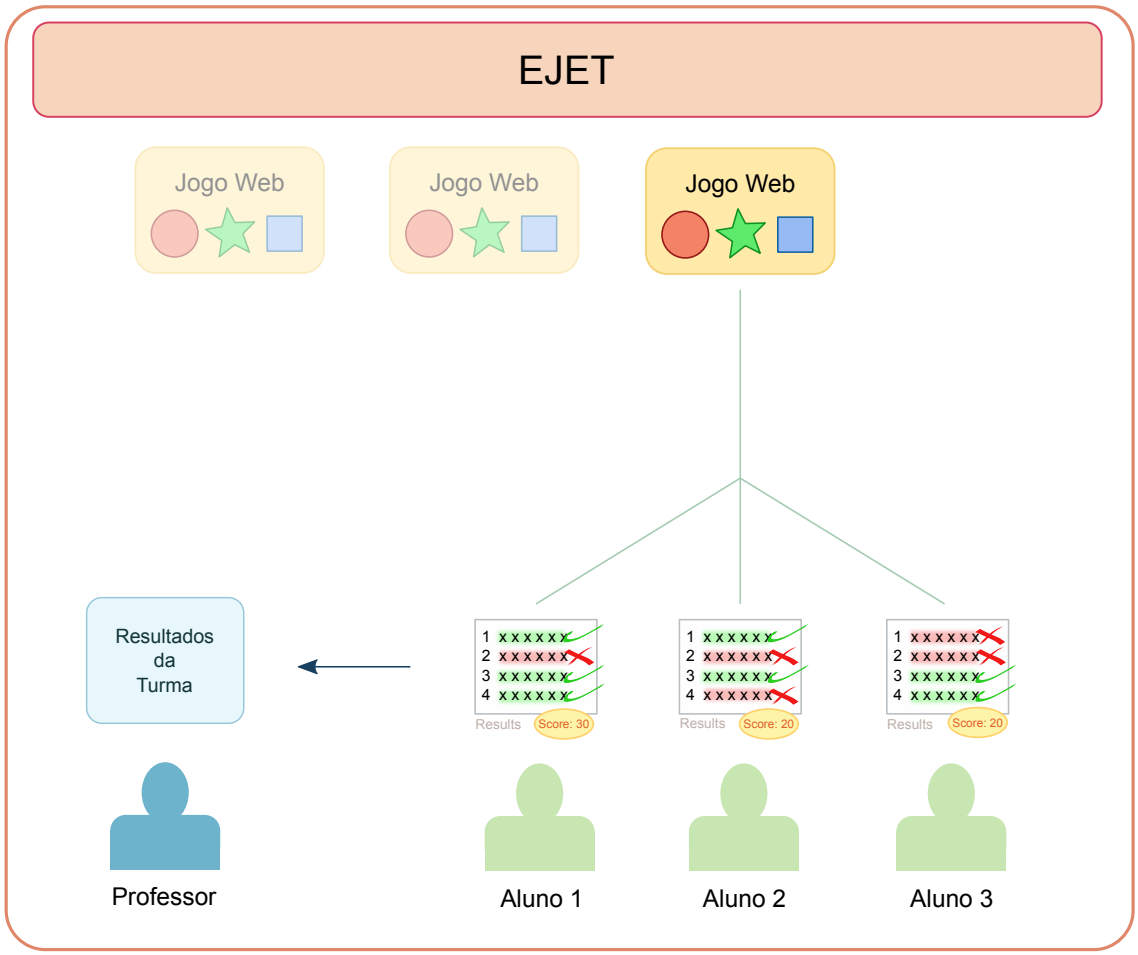
Fonte: Dos autores

O professor cria um jogo simplesmente escolhendo um *template* e fornecendo uma descrição do seu conteúdo desejado conforme a estrutura do *template*. Diferentes *templates* podem ter diferentes estruturas, ficando a cargo de cada *template* especificar como deve ser a estrutura da entrada. Em todo caso, adota-se, na presente implementação, o formato geral JSON para especificação dos dados. Além disso, para facilitar o uso, o sistema oferece também um exemplo abstrato de conteúdo que ilustre como um determinado *template* espera sua entrada. Assim, o professor pode simplesmente baixar o exemplo abstrato e colocar seu conteúdo específico sobre o mesmo, para depois submeter ao sistema.

Vários jogos poderão ser criados pelo professor. Os alunos podem entrar num

jogo e começar a sua atividade nele. Em cada jogo, os alunos terão seus resultados coletados e compartilhados com o professor que criou aquele jogo (Figura 8). Assim, o professor tem como acompanhar a situação de cada um em particular.

Figura 8 – Interação dos alunos com um jogo, com resultados coletados para o professor



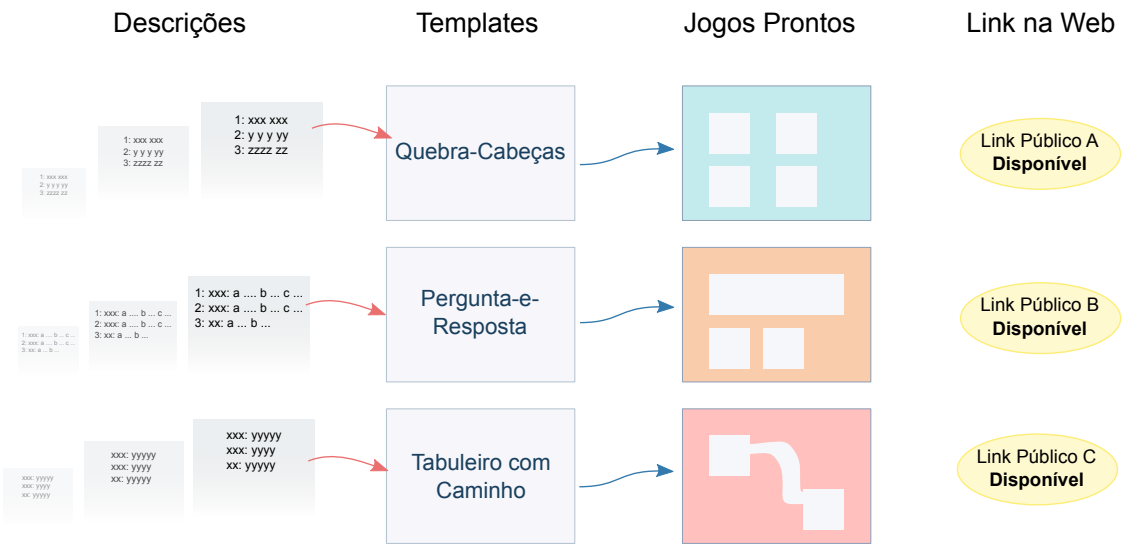
Fonte: Dos autores

Cada jogo gerado pelo professor é associado com um *link* público, que poderá ser compartilhado com os alunos. A partir do *link* fornecido, os alunos poderão interagir com o jogo.

Ao longo do tempo de uso da plataforma, pode-se ter uma visão geral caracterizada como apresenta a Figura 9: os professores vão elaborando descrições de conteúdo que, combinadas com um determinado *template*, definem como ficará o jogo para os alunos. Os jogos gerados por cada *template* podem ter naturezas bem distintas; todas, entretanto, alimentando-se de descrições textuais que o professor passa e

que formam o conteúdo educativo do jogo. Cada jogo gerado no sistema é criado com um *link* público único, que é compartilhado com os alunos e estes jogam aquele jogo, tendo seus resultados coletados para o respectivo professor.

Figura 9 – Professores alimentam *templates* com descrições de conteúdo, gerando jogos disponibilizados para alunos por meio de link único



Fonte: Dos autores

Ademais, o sistema suporta o gerenciamento de contas da seguinte forma. O professor com interesse no sistema deve inicialmente fazer o *login* (cf. Figura 10 mais adiante nos resultados). Caso não possua conta, deve se registrar (Figura 11). O registro é simples e rápido. Uma vez registrado e tendo feito *login*, o professor terá acesso a uma Tela Principal de Atividades (Figura 12), onde pode visualizar todos os jogos já criados por ele ou criar um novo jogo (Figura 13) ou ainda remover um jogo existente. Os jogos são criados selecionando-se um *template* pronto disponível e de jogo e submetendo um arquivo de texto simples, seguindo o modelo abstrato fornecido 14).

Para compartilhar a atividade, basta compartilhar o *link* gerado, havendo já um botão correspondente para isso. A ideia é que a atividade possa ser facilmente compartilhada por meio da postagem num aplicativo de comunicação ou numa rede social da turma, por exemplo. Quando um aluno, ou usuário qualquer do sistema, tiver respondido à atividade daquele *link*, o resultado é enviado para o professor e fica disponível através do botão vermelho em sua tela, para o criador da atividade.

Por sua vez, o aluno usa o sistema da seguinte forma: inicialmente, deverá acessar o *link* enviado pelo professor. Caso não esteja logado, isso lhe será apontado e assim o aluno deverá fazer o *login*. Caso não tenha conta, terá a opção de se registrar. Uma vez tendo registrado e logado no sistema, poderá então realizar a atividade apontada pelo professor.

Quando o aluno finalizar a atividade proposta, uma mensagem irá alertá-lo que seu resultado foi enviado para o criador da atividade, isto é, o professor, para propósitos de transparência.

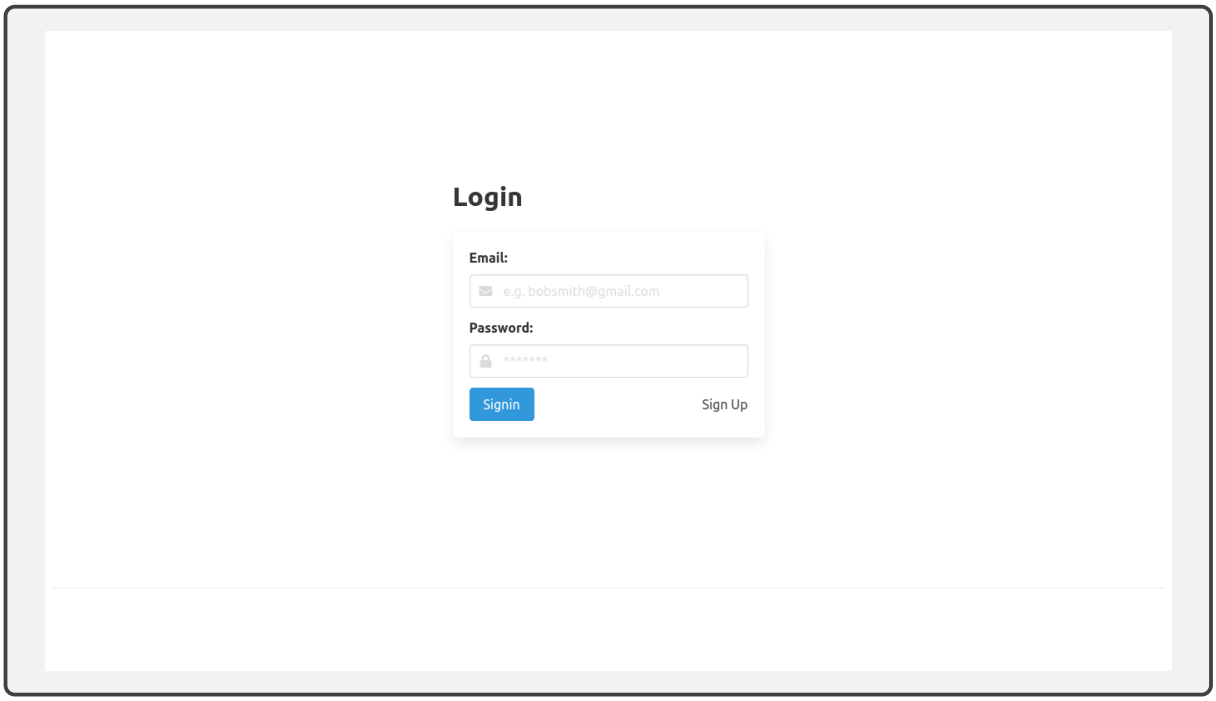
5 Resultados

“A educação é para melhorar a vida das pessoas e para deixar sua comunidade e o mundo melhor do que você encontrou.”

— Marian Edelman

Em sintonia com as descrições feitas no capítulo anterior da proposta, são apresentadas agora os resultados obtidos com a implementação da proposta.

Figura 10 – Tela de Login



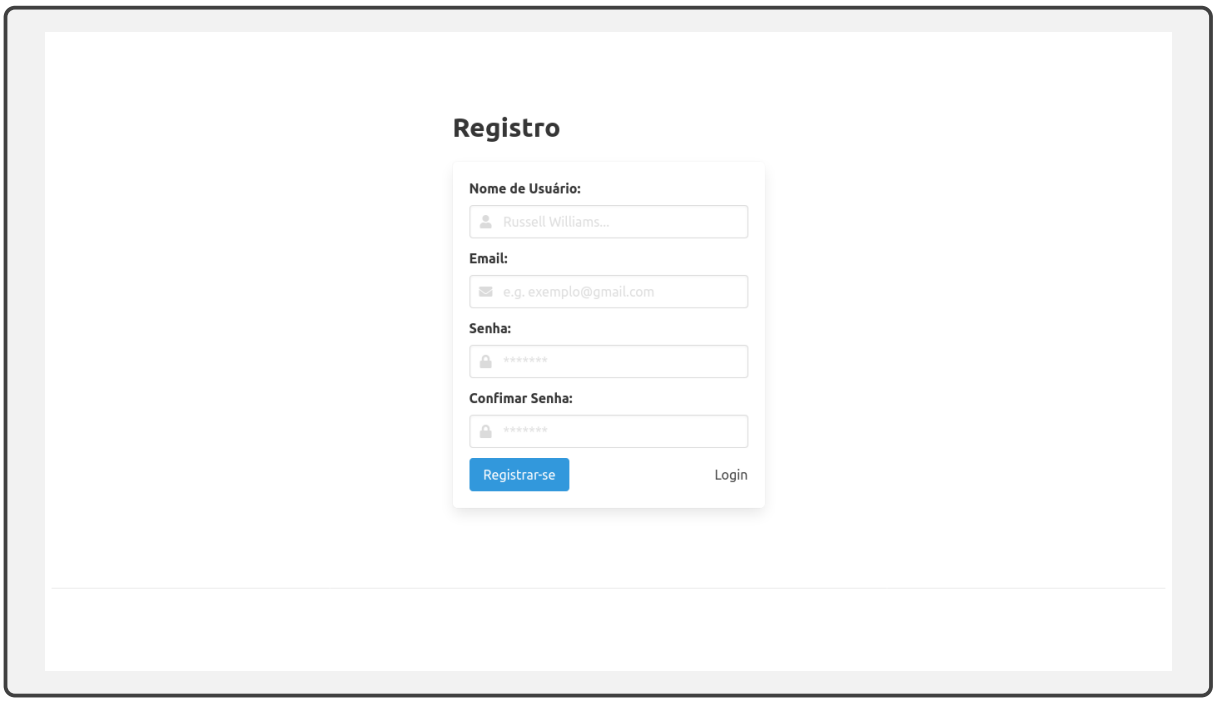
Fonte: Dos autores

As Figuras 10 e 11 mostram as telas do sistema implementado para a realização de Login e Registro de usuários (professores e alunos).

A Tela de Atividades (Figura 12) mostra, na perspectiva do usuário do sistema, todas as atividades criadas por ele. Através dessa tela pode-se criar uma nova atividade, como mostrado na Figura 13, remover uma existente ou acessar o *link* para compartilhar aquela atividade; e oferece ainda o botão de visualizar os resultados dos alunos (ou mesmo outros professores que possam ter respondido naquele jogo).

Na Figura 14 apresenta-se um exemplo abstrato de dados que pode ser usado para como entrada para um *template*, no formato JSON. O professor pode inserir conceitos e suas respectivas explicações; esses pares são então usados na geração da partida. Tendo criado uma atividade e compartilhado a mesma com outras pessoas, os usuários também podem visualizar o seu conteúdo (Figura 15).

Figura 11 – Tela de Registro



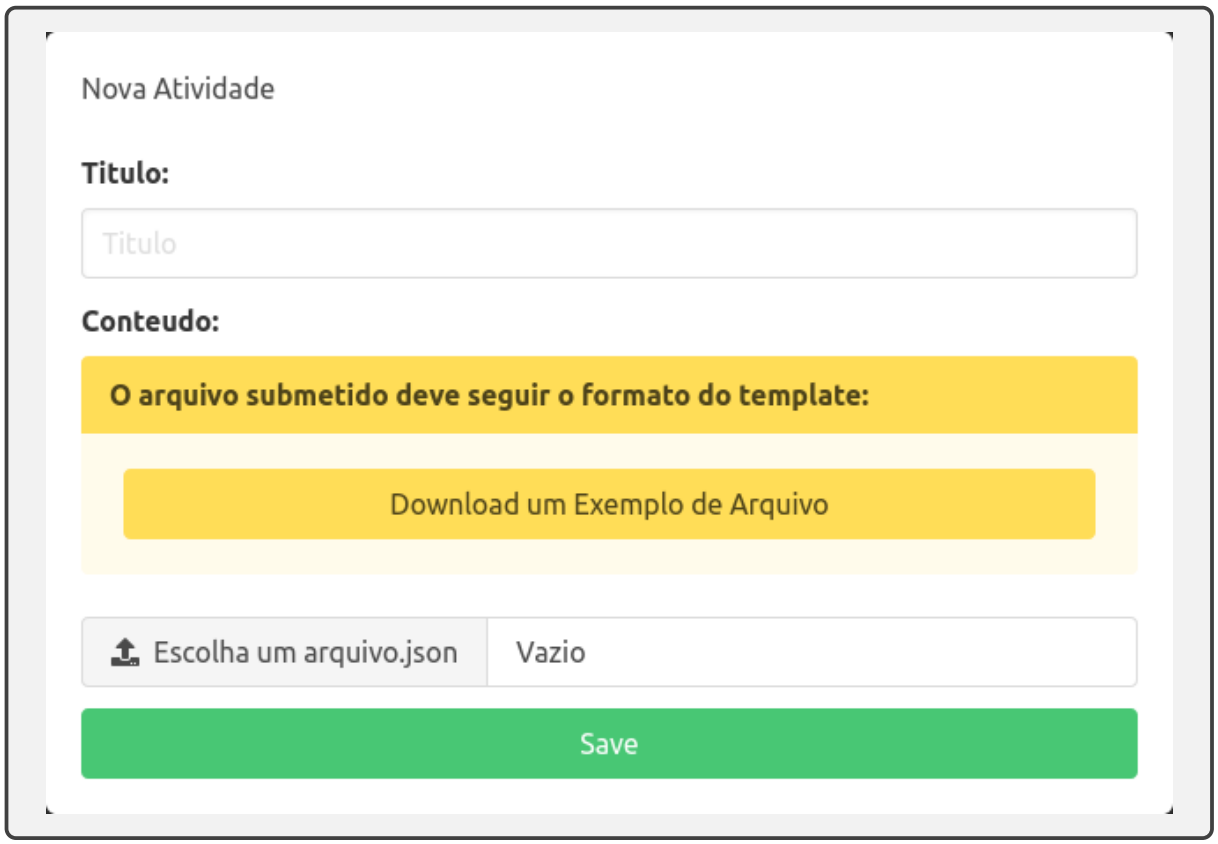
Fonte: Dos autores

Figura 12 – Tela de Atividades

Fonte: Dos autores

Quando o aluno finaliza a atividade, uma mensagem de informação é passada e um breve *Ranking* atual é exibido com as três melhores pontuações até aquele

Figura 13 – Caixa de Diálogo para Criar Nova Atividade



Fonte: Dos autores

Figura 14 – Exemplo de Arquivo JSON

```
{
    "Conteudo1": "Descrição do Conteudo1",
    "Conteudo2": "Descrição do Conteudo2",
    "Conteudo3": "Descrição do Conteudo3"
}
```

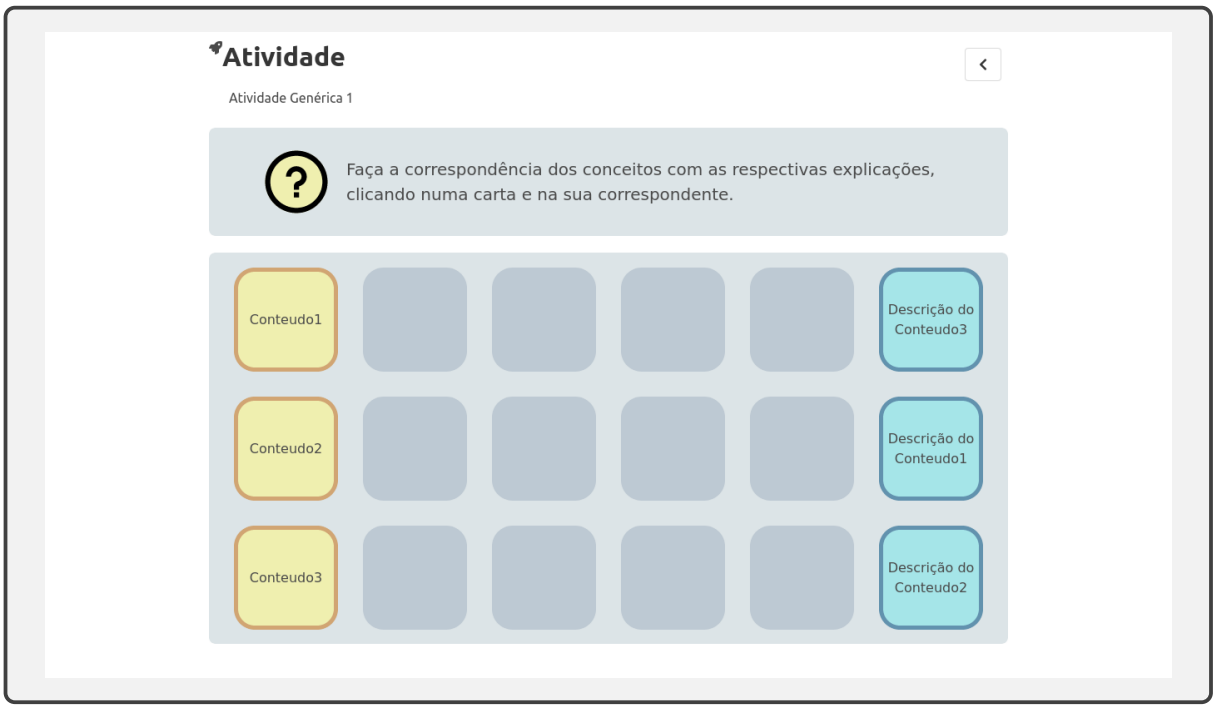
Fonte: Dos autores

momento (Figura 16). O *Ranking* é um importante ingrediente da gamificação na educação, por incentivar que os alunos tentem obter resultados melhores em competitividade de uns com os outros, servindo de desafio.

Os resultados contendo as pontuações de cada aluno são apresentados para o professor (Figura 17), que poderá baixar como um arquivo .CSV para fins de arquivamento ou análise posterior.

O resultado obtido com a implementação ilustra os conceitos propostos numa

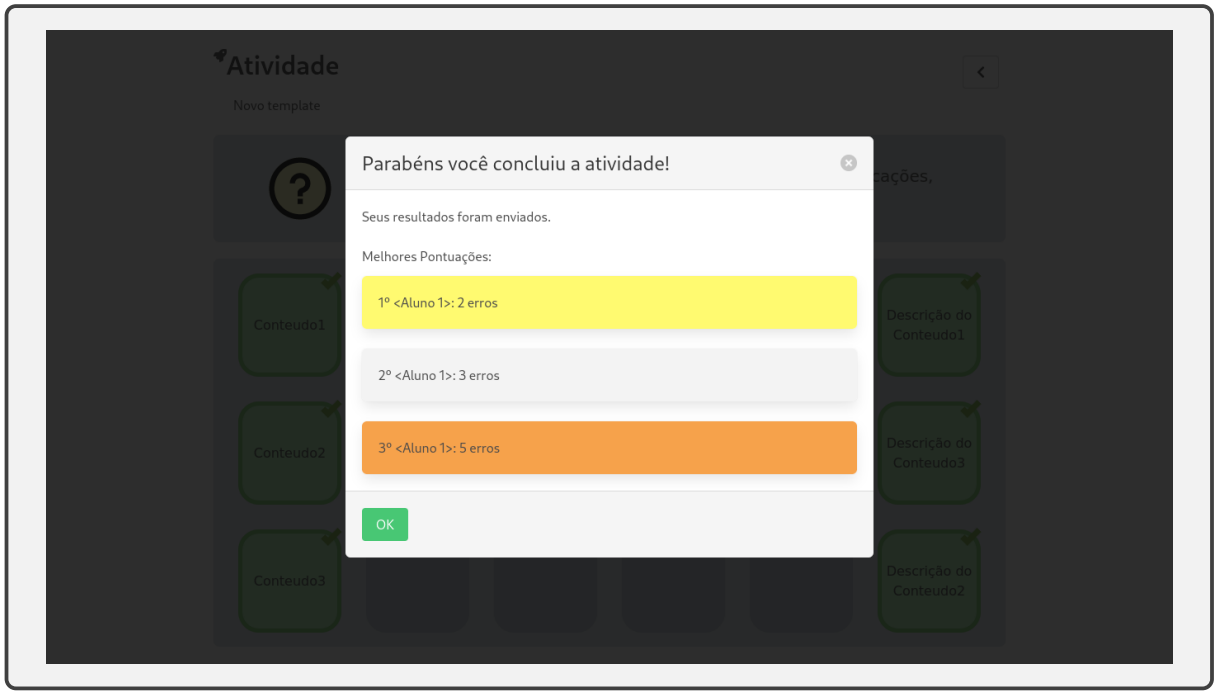
Figura 15 – Exemplo de Atividade Criada



Fonte: Dos autores

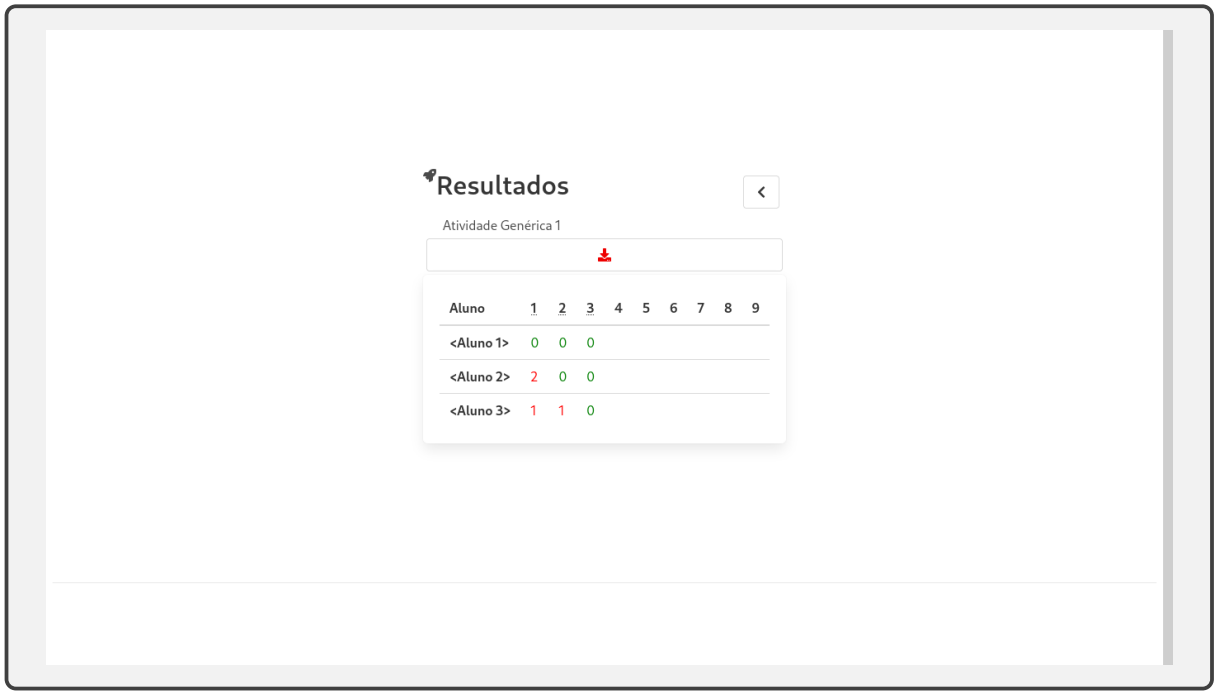
plataforma integrada com o *framework* usado como base para jogos, deixando o caminho aberto para que novas contribuições possam ser feitas com outros *templates*. O *framework* traz os elementos de base que podem ser usados para jogos educativos gráficos e reúne as características obrigatórias e desejadas estabelecidas previamente no trabalho. Assim, com base na pesquisa que explorou os diferentes *frameworks* relacionados, pode-se chegar no comparativo apresentado na Tabela 3, onde podem ser vistos que, apesar de alguns *frameworks* se aproximarem um pouco mais, nenhum realmente contemplou todas as características demarcadas conforme o trabalho proposto.

Figura 16 – Melhores Pontuações



Fonte: Dos autores

Figura 17 – Tela de Resultados



Fonte: Dos autores

Tabela 3 – Comparativo entre os *Frameworks*

Projeto	CD1	CD2	CD3
Boardgame.io	X	V	X
Contruct	V	X	X
CreateJS	X	X	X
Crafty	X	X	V
Enchant.JS	X	X	X
FrozenJS	X	X	X
Gdevelop	V	X	X
iLearnTest	V	V	X
ImpactJS	V	X	X
Kiwi.js	X	X	X
LimeJS	X	X	X
MelonJS	V	X	X
Panda Engine	X	X	X
Phaser	X	X	X
PixiJS	X	X	V
Turbulenz	V	X	X
eJET	V	V	V

Fonte: Dos Autores.

6 Considerações Finais

Diversos professores e profissionais da educação em geral estão sempre trilhando um caminho contínuo de melhorias para o processo educacional. Várias propostas são feitas buscando mediar um ensino mais rico e produtivo tanto para alunos quanto professores. A ideia de jogos educativos tem crescido e tido boa recepção em várias áreas e lugares. Se a produção de conteúdo de tais jogos puder ser mais fácil e automatizada para profissionais diversos da educação, ainda mais em meios digitais que sejam acessíveis a todos, abertos para modificação e expansão e sem custos adicionais, tanto melhor ainda poderá ser a situação, e mais facilmente poderão ser compartilhados conteúdos diversos, autorais e adaptados conforme a necessidade de cada contexto de ensino de ciência, tecnologia, arte e cultura. Assim, o presente trabalho buscou se inserir dentro desse contexto, fornecendo uma plataforma de jogos educativos chamada eJET que compreende um *framework* para desenvolvimento de jogos e um sistema *Web* ligado ao *framework*, com a infraestrutura necessária e pronta para o uso, envolvendo cadastro de alunos e professores, além de mediar a criação de jogos e o consumo e acompanhamento dos mesmos.

A plataforma ilustra a ideia da criação de conteúdo por parte dos professores por meio de uma alimentação de dados para *templates* prontos de jogos baseados em tecnologias *Web*. O sistema adotou um *framework* aberto e expansível, implementado com a tecnologia de base da *Webgl*, permitindo assim maior desempenho e estando inclusive aberto para possibilidades futuras de expansão com novos recursos e efeitos atingíveis por meio das GPUs. O uso de uma plataforma *Web* para isso permite um grande alcance de usuários. O jogo criado pelo professor fica disponível como um *link* público, que pode ser repassado aos alunos via *e-mail* ou aplicativos de comunicação e redes sociais, etc. Basta que os alunos acessem o link e poderão se cadastrar, jogar e terem seus resultados coletados e transmitidos para o professor, que conta com estrutura para verificar os resultados da turma e até baixar, se desejar, tais dados. Além disso, o trabalho também realizou um levantamento comparativo com vários outros *frameworks* relacionados a jogos na *Web*, trazendo ainda os conceitos básicos de jogos digitais 2D empregados nesse domínio e a documentação geral do sistema.

Ademais, optou-se por um modelo de Software Livre, num esquema sem dependências a algum sistema em particular e nem de outras bibliotecas, incentivando assim que mais pessoas possam usar a implementação e fazer contribuições e melhorias em comunidade, visando com isso uma difusão da tecnologia e do ideal proposto em prol de soluções educativas.

Para trabalhos futuros, pode-se expandir o repositório do sistema desenvol-

vido com vários outros *templates* de jogos e verificar a possibilidade de interação com outras ferramentas populares utilizadas no ensino remoto. Uma maior adoção do SVG em pontos diversos do sistema também pode ser interessante, bem como explorar mais determinados aspectos relacionados a dispositivos móveis e maiores possibilidades de interação entre os alunos nos jogos.

REFERÊNCIAS

ANGEL, E.; SHREINER, D. *Interactive Computer Graphics with WebGL*. [S.l.]: Addison-Wesley Professional, 2014.

BASTEN, D. Gamification. *IEEE Software*, IEEE Computer Society, Los Alamitos, CA, USA, v. 34, n. 05, p. 76–81, sep 2017. ISSN 1937-4194.

BEKOFF, M.; DIMOTTA, M. J. *Animals at Play: Rules of the Game*. [S.l.]: Temple University Press, 2008.

BEN-KIKI, O.; EVANS, C.; INGERSON, B. Yaml ain't markup language (yaml™) version 1.1. *Working Draft 2008-05*, v. 11, 2009.

BIERMAN, G.; ABADI, M.; TORGERSEN, M. Understanding typescript. In: SPRINGER. *European Conference on Object-Oriented Programming*. [S.l.], 2014. p. 257–281.

BRAY, T. et al. *Extensible markup language (XML) 1.0*. [S.l.]: W3C recommendation October, 2000.

CHARLAND, A.; LEROUX, B. Mobile application development: web vs. native. *Communications of the ACM*, ACM New York, NY, USA, v. 54, n. 5, p. 49–53, 2011.

DETERDING, S. et al. From game design elements to gamefulness: defining"gamification". In: *Proceedings of the 15th international academic MindTrek conference: Envisioning future media environments*. [S.l.: s.n.], 2011. p. 9–15.

DJAOUTI, D.; ALVAREZ, J.; JESSEL, J.-P. Classifying serious games: the g/p/s model. In: *Handbook of research on improving learning and motivation through educational games: Multidisciplinary approaches*. [S.l.]: IGI Global, 2011. p. 118–136.

FAAS, T. *An Introduction to HTML5 Game Development with Phaser. js*. [S.l.]: AK Peters/CRC Press, 2017.

FERRAIOLO, J.; JUN, F.; JACKSON, D. *Scalable vector graphics (SVG) 1.0 specification*. [S.l.]: iuniverse Bloomington, 2000.

FLANAGAN, D. *JavaScript: the definitive guide*. [S.l.]: "O'Reilly Media, Inc.", 2006.

FULTON, S.; FULTON, J. *HTML5 canvas: native interactivity and animation for the web*. [S.l.]: O'Reilly Media, Inc., 2013.

GIRARD, C.; ECALLE, J.; MAGNAN, A. Serious games as new educational tools: How effective are they? a meta-analysis of recent studies. *Journal of Computer Assisted Learning*, v. 29, 06 2013.

HAAS, A. et al. Bringing the web up to speed with webassembly. In: *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*. [S.l.: s.n.], 2017. p. 185–200.

HEARN, D.; BAKER, M. P.; CARITHERS, W. R. *Computer graphics with OpenGL*. [S.l.]: Upper Saddle River, NJ: Pearson Prentice Hall„ 2014.

LIU, E. Z. F.; CHEN, P.-K. The effect of game-based learning on students' learning performance in science learning–a case of "conveyance go". *Procedia-Social and Behavioral Sciences*, Elsevier, v. 103, p. 1044–1051, 2013.

MANDERSCHEID, B. *Beginning HTML5 games with CreateJS*. [S.l.]: Apress, 2014.

MEYER, E. *CSS: The Definitive Guide: The Definitive Guide, 4th Ed.* [S.l.]: O'Reilly Media, Inc., 2017.

MILGROM, P.; ROBERTS, J. Rationalizability, learning, and equilibrium in games with strategic complementarities. *Econometrica: Journal of the Econometric Society*, JSTOR, p. 1255–1277, 1990.

MUSCIANO, C.; KENNEDY, B. et al. *HTML, the definitive Guide*. [S.l.]: O'Reilly & Associates, 1996.

NETO, J. C.; BLANCO, M. B.; SILVA, J. Aléssio da. O uso de gamificação e dificuldades matemáticas: possíveis aproximações. UFRGS, 2017.

NICKOLLS, J.; DALLY, W. J. The gpu computing era. *IEEE micro*, IEEE, v. 30, n. 2, p. 56–69, 2010.

NIETO-ESCAMEZ, F. A.; ROLDÁN-TAPIA, M. D. Gamification as online teaching strategy during covid-19: a mini-review. *Frontiers in Psychology*, Frontiers Media SA, v. 12, 2021.

OWENS, J. D. et al. Gpu computing. *Proceedings of the IEEE*, IEEE, v. 96, n. 5, p. 879–899, 2008.

PAIVA, A. C. et al. ilearntest–framework for educational games. *Procedia-Social and Behavioral Sciences*, Elsevier, v. 228, p. 443–448, 2016.

PERNELLE, P. et al. Fast gamification approach: Increase of the motivation in remote classes. In: *CSEDU (2)*. [S.l.: s.n.], 2021. p. 282–287.

RICHARDS, B. *Kiwi.js 1.4.0*. 2015. `https://github.com/gamelab/kiwi.js/commits/master`. [Online; accessed 26-09-2019].

SHAPPIR, D. *Performing binary composition of images onto an html canvas element*. [S.l.]: Google Patents, 2012. US Patent App. 13/414,735.

SPUY, R. Van der. *Learn Pixi. js*. [S.l.]: Apress, 2015.

STALLMAN, R. M. What is free software. *Free Society: Selected Essays of*, v. 23, 2002.

SU, C.-H. The effects of students' motivation, cognitive load and learning anxiety in gamification software engineering education: a structural equation modeling study. *Multimedia Tools and Applications*, Springer, v. 75, n. 16, p. 10013–10036, 2016.

ZICHERMANN, G.; CUNNINGHAM, C. *Gamification by Design: Implementing Game Mechanics in Web and Mobile Apps*. 1th. ed. [S.l.]: O'Reilly Media, 2011.