UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS DE RUSSAS

CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

CARLOS EMANUEL CORDEIRO ALVES

THE TURING TRIALS: PROPOSTA DE ELABORAÇÃO DE UM JOGO EDUCACIONAL PARA MELHORAR O APRENDIZADO EM MÁQUINAS ABSTRATAS E TEORIA DA COMPUTAÇÃO

RUSSAS

2021

CARLOS EMANUEL CORDEIRO ALVES

THE TURING TRIALS: PROPOSTA DE ELABORAÇÃO DE UM JOGO EDUCACIONAL PARA MELHORAR O APRENDIZADO EM MÁQUINAS ABSTRATAS E TEORIA DA COMPUTAÇÃO

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do Campus de Russas da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Ciência da Computação.

Orientador: Prof. Dr. Bonfim Amaro Júnior

RUSSAS

2021

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

C818t Cordeiro Alves, Carlos Emanuel.
The Turing Trials : proposta de elaboração de um jogo educacional para melhorar o aprendizado em máquinas abstratas e teoria da computação / Carlos Emanuel Cordeiro Alves. – 2021.
45 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Russas, Curso de Ciência da Computação, Russas, 2021.
Orientação: Prof. Dr. Bonfim Amaro Júnior.

1. Gamificação. 2. Linguagens Formais. 3. Autômatos. 4. Teoria da Computação. 5. Jogo Educacional. I. Título.

CDD 005

CARLOS EMANUEL CORDEIRO ALVES

THE TURING TRIALS: PROPOSTA DE ELABORAÇÃO DE UM JOGO EDUCACIONAL PARA MELHORAR O APRENDIZADO EM MÁQUINAS ABSTRATAS E TEORIA DA COMPUTAÇÃO

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do Campus de Russas da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Ciência da Computação.

Aprovada em: ___/___/_____

BANCA EXAMINADORA

Prof. Dr. Bonfim Amaro Júnior (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Márcio Costa Santos
Universidade Federal do Ceará (UFC)

Profa. Ms. Eurinardo Rodrigues Costa
Universidade Federal do Ceará (UFC)

À minha Mãe, por acreditar e investir em mim e por me inspirar à ser uma pessoa melhor a cada dia, me motivando e me instigando nos meus melhores e piores momentos.

AGRADECIMENTOS

Ao meu orientador, Prof. Dr. Bonfim Amaro Júnior, o qual eu sou grato por sua dedicação e pelo enorme aprendizado proporcionado. Seus conselhos e recomendações sempre me ajudaram e me deram foco em momentos de dificuldade.

Aos professores Eurinardo Rodrigues e Márcio Costa por terem aceitado o convite para participar da banca deste trabalho e por suas contribuições quando ainda era um projeto de pesquisa.

À todos os meus familiares, em especial à minha mãe, que cuidaram de mim quando eu mais precisei e me estimularam a persistir nos meus piores momentos.

À todos os professores que me apoiaram e me orientaram ao longo da minha jornada acadêmica, contribuindo para o meu conhecimento e aptidão profissional.

Às amizades que a UFC me proporcionou, incluindo, mas não se limitando a, Vinícius Gonçalves, Abraão Victor, Gabriela Leal, Gabriel Trentini, Ana Iza, George Masullo, Hellen Maria, entre muitos outros.

“Não sabendo que era impossível, ele foi lá e fez.”

(Jean Cocteau)

RESUMO

A gamificação é um tema que se dissemina cada vez mais dentro dos âmbitos profissional e educacional. Conforme o tempo avança, empresas, escolas e universidades ao redor do mundo aplicam a gamificação para otimizar o aprendizado, melhorar o conhecimento sobre uma área específica e fornecer práticas necessárias em ambientes livres de riscos. Porém, o conceito ainda é muito recente e existem muitas áreas de atuação que ainda não se beneficiaram de sua aplicação, incluindo as áreas de linguagens formais, autômatos e teoria da computação. Esse projeto busca apresentar uma aplicação da gamificação dentro do contexto de linguagens formais, autômatos e teoria da computação, através de um jogo educacional que estimula a prática do conteúdo coberto por essas áreas. Para a criação de tais aplicações, foi feito um estudo de caso, que busca avaliar a situação atual do desenvolvimento dentro da tecnologia, seus principais méritos e falhas. Em seguida, esse estudo é contextualizado dentro do ambiente supracitado, sugerindo um projeto de gamificação dentro da área de máquinas de estado finitas, inicialmente. Por fim, os resultados da contextualização são utilizados em um projeto de software, com o intuito de desenvolver um módulo prático que possa oferecer uma ferramenta auxiliar ao ensino das disciplinas em questão e o engajamento característico de um programa de computador gamificado.

Palavras-chave: Gamificação; Linguagens Formais; Autômatos; Teoria da Computação; Jogo Educacional.

ABSTRACT

Gamification is a topic that spreads more and more inside the professional and educational scopes. As time goes by, companies, schools, and universities worldwide implement gamification to optimize learning, improve knowledge about a specific field, and provide necessary practise in risk-free environments. However, the concept is still very recent and there are many areas of expertise that still don't benefit from its application, including formal languages, automata theory and theory of computation areas. This project seeks to present a gamification application inside the context of formal languages, automata theory and theory of computation through an educational game that stimulates the practice of the content covered by these areas. To create that application, a case study was made, which seeks to evaluate the actual situation of this technology's development, its main merits and flaws. After this, the study is contextualized inside the environment mentioned above, sugesting a gamification project inside the finite state machine's scope. Finally, the contextualization results are used in a software project to develop a practical module that can offer a support tool for the learning of the mentioned subjects and the distinctive engagement of a gamified computer program.

Keywords: Gamification; Formal Languages; Automata Theory; Theory of Computation; Educational Game.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

ACM	*Association for Computer Machinery*
AFD	Autômato Finito Determinístico
AFND	Autômato Finito Não Determinístico
ICPC	*International Collegiate Programming Contest*
SBC	Sociedade Brasileira de Computação
URI Erechim	Universidade Regional Integrada do Alto Uruguai e das Missões

SUMÁRIO

1 INTRODUÇÃO

O mercado de jogos eletrônicos tem se desenvolvido em grande velocidade no decorrer do século XXI (KIRRIEMUIR; MCFARLANE, 2004), bem como seus consumidores. Pesquisas recentes mostram que, até alcançar os vinte e um anos, jogadores tendem a passar cerca de dez mil horas consumindo jogos eletrônicos (MCGONIGAL, 2011). Estas dez mil horas os tornam extraordinários em atividades estimuladas por esses jogos: criatividade, cooperação e coordenação. Porém, mesmo com tamanhas qualidades, o tempo dedicado a jogos eletrônicos é fortemente questionado por pais e professores, que acreditam que tamanha dedicação e engajamento seriam mais benéficas se direcionadas aos estudos, por exemplo (KIRRIEMUIR; MCFARLANE, 2004).

Para solucionar esse problema, as metodologias ativas de ensino incorporaram a gamificação, método que aplica elementos presentes nos jogos dentro do cotiano, no intuito de tornar o ensino em sala de aula mais divertido e dinâmico (NAVARRO, 2013). Cientistas e educadores desenvolvem e aplicam jogos educacionais para gerar engajamento e foco dentro do aprendizado. Devido ao sucesso da gamificação na metodologia ativa, empresas, desenvolvedores e *game designers* viram-na como um nicho de mercado e trouxeram sua expertise ao ensino, popularizando-o ainda mais. Além disso, conseguiram aplicar vantagens que os educadores e cientistas não possuíam: maior capacitação, maior tempo de desenvolvimento e maiores recursos (FOLMER, 2007).

Essas técnicas de gamificação foram constantemente construídas e adaptadas, o que permitiu que elas se espalhassem e atingissem cada vez mais áreas de educação e especialização, eventualmente chegando à área de ciência da computação. Projetos de gamificação, como o que foi realizado por Maekawa *et al.* (2015) começaram a se tornar mais frequentes nesse curso, e, da mesma forma, começaram a atrair empresas que buscavam encontrar pessoas qualificadas para trabalhos relacionados, como desenvolvedores. Isso, inclusive, criou nichos de mercado para certas empresas. Um exemplo disso pode ser visto na empresa Qualified, que busca conectar outras empresas à desenvolvedores *freelancers*. Esses desenvolvedores usam uma plataforma gamificada chamada CodeWars, criada pela Qualified para serem treinados e avaliados. Essa plataforma pode ser utilizada mesmo dentro do ambiente universitário, conectando-o ainda mais com o ambiente profissional.

Porém, devido a alguns problemas, as tecnologias de gamificação tiveram dificuldades para serem aplicadas em disciplinas mais teóricas. Uma dessas dificuldades é o fato

de que essas disciplinas estão mais ligadas à vida acadêmica e a carreira universitária, o que, apesar de ser benéfico para a comunidade como um todo, não atende a demanda do mercado de trabalho. Por não oferecer os benefícios que as empresas buscam, essas áreas são vistas com menor prioridade pelos projetos de gamificação profissionais.

Outro problema notável é o próprio teor dessas disciplinas. Por serem disciplinas mais focadas em teoria, elas precisam abordar conceitos, terminologias e metodologias. Isso pode ser feito sem muita dificuldade através do ensino e da leitura, porém, ainda existe uma parte prática a ser abordada, e a falta de metodologias ativas nesses segmentos mostra uma carência a ser preenchida.

Algumas dessas áreas são linguagens formais, autômatos e teoria da computação. Devido ao fato de possuírem uma relevância preterida dentro do cenário profissional, não há muitos investimentos em projetos de gamificação dentro do âmbito, o que leva a um prejuízo no potencial de aprendizado desses setores. A maior parte das ferramentas de apoio relacionadas à máquinas abstratas são pouco intuitivas e ergonômicas, além de não possuírem elementos que estimulem seu uso, o que é amplamente presente em ferramentas gamificadas.

Através da necessidade de ferramentas complementares dentro do escopo, bem como a insatisfação de alguns alunos nessas disciplinas devido à falta de metodologias de ensino alternativas, nasce uma oportunidade de explorar a gamificação desse ramo e criar uma ferramenta computacional que facilite o aprendizado e estimule o engajamento nas aulas.

Com base no cenário e problemática acima mencionados, o objetivo deste trabalho é introduzir e descrever o projeto de desenvolvimento de um jogo educacional fundamentado nos conceitos de linguagens formais, autômatos finitos e teoria da computação, com o propósito de oferecer uma ferramenta de auxílio ao ensino dessas disciplinas para professores, um software de aprendizado complementar para alunos e uma plataforma de prática para entusiastas. Com base nisso, a essência desse projeto é descrever e contextualizar o desenvolvimento de um jogo computacional, bem como as ideias que o embasam.

Nesse contexto, a seção 2 apresenta a fundamentação teórica que embasou o desenvolvimento do projeto. A seção 3 expõe os trabalhos relacionados que oferecem um estado da arte que contextualiza o projeto. Já a seção 4 descreve a idealização e implementação do jogo educacional e, finalmente, a seção 5 compila as conclusões e ponderações das possibilidades de pesquisas e implementações futuras.

2 FUNDAMENTAÇÃO TEÓRICA

Esta seção busca introduzir e apresentar os conceitos necessários para o desenvolvimento adequado deste trabalho. Primeiramente, a seção 2.1 introduz o conceito de gamificação e como ele foi trabalhado ao longo dos anos. Em seguida, a seção 2.2 exibe softwares e jogos educacionais ativos em mercado que fundamentam o software desenvolvido nesta obra. Posteriormente, a seção 2.3 disserta brevemente sobre a teoria da computação e seus conceitos básicos, necessários para o entendimento da relevância do projeto e para um melhor embasamento de autômatos finitos, tema de discussão na seção 2.4.

2.1 Gamificação

A gamificação consiste na aplicação de elementos, mecanismos, dinâmicas e técnicas frequentemente utilizadas em jogos. Várias esferas da sociedade pós-moderna incorporam características lúdicas em seu cotidiano. Por exemplo, simuladores, como o que pode ser visto na figura 1, são usados por profissionais buscando aprender a utilização de ferramentas e veículos complexos e pesados. Além disso, existem aplicativos que organizam objetivos e metas para a realização de pesquisas e atividades realizadas por usuários em redes sociais que pontuam seus perfis com base na popularidade de uma publicação (NAVARRO, 2013).

A palavra gamificação origina-se do termo em inglês *gamification*, criado pelo programador britânico Nick Pelling. Porém, apesar de sua criação ter ocorrido apenas em 2003, há registros de sua aplicação em contextos profissionais datados do início do século XX, principalmente pela similaridade dos jogos com o comércio (NAVARRO, 2013). O mundo corporativo possui elementos que estimulam a solução de problemas, a cooperatividade e a competitividade. De forma semelhante, em uma experiência educacional gamificada, os estudantes cooperam entre si em equipes que competem umas contra as outras em busca de um objetivo (ATTLE; BAKER, 2007).

Devido a grande quantidade de expressões idiomáticas que empregam o termo "jogo"(do latim, *jocus*), tendo por exemplos "abrir o jogo", "jogo da vida", "jogo para dois", entre muitos outros, esta palavra adquiriu uma grande abrangência, tornando-se referência para uma amplitude de situações e atividades e tornando quase impossível dar um significado objetivo à palavra (Navarro, 2013). O termo pode ser aplicado em qualquer situação do cotidiano em que se busque alcançar algum objetivo, mas, nesta pesquisa, gamificação restringe-se apenas a

Figura 1 – Simulador Cat® para operação de maquinário pesado

Fonte: https://catsimulators.com/wp-content/uploads/2019/01/AD-4SCREEN.jpg

aplicação de elementos de jogos no âmbito educacional.

A análise e implementação da gamificação em ambientes de ensino fundamental mostrou que ela é capaz de gerar maior engajamento dos alunos nas disciplinas e tornar a experiência de aprendizado mais significativa (CUNHA *et al.*, 2017). Isso inspirou cientistas, desenvolvedores e organizações a evoluir a gamificação e adaptá-la ao ambiente do ensino superior. Dentro do escopo de Ciência da Computação, o sistema gamificado mais conhecido mundialmente é o *International Collegiate Programming Contest* (ICPC). Em 1970, a Universidade do Texas deu início a uma competição de programação que, alguns anos depois, tornou-se a ICPC, organizada pela *Association for Computer Machinery* (ACM), promovida pela IBM e sediada na Universidade de Baylor. Inicialmente frequentada apenas por times americanos e canadenses, a competição passou a ocorrer em etapas qualificatórias regionais e nacionais a partir de 1977. Em 2014, foi registrada a participação de mais de 101 países (PIEKARSKI *et al.*, 2015).

Há muitas características presentes dentro do ambiente das maratonas de programação que tornam o fluxo de aprendizado superior ao ensino tradicional. O engajamento provocado pela competitividade, somado com o desafio proposto pela solução de problemas, leva os participantes a pensarem em estratégias diferentes e consultas teóricas, o que gera conhecimento aos participantes, tanto de forma individual quanto de forma coletiva. Além disso, a classificação fi-

nal não afeta diretamente a experiência adquirida. Logo, uma equipe que não conseguiu alcançar as colocações superiores não é prejudicada em seu desempenho acadêmico.

Outro fator que também afeta positivamente a educação dos participantes é a forma como os problemas de uma maratona são elaborados. Os problemas são apresentados dentro de um contexto, o que estimula o estudo, por permitir que os usuários visualizem como o conhecimento adquirido se aplica diariamente, bem como o pensamento analítico, pois os competidores precisam abstrair o problema e extrair as informações relevantes para a sua solução. A competitividade também promove experiências mais amplas, porém essenciais ao mercado de trabalho, como a cooperação, a criatividade e a aptidão estratégica.

Segundo Zanini e Raabe (2012), nas disciplinas iniciais de 51 cursos de Ciência da Computação espalhados pelo Brasil, a maioria dos enunciados das questões nas bibliografias não estão inseridos em um contexto (cerca de 65% dos enunciados) e, dos enunciados que estão em um contexto, 55% estão em um contexto matemático. Em oposição, os problemas apresentados nas maratonas de programação são bem contextualizados e de acordo com a metodologia de solução de problemas.

Burguillo (2010) observou que a abordagem competitiva no aprendizado incluía vantagens, como:

- A independência entre o ensino e a classificação dos participantes da competição.
- Internamente, os grupos precisam colaborar para melhorar suas chances de vitória.
- Tal abordagem pode ser complementada com outros métodos de aprendizado.

De um modo geral, as atividades de gamificação despertam nos alunos o interesse por problemas de competições, ampliam o conhecimento dos participantes e estimulam a resolução de problemas e o trabalho em equipe. Nesse contexto, sugiram algumas ferramentas computacionais que dão suporte ao ensino de diversos conteúdos, inclusive, a programação de computadores.

2.2 Softwares de apoio ao ensino de programação de computadores

Após a disseminação do ICPC e a criação da maratona de programação da Sociedade Brasileira de Computação (SBC) era visível a necessidade da criação de uma plataforma complementar, que permitiria o acompanhamento em tempo real de todos os competidores. Essa necessidade se consolidou na criação do BOCA, software utilizado oficialmente nas competições da SBC. O sistema permite que os usuários pré-cadastrados submetam soluções para problemas propostos, que são avaliadas de forma automática. A aplicação também permite um acompa-

nhamento em tempo real da classificação das equipes (PIEKARSKI *et al.*, 2015). Ao submeter uma possível solução para o problema proposto na maratona, a equipe recebe uma resposta do sistema (CAMPOS; FERREIRA, 2004). No contexto da maratona de programação da SBC, se a solução está correta, a equipe é notificada e recebe um balão. O propósito do balão é criar um engajamento visível para todas as equipes da competição e descontrair o ambiente (PIEKARSKI *et al.*, 2015).

Dito isso, após a popularização da gamificação, educadores encontraram uma forma de integrá-la ao ensino de programação de computadores, área bastante disseminada no curso de Ciência da Computação. Tal integração mostrou um cenário bastante promissor, mas também mostrou falhas no processo. A falha mais evidente residia no fato de que cientistas e educadores, apesar de possuírem capacidade, não possuíam tempo, recursos ou até mesmo expertise para desenvolver jogos educacionais (SAVI; ULBRICHT, 2008).

Com base nesse contexto, tendo em vista as possibilidades e os desafios desse novo conceito, desenvolvedores profissionais e *game designers* transformaram a gamificação na programação em um nicho de mercado. Através da colaboração entre as áreas profissional e didática, os jogos educacionais foram se tornando cada vez mais abrangentes em conhecimento e adquirindo maior atratividade.

Dois exemplos bastante populares de softwares para a educação em programação são o URI Online Judge e o Codewars. Ambos seguem conceitos similares: o participante recebe tarefas, contendo objetivos a serem alcançados, entradas definidas e saídas esperadas. Levando isso em consideração, o jogador deve criar um algoritmo usando a linguagem de sua escolha que receba a entrada e gere a saída esperadas, com base no objetivo estipulado. Resoluções corretas nestas tarefas tendem a desbloquear tarefas mais complexas, oferecendo um crescimento de dificuldade linear e um desafio motivador ao usuário.

Com o avanço dos softwares atuais e a criação de novos, duas vertentes de software de apoio para programação foram criadas. A primeira, criada por educadores e voltada para alunos universitários, possui maior objetividade, prejudicando sua atratividade em prol de uma capacidade educacional maior. A segunda vertente, criada por *game designers* e voltada para o público em geral, possui uma maior atratividade e engajamento, mas devido ao seu maior público-alvo, ela é prejudicada ao abordar um ensino mais avançado. Tanto educadores quanto *game designers* trabalham arduamente para encontrar um ponto de equilíbrio entre essas duas vertentes, buscando desenvolver um software que seja simultaneamente atraente e instrutivo.

Porém, mesmo com um avanço tão significativo e com o apoio de vários especialistas distintos, a tendência dos jogos educacionais ainda é guiada pela economia de mercado, e nesse sentido, os jogos educacionais relacionados à ciência da computação se afunilam em poucas de suas diversas especializações, quase todas centradas na programação prática para o desenvolvimento de aplicativos e sistemas, a serem usados de forma comercial.

Apesar de ser uma forma relevante e útil de trazer e integrar o conhecimento universitário ao ambiente de mercado, o que é, em si, uma das diversas aplicações da universidade, afeta negativamente o interesse e dedicação em outras vertentes e especializações da área. A falta de jogos educacionais que cobrem outras áreas da computação, especialmente disciplinas mais teóricas e relevantes à carreira acadêmica, obscurece a descoberta e o interesse daqueles que desconhecem tais disciplinas, dificulta autodidatas a melhorarem suas habilidades nessas áreas e torna a experiência daqueles que precisam aprender sobre elas mais difícil, o que na prática leva ao aumento de reprovações, a insatisfação de alguns alunos em relação a essas disciplinas, e possivelmente, a evasão dos cursos de ciência da computação.

Como forma de apresentar uma alternativa para resolver tal problema, esse projeto busca propor, planejar e iniciar o desenvolvimento de uma aplicação gamificada que dinamize o aprendizado e prática do conteúdo de disciplinas relacionadas a linguagens formais, autômatos e teoria da computação. Mas para que esse projeto possa cumprir sua função de instruir, ele precisa se conectar adequadamente aos conceitos e teorias que o embasam.

2.3 Teoria da computação

Em seu conceito mais básico, segundo Diverio e Menezes (2009), a ciência da computação reúne todo o conhecimento sistemático relacionado à computação. Tendo origem no desenho de algoritmos por Euclides, no século III a.C., e nos estudos feitos na Babilônia sobre complexidade e redutibilidade de problemas, existem duas ênfases de interesse: a ênfase teórica, de ideias fundamentais e modelos computacionais, e a ênfase prática, de projeto de sistemas computacionais.

Conforme a ciência da computação se expandiu e envolveu diversas áreas, como biologia, eletrônica, matemática e linguística, esses estudos criaram o que viria a ser a base da teoria da computação. Conforme os conceitos da teoria da computação foram sendo definidos e expandidos, várias pesquisas foram desenvolvidas a partir do século XX com o objetivo de encontrar um modelo computacional que fosse capaz de implementar qualquer função

computável. Entre os diversos formalismos existentes para cumprir esta função, estão a Máquina de Turing (1936), a Máquina Norma (1976), o Sistema Canônico de Post (1943), os Algoritmos de Markov (1954), a Linguagem Snobol (1954), a Máquina de Registradores (1963), o RASP (1964), entre vários outros (DIVERIO; MENEZES, 2009).

Para que a ideia de teoria da computação seja melhor transmitida, é preciso se entender o que são programas, máquinas e computações. Esses elementos serão apresentados dentro do escopo teórico que é apresentado nesse documento.

Segundo Diverio e Menezes (2009), um programa é um conjunto de instruções que tornam uma máquina capaz de aplicar uma sucessão de operações e testes, transformando dados inicialmente fornecidos em dados desejáveis. O primeiro programa foi publicado em 1843 por Augusta Ada Byron, Condessa de Lovelace, como uma forma de calcular os números de Bernoulli através da máquina analítica de Charles Babbage (MENABREA; LOVELACE, 1842). Desde então, os programas tem se desenvolvido com o objetivo de estabelecer sua capacidade, definir e testar seus limites e se tornar suficientemente simples para serem criados e usados em diversas áreas da sociedade. Em relação aos programas atuais, existem dois aspectos independentes que os categorizam, sendo eles a sua estruturação e a composição de suas instruções. Segundo Diverio e Menezes (2009), existem diversos tipos de estruturação, mas entre eles, três se destacam:

- Monolítica, sendo baseada em desvios condicionais e incondicionais, não possuindo mecanismos de iteração, subdivisão ou recursão.
- Iterativa, possuindo mecanismos para controlar trechos de programas e iterações. Esse tipo não permite desvios incondicionais.
- Recursiva, com mecanismos de estruturação em sub-rotinas recursivas, permitindo que o programa possa ser definido de forma indutiva. Assim como a estruturação iterativa, esta também não permite desvios incondicionais.

Além e independentemente disso, um programa podem ter diversos tipos de composição entre suas operações, sendo eles:

- Sequencial, onde a operação seguinte começa apenas após o fim da anterior.
- Não-determinista, onde uma operação composta pode ou não ser executada.
- Concorrente, onde as operações podem ser feitas em qualquer ordem, ou mesmo simultaneamente.

Entre essas composições, a não-determinista explora os limites da capacidade prática de solucionar problemas (DIVERIO; MENEZES, 2009).

Essas definições basearam a computação e levaram ela a níveis muito avançados. Dispositivos foram criados para permitir a construção e solução desses problemas, especialmente os problemas que uma pessoa ou mesmo um grupo de pessoas levaria anos ou até décadas para solucionar. Essas máquinas, os computadores, progrediram ao longo dos anos para se tornar cada vez menores e mais poderosos, ao ponto de que a própria ideia de um computador se tornou corriqueira na sociedade, embora a definição de computação esteja longe de ser tão trivial.

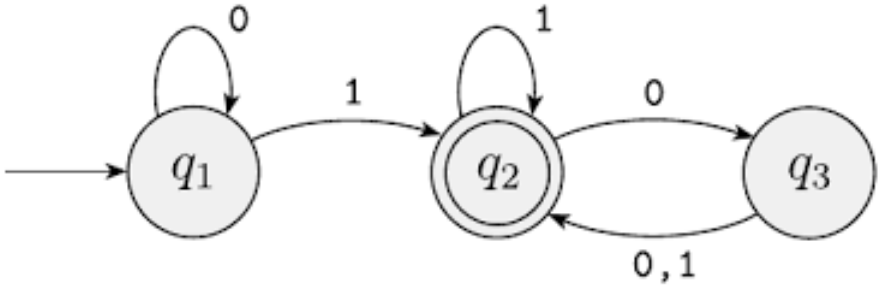
Como uma forma de simplificar a explicação do que é um computador Sipser (2007) idealizou um "modelo computacional", que funciona como a ideia de um computador. Como essa ideia é precisa de algumas formas, mas não em outras, vários modelos computacionais diferentes foram pensados, cada um focado em um conjunto diferente de características. Esse projeto foi feito levando em consideração vários desses modelos, mas, para fins de facilidade de compreensão, este artigo cobrirá o modelo mais simples de todos e o único até então que foi totalmente implementado no jogo educacional: o autômato finito.

2.4 Autômatos finitos

Em uma definição comum, elaborada por Sipser (2007), um autômato finito, também conhecido como máquina de estados finita, é um modelo computacional com uma memória limitada. Esse tipo de modelo é muito utilizado quando buscamos reconhecer padrões em dados e ele é aplicado em muitos tipos diferentes de dispositivos, de ventiladores à portas automáticas.

Em uma definição mais teórica, um autômato finito pode ser representado na forma de um diagrama de estados, ilustrada na figura 2. Esse diagrama representa um autômato finito com três estados. q_1 é o estado inicial, representado pela seta que surge do nada e aponta para ele. O estado de aceitação, q_2, é representado por um duplo círculo. Já as setas que saem de um estado e vão para o outro, são chamadas de transições (SIPSER, 2007).

Figura 2 – Ilustração de um autômato finito



Fonte: https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcQJ0ZbxYy2kEwFd6K8J5YX-FWKQsAAVO02hAgusqp=CAU

O propósito de um autômato é receber uma cadeia de entrada, processá-la e gerar uma saída. O processamento inicia no estado inicial, move-se pelas transições de um estado para o outro após ler cada símbolo, conforme lê os elementos da cadeia de entrada, um por um, da esquerda para a direita. Após ler o último elemento da cadeia de entrada, o autômato gera sua saída, aceitando se ele estiver em um estado de aceitação nesse momento e rejeitando caso contrário.

Sendo assim, segundo Sipser (2007), um autômato finito é, em sua definição formal, uma 5-upla (uma tupla de cinco elementos) representada por $(Q, \Sigma, \delta, q_0, F)$, onde:

- "Q"é um conjunto finito e não vazio que representa os estados do autômato.
- "Σ"é um conjunto finito que representa o alfabeto do autômato.
- "δ"é uma função de transição do tipo $Q \times \Sigma \rightarrow Q$.
- "q_0"é o estado inicial do autômato pertencente a "Q".
- "F"é o conjunto dos estados de aceitação do autômato, sendo que $F \subseteq Q$.

Como já foi citado anteriormente, um autômato tem como funcionalidade receber uma cadeia e gerar um resultado, este basicamente sendo "aceitação"ou "rejeição". Com isso em mente, pode deduzir que, para um autômato finito M, existe um conjunto que contem todas as cadeias que M aceita, e somente elas. Esse conjunto, chamado $L(M)$, é a linguagem de M.

Dentro do conceito de modelos computacionais, abordado na seção anterior, existem muitos teoremas e teses que regem seu funcionamento. Para a construção e funcionamento adequado do jogo educacional, alguns desses foram levados em maior consideração. Especialmente a Tese de Church-Turing, fundamental para o conceito de decidibilidade. Segundo Sipser (2007), a tese de Church-Turing, nomeada em referência a Alonzo Church e Alan Turing, assume que um algoritmo deve satisfazer a quatro requisitos:

1. O algoritmo consiste em um conjunto finito de instruções simples e descritas com um número finito de símbolos.
2. O algoritmo sempre produz um resultado em um número finito de passos.
3. Por princípio, um humano pode executar esse algoritmo usando apenas papel e lápis.
4. A execução desse algoritmo não requer inteligência do ser humano, além do necessário para entender e executar as instruções.

Essa tese estabeleceu a fronteira do que pode ser considerado decidível por qualquer algoritmo. Isso se provou essencial em nível prático para esse projeto, pois a corretude dos autômatos finitos apresentados pelos alunos é decidida através do teorema de equivalência entre

autômatos finitos. Esse teorema só se tornou possível graças ao fato de que foi provado que um algoritmo que recebe dois autômatos finitos e comprara se esses autômatos são equivalentes é decidível, conforme consta o Teorema 2.4.1.

Teorema 2.4.1 *Sendo* $EQ_{AFD} = \{< A,B > \mid A \text{ e } B \text{ são AFD's e } L(A) = L(B)\}$ *uma linguagem formal,* EQ_{AFD} *é decidível.*

Ainda em Sipser (2007), o autor expõe a decidibilidade desse algoritmo por meio de uma série de teoremas que reduzem a complexidade da prova a níveis cada vez menores. Suponha um algoritmo que recebe dois autômatos finitos A e B como entrada. Esse algoritmo deve construir um terceiro autômato finito, C, de forma que $L(C)$ seja a diferença simétrica entre $L(A)$ e $L(B)$. Na prática, o autômato C aceitaria apenas as cadeias que um dos autômatos A ou B aceitaria exclusivamente, rejeitando as cadeias que ambos aceitam ou ambos rejeitam. Ao provar que $L(C) = \emptyset$, pode-se deduzir que qualquer cadeia que A aceitar, B também aceitará, bem como qualquer cadeia que A rejeitar, B também rejeitará. Logo, A e B são equivalentes e reconhecem a mesma linguagem.

Tendo em vista o que foi constatado acima, para provar que a equivalência entre dois autômatos finitos é decidível (Teorema 2.4.1), é preciso provar dois pontos: se a diferença simétrica entre dois autômatos finitos gera um autômato finito, e se o problema de descobrir se uma determinada linguagem regular é nula é decidível (Teorema 2.4.2).

Teorema 2.4.2 *Sendo* $V_{AFD} = \{< A > \mid A \text{ é um AFD e } L(A) = \emptyset\}$ *uma linguagem formal,* V_{AFD} *é decidível.*

Tendo início com o Teorema 2.4.2, é possível desenvolver um algoritmo de marcação que avalia se um autômato finito consegue, através de uma busca em profundidade, sair de seu estado inicial e chegar em um estado de aceitação. O algoritmo, partindo do estado inicial, marca o estado onde ele está e segue por todas as transições possíveis partindo desse estado, repetindo então esse processo em todos esses estados, até que todos eles tentem marcar estados que já foram marcados. Depois disso, verifique as marcações. Se nenhum estado de aceitação foi marcado, então não é possível chegar até eles de nenhuma forma, logo, a linguagem do autômato finito em questão é nula. Se houver pelo menos um estado de aceitação marcado, a linguagem não é nula. Pela Tese de Church-Turing, uma Máquina de Turing é capaz de executar esse algoritmo, logo, ele é decidível, provando assim o Teorema 2.4.2.

Uma vez que isso tenha sido estabelecido, o próximo passo é verificar se a diferença simétrica entre duas linguagens gera uma linguagem. Para isso, devemos estabelecer a diferença simétrica em termos da álgebra booleana. A diferença simétrica entre duas linguagens $L(A)$ e $L(B)$, resultando em $L(C)$, pode ser escrita da seguinte forma:

$$L(C) = (L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$$

Uma vez que a diferença simétrica pode ser estabelecida dessa forma, se pudermos provar que as operações de união, interseção e complemento entre linguagens são decidíveis, então, por dedução, provamos que a operação de diferença simétrica também é decidível.

Para a operação de união entre duas linguagens $L(A)$ e $L(B)$, podemos criar um algoritmo que receba dois autômatos que reconhecem essas linguagens (por convenção, chamaremos esses autômatos de A e B, respectivamente) e construa um autômato C, definido pela tupla $(Q, \Sigma, \delta, q_0, F)$, onde:

- Q é o conjunto de estados de C, contendo os estados de A, os estados de B e um estado adicional q_0.
- Σ é o alfabeto de C, que contém o alfabeto de A e o alfabeto de B.
- δ é a função de transição de C, definida por $Q \times \Sigma \to Q$. De modo mais informal, δ contém as transições de A, as transições de B, e duas transições em vazio adicionais. Essas duas transições partem de q_0 e seguem cada uma para os estados iniciais de A e B.
- q_0 é o estado inicial de C, recém-criado.
- F é o conjunto dos estados de aceitação de C, que contém o conjunto de estados de aceitação de A e o conjunto de estados de aceitação de B.

Tendo ciência da estrutura do autômato finito C, é perfeitamente possível construir um algoritmo que produza C, de forma que $L(C) = L(A) \cup L(B)$, logo, pela Tese de Church-Turing, a operação de união entre duas linguagens é decidível.

Seguindo para a operação de interseção entre duas linguagens $L(A)$ e $L(B)$, nós podemos criar um algoritmo que cria o autômato finito D, a partir dos autômatos finitos A e B, representado pela tupla $(Q, \Sigma, \delta, q_0, F)$, onde:

- Q é o conjunto de estados de D, que contém o conjunto dos estados de A e o de B.
- Σ é o alfabeto de D, que contém o alfabeto de A e de B.
- δ é a função de transição de D, definida por $Q \times \sigma \to Q$. δ contém as transições de A, as de B, e transições em vazio adicionais. Cada uma dessas novas transições parte de um estado de aceitação de A para o estado inicial de B.

- q_0 é o estado inicial de D, que é equivalente ao estado inicial de A.
- F é o conjunto dos estados de aceitação de D, que é equivalente ao conjunto dos estados de aceitação de B.

Da mesma forma que foi possível construir um algoritmo para criar um autômato finito que reconheça a união entre duas linguagens, também é possível construir um algoritmo que produza um autômato que reconheça a interseção entre elas, e, da mesma forma, é possível provar que esse algoritmo é decidível pela Tese de Church-Turing.

Por fim, é preciso provar que a operação de complemento sobre uma linguagem $L(A)$ é decidível. Esta pode ser provada por um método muito mais simples e teórico, tendo como base a mesma Tese de Church-Turing. Uma forma de descrever a tese, segundo o próprio Turing (1937), seria que toda função naturalmente computável pode ser computada por uma Máquina de Turing. Com base nisso, é lógico deduzir que um autômato finito A pode ser computado em uma Máquina de Turing. Em uma descrição sucinta e informal, uma Máquina de Turing, também conhecida como Máquina Universal, funciona de forma semelhante a um autômato finito, porém, ela possui uma memória ilimitada e irrestrita, sendo, desta forma, um modelo computacional mais preciso e mais próximo dos computadores reais. Uma Máquina de Turing pode fazer tudo o que um computador real faz, porém ele não resolve todos os problemas. Esses problemas que estão além das capacidades de uma Máquina de Turing são conhecidos como problemas indecidíveis e estão além dos limites teóricos da computação (SIPSER, 2007).

Nesse caso, suponhamos uma Máquina de Turing M que compute A, mas com uma diferença:

- Onde A aceitar, M rejeita.
- Onde A rejeitar, M aceita.

Por dedução lógica, a linguagem $L(M)$ atua como um complemento de $L(A)$. Como M é uma Máquina de Turing e qualquer problema que possa ser resolvido por uma Máquina de Turing é decidível, o complemento de $L(A)$ é decidível. Uma vez que nós sabemos que as operações de união, interseção e complemento sob linguagens são decidíveis, nós provamos que a diferença simétrica entre duas linguagens também é decidível. Isso, combinado com a prova do teorema 2.4.2, prova o teorema 2.4.1.

Finalmente, toda a fundamentação teórica discutida acima foi considerada como uma maneira de aplicar as correções dentro da ferramenta, comparando o autômato criado pelo professor como uma solução do problema com o autômato criado pelos usuários/alunos.

3 TRABALHOS RELACIONADOS

O estudo dos jogos educacionais é relativamente recente, tendo ganhado destaque apenas no século XXI. Os primeiros trabalhos cujos objetivos eram catalogar os benefícios e desafios do desenvolvimento de um jogo educacional foram efetuados por Savi e Ulbricht (2008), e desde então, com o desenvolvimento dos jogos e plataformas, tem se potencializado. Uma revisão do estado da arte mais recente foi feita por Signori *et al.* (2016), retratando o avanço das plataformas e os principais problemas ao tentar conciliar entretenimento e aprendizado.

Diante das abordagens práticas existentes que basearam o estudo do desenvolvimento da aplicação envolvida, essa seção apresenta alguns dos projetos existentes na área de gamificação em computação e como elementos presentes dentro deles podem ser aproveitados para intensificar a relevância desse software. A subseção 3.1, descreve um software planejado para desenvolver o pensamento computacional em escolas de forma lúdica. A subseção 3.2 cita uma aplicação de aprendizado, treinamento e aperfeiçoamento em linguagens de programação voltada para estudantes universitários.

3.1 Máquina do Curupira

Com a constante evolução tecnológica, organizações e universidades têm trabalhado em projetos que envolvem a área de programação e desenvolvimento de aplicativos. Todavia, infelizmente, as disciplinas relacionadas à interpretação e compreensão de linguagens formais, bem como construção de máquinas abstratas, são deixadas em escassez. Um projeto desenvolvido por Pires *et al.* (2019), chamado "Máquina do Curupira", tem chamado a atenção por lidar com um elemento ensinado na teoria da computação: a Máquina de Turing. Porém, apesar de lidar com algo presente no âmbito universitário de forma lúdica dentro dos ensinos fundamental e médio, bem como envolver questões relevantes como a conscientização ambiental e a preservação cultural através do enredo, este jogo possui um público-alvo diferente do âmbito universitário e um objetivo de ensino diferente da interpretação de linguagens formais e construção de máquinas abstratas, buscando, ao invés disso, estimular o pensamento computacional em crianças e adolescentes.

A principal motivação por trás do projeto de Pires *et al.* (2019) reside na necessidade atual de destacar o processo de resolução de problemas (MCGONIGAL, 2017) (RESNICK; ROBINSON, 2017), e na crescente relevância do termo "Pensamento Computacional", utilizado

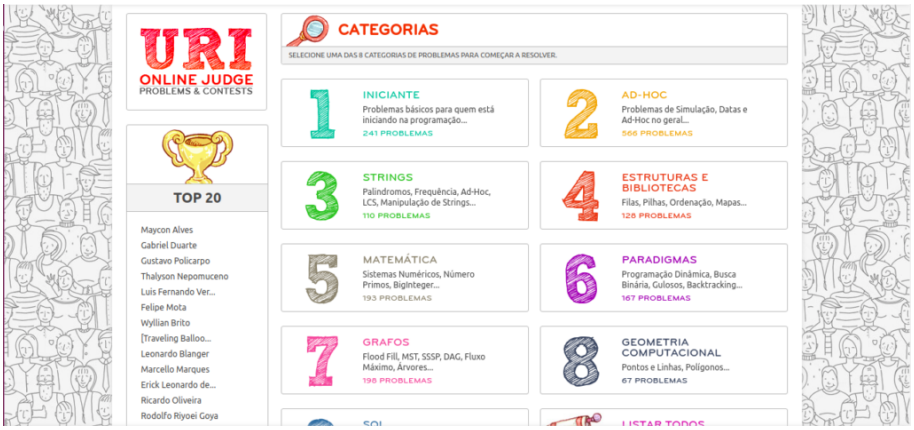
no discurso de Wing (2006). Porém, o trabalho de Pires *et al.* (2019) leva seu foco ao público infantil, considerando estimular o pensamento computacional desde a infância, quando se tem maior estímulo da memória de longo prazo (PIRES *et al.*, 2018). Por isso, o aplicativo desenvolvido simultaneamente ao artigo carrega elementos atrativos para o público-alvo, como desenhos cartunescos, objetos coloridos e efeitos sonoros.

Todavia, isso não impede que o aplicativo seja utilizado como base para a concepção da aplicação envolvida neste trabalho de conclusão de curso. Considerando o fato de que o Enigma do Curupira utiliza a abordagem de máquinas abstratas para construir os problemas a serem resolvidos pelo usuário, o mesmo conceito lógico pode ser adaptado e aplicado no projeto *The Turing Trials*.

3.2 URI Online Judge

O jogo educacional proposto e desenvolvido neste trabalho busca levar as bases de computação e programação de computadores ao âmbito das linguagens formais. Sendo uma adaptação de sistemas lúdicos preexistentes, como a maratona de programação da SBC, em vigor desde 1996, o ICPC, em vigor desde 1977 (PIEKARSKI *et al.*, 2015), e as plataformas online de treinamento de programação, como o URI Online Judge, mostrado na figura 3.

Figura 3 – Captura de tela do URI Online Judge, demonstrando a divisão entre tópicos e dificuldade



Fonte: https://programadorsagaz.com.br/wp-content/uploads/2017/10/uri-judge-1024x475.png

Em relação ao URI Online Judge, esta plataforma online brasileira é um projeto em desenvolvimento pelo Departamento de Ciência da Computação da Universidade Regional Integrada do Alto Uruguai e das Missões (URI Erechim), coordenado por BEZ e TONIN (2014). Ela foi inspirada pelo *UVa Online Judge*, criação da Universidade de Valadolid, pelo *uHunt*,

ferramenta complementar ao *UVa Online Judge*, pelo *SPOJ*, site com foco em competições entre usuários, e pelo *BOCA Online Contest Administrator*, ferramenta utilizada na fase seletiva da Maratona de programação da SBC.

Esta ferramenta foi fundamentalmente pensada com o propósito de simplificar a exaustiva porém constante atividade de formular e corrigir atividades individualmente, enfrentada diariamente por professores. Com essa aplicação, as tarefas são formuladas uma única vez e corrigidas automaticamente e independentemente do professor. Nesse sentido, a plataforma se comporta de forma muito semelhante ao *UVa Online Judge*, porém, o motivo que levou à criação de uma ferramenta independente foi a indisponibilidade da ferramenta durante o período da noite. Devido à diferença de fuso-horário, a ferramenta entrava em estado de manutenção durante o período da noite nas universidades brasileiras. Além disso, o sistema possui problemas apenas em inglês, dificultando seu uso (BEZ; TONIN, 2014).

Outro fator que também atuou como um empecilho foi o fato de que o *UVa Online Judge* atua apenas como um repositório de problemas. Sem separá-los por tópico ou dificuldade, o sistema acaba por desestimular o usuário iniciante (BEZ; TONIN, 2014).

Tendo esses problemas em vista, o *URI Online Judge* buscou criar uma nova plataforma, tendo como principais diferenciais uma melhor usabilidade, funcionamento constante, suporte à língua portuguesa, recursos para o acompanhamento dos professores, categorização por tipo de problema e nível de dificuldade, ambiente de comunicação e troca de experiências, bem como uma forma de permitir aos usuários iniciantes obterem dicas e conselhos dos mais experientes (BEZ; TONIN, 2014).

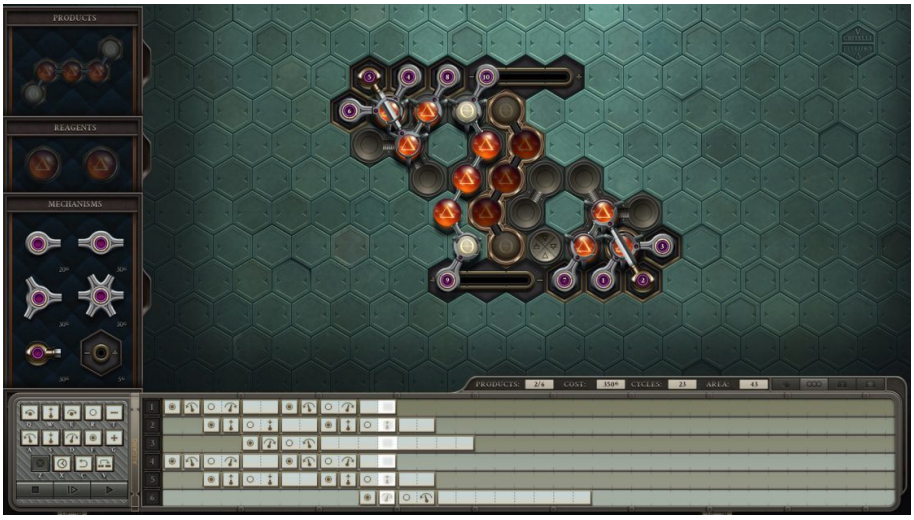
Esta ferramenta possui diversos elementos que gamificam a experiência, como a progressão da dificuldade os conceitos de pontuação e ranking, bem como a competitividade, uma vez que ela oferece suporte à torneios. Porém, o principal diferencial entre o sistema e a aplicação proposta nesse projeto é o conteúdo a ser abordado. O *URI Online Judge* trabalha com programação direta em código-fonte, enquanto o *The Turing Trials* lida com programação visual de máquinas abstratas. Porém, independentemente disso, muitos recursos e funcionalidades podem ser adaptadas e inseridas na base deste *software*.

3.3 Opus Magnum

Após analisar e abstrair as pesquisas com características fundamentais da gamificação observou-se que trazem atributos e características similares às existentes em jogos de quebra-

cabeça (também conhecidos como jogos *puzzle*), onde problemas e situações de dificuldade e complexidade gradual são apresentados ao jogador, que deve usar de raciocínio, lógica e criatividade para resolvê-los. Progredindo, dessa maneira, dos problemas mais simples, que o permitem compreender a lógica do jogo, até os mais desafiadores, que expandem os limites de suas habilidades. Levando isso em consideração, o software vinculado a este projeto deve trazer o engajamento de um jogo com esse mecanismo, para que ele seja atrativo e agradável.

Figura 4 – Captura de tela do jogo Opus Magnum



Fonte: https://media.contentapi.ea.com/content/dam/news/www-ea/common/ea-featured-tile-oa-play-opus-magnum-16x9.jpg.adapt.crop191x100.628p.jpg

Com isso em mente, chegamos ao jogo que agiu como inspiração para o projeto, mostrado na figura 4: Opus Magnum. Esse jogo foi desenvolvido pela Zachtronics, uma empresa de videogames independente com foco em jogos de programação e quebra-cabeças. A empresa foi fundada por Zach Barth, que atua como *designer-chefe*. Atualmente, a Zachtronics é uma divisão da Alliance Media Holdings. Além do Opus Magnum, a desenvolvedora é responsável pela criação dos jogos EXAPUNKS, SHENZHEN I/O, TIS-100, Infinifactory, entre outros. O sucesso promovido por esses jogos gerou uma categoria própria dentro dos jogos de quebra-cabeça: os *Zachlikes* (BARTH, 2020).

O jogo Opus Magnum funciona com a ideia de elementos alquímicos (representados por esferas com cores e inscrições simbólicas diferentes) que são transportados de sua origem por braços mecânicos até o seu destino. Durante o transporte, os elementos são transformados, combinados e separados para se igualarem com o elemento requisitado no destino. Para alcançar esse objetivo, o jogador se dispõe de dois recursos. O primeiro é a montagem do sistema contendo os elementos de entrada, os braços mecânicos, os "transformadores"que permitem

a modificação dos elementos e os espaços para a inserção dos elementos finais. O segundo é a "programação"dos braços mecânicos para manusear os elementos e levá-los para onde são necessários. O jogador faz isso em uma interface de "linha do tempo", onde ele pode inserir comandos visuais que devem ser executados paralelamente e em sincronia para garantir o resultado esperado. Para os jogadores iniciantes, o jogo possui uma campanha onde deve-se resolver problemas progressivamente mais difíceis. Para os jogadores experientes, o jogo possui duas formas de desafio: a primeira é uma série de desafios extremos e complexos, que exigem extremo raciocínio e criatividade do jogador; a segunda é um sistema de pontuação, que permite que cada desafio seja re-jogável, no intuito de que o jogador encontre soluções melhores, menores e mais rápidas para o problema. Apesar da mecânica deste jogo não concordar totalmente com o conceito de máquinas abstratas, ela foi suficientemente semelhante para atuar como base do software planejado neste trabalho.

Entre os jogos da Zachtronics, o Opus Magnum é o jogo de programação mais visual e intuitivo, podendo ser facilmente aprendido por leigos e pessoas que não possuem conhecimento de programação. Pessoas que jogam Opus Magnum, apesar de não lidarem com nenhuma linguagem de programação específica, aprendem a lidar com conceitos presentes em programação, como sequência, repetição e sincronia. É preciso lidar com a noção de dividir um grande problema em sequências de problemas menores, entender o que fazer para resolver o enigma, bem como em que ordem e de que forma. Todo esse processo está diretamente conectado à produção de máquinas abstratas através de um processo imprescindível existente em ambos os elementos: a análise e construção de algoritmos e grafos.

4 THE TURING TRIALS

Para que o software fosse desenvolvido com sucesso e alcançasse uma grande quantidade de usuários, ele foi planejado com base em três requisitos:

- Didática: o software deve instruir aos alunos sobre a interpretação de linguagens formais, bem como estimular o aprendizado de autômatos finitos.
- Ludicidade: o software deve ser estimulante e divertido, gerando atratividade e permanência em seu uso.
- Usabilidade: o software deve ser prático e de fácil entendimento, para que o usuário não necessite de documentação externa para usá-lo.

Dadas as exigências anteriores, o sistema de regras da maratona de programação foi escolhido como base para o software. Sendo assim, seu desenvolvimento visa similaridade com maratonas de programação formais e informais e compatibilidade com suas regras.

Originalmente, o sistema foi pensado para agir de forma descentralizada, tendo o computador do professor como *host* e os computadores dos alunos como clientes, mas esse método se provou questionável por alguns motivos. Primeiro, para que o problema ou evento possa ser acessado pelos alunos, o computador do professor precisa estar funcionando e conectado constantemente. Segundo, no caso de eventos em larga escala, é preciso que haja uma forma de conectar vários computadores ao *host*. Isso não apenas sobrecarrega a rede desse computador, como gera possíveis brechas de segurança. Por fim, a arquitetura descentralizada dificulta a organização e manutenção dos sistemas e da base de dados. Após algumas reuniões, foi estabelecido que a centralização dos dados era necessária para organizar os problemas em um banco de dados, o que facilitaria sua organização e manutenção. Assim, a equipe envolvida no projeto adotou uma arquitetura cliente/servidor, todos aqueles que acessam o sistema passam a ser clientes. Esses clientes possuem os papeis distintos de avaliador e participante, cada qual com suas interfaces e permissões.

Na interface do avaliador, o usuário cria problemas, que devem ser resolvidos pelo participante em sua respectiva interface. A interface do avaliador também permite a criação de eventos, que reúnem um conjunto de questões que devem ser resolvidas em um limite de tempo. Durante esses eventos, a interface do avaliador também permitirá gerenciar e monitorar os participantes do evento em tempo real.

Já na interface do competidor, o usuário poderá selecionar tarefas individuais ou participar de eventos em andamento. Uma vez que uma tarefa individual ou uma tarefa do evento

tenha sido selecionada, o usuário terá uma interface para projetar um autômato finito que resolva o problema proposto. Após desenvolver uma possível solução, ela é enviada para o computador do avaliador do evento. Após o computador do avaliador fazer sua própria avaliação da solução, o resultado é enviado de volta para o competidor, que, se errar, deve corrigir sua solução, mas se acertar, pode otimizá-la ou seguir para outra tarefa. Essa otimização deve ser levada em consideração, uma vez que o jogo foi pensado com um sistema de pontuação baseado em certos elementos que incluem, mas não se resumem a, o tipo de autômato utilizado, sua complexidade, o tamanho do diagrama de estados, a dificuldade da questão e o tempo levado para elaborar a solução.

Para tornar esses mecanismos possíveis, algumas informações sobre o problema a ser elaborado dentro do sistema devem ser obtidas. Essas informações são:

1. Número de identificação da questão.
2. Descrição da questão.
3. O tipo de linguagem, segundo a Hierarquia de Chomsky.
4. Nível de dificuldade da questão, para fins de pontuação.
5. Fonte da questão, opcionalmente, para fins de documentação.
6. O "autômato-resposta", para ser comparado com o autômato do aluno.

Para distinguir cada usuário e aplicar suas restrições corretamente, um sistema de autenticação criptografado foi implementado por Abraão Victor Lopes de Farias, aluno da Universidade Federal do Ceará, campus Russas. O usuário pode optar por criar suas próprias credenciais de autenticação ou realizar essa autenticação pelo *Google*.

Em relação aos tipos de autômatos utilizados, a ideia é que, quanto mais limitado o tipo de autômato, levando em consideração a hierarquia de Chomsky, maior a pontuação. Isso se deve à ideia de que, entre dois competidores que devem encontrar soluções para o mesmo problema, vence aquele que encontrou a solução de tipo mais simples, com menor poder computacional.

Levando em consideração a comunicação via rede entre as interfaces e o servidor, o software foi pensado para ser projetado com modelo *web* por meio das linguagens *Javascript* e *Python*. Dentro da lógica do *Javascript*, o motor principal, utilizado para a construção dos autômatos, é construído com base em uma biblioteca de codificação criativa: o p5.js.

Criado por Lauren Lee McCarthy e liderada por Qianqian Ye e Evelyn Masso, o p5.js tem sua inspiração em outro projeto, o Processing. Este, criado por Ben Fry e Casey Reas

em 2001, era uma plataforma desenvolvida para trazer a simplicidade de algumas linguagens simples e educacionais, como Logo e BASIC, para linguagens mais comuns e típicas do mercado, como C++ e Java. O Processing culminou em uma linguagem que combinava design gráfico e programação, possuindo utilidade para designers que queriam fazer projetos mais interativos, mas tinham dificuldade em codificar programas complexos, e para programadores mais técnicos que queriam trabalhar com design gráfico de forma mais robusta. Estruturas de dados e saídas em texto podiam ser facilmente substituídas por informações gráficas mais dinâmicas e interativas (MCCARTHY *et al.*, 2015).

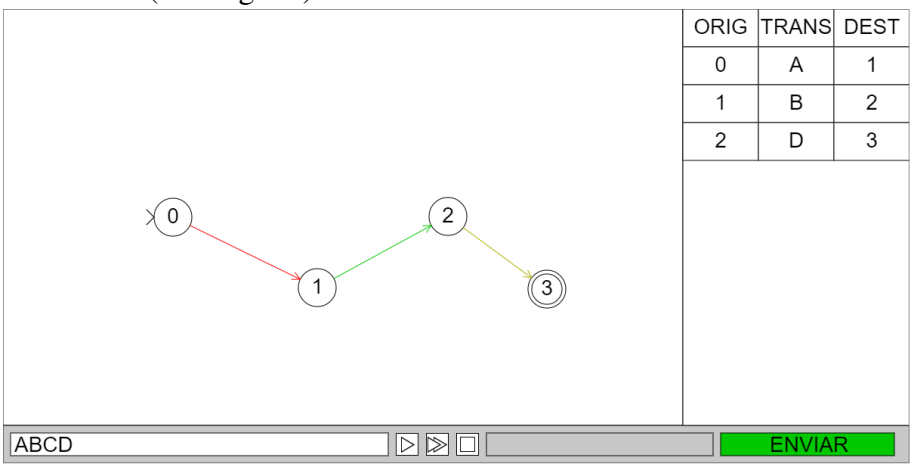
O sucesso da plataforma levantou uma possibilidade de expansão: o potencial e simplicidade do Processing poderiam ser aplicados para a web? O resultado desse questionamento foi a criação do p5.js, que trás o objetivo do Processing ao ambiente da web, através do uso de HTML e JavaScript. O propósito do p5.js era claro: integrar as capacidades do Processing, mantendo a integração com os recursos, ferramentas e *frameworks* do JavaScript, para que aqueles que tem domínio com a linguagem não sofressem qualquer tipo de restrição ao usar o p5.js (MCCARTHY *et al.*, 2015).

A decisão de usar essas linguagens e ferramentas envolve não apenas a constante comunicação entre as versões, mas também a portabilidade e a facilidade de acesso. Uma vez que o software está em suas primeiras versões, ele possui poucos recursos e é leve em tamanho. Além disso, a preocupação com a didática era crucial e deveria ser pensada desde o primeiro dia, não apenas para garantir que ela esteja correta, como também para certificar que fosse impactante. Uma das preocupações do autor deste artigo era de que o fator lúdico do jogo educacional não prejudicasse a qualidade de ensino do mesmo. O programa deve, em sua essência, instruir e divertir. Prioritariamente nessa ordem.

O primeiro passo foi desenvolver um esqueleto para a interface, considerando ceder espaço para todos os elementos sem causar excessiva poluição visual. Inicialmente, após planejamento, duas áreas principais foram pensadas para a interface inicial: uma para a construção e uma para a execução do autômato. Porém, conforme o desenvolvimento progrediu, uma terceira área foi pensada para aumentar a funcionalidade e usabilidade da ferramenta: uma tabela para exibição das transições. A figura 5 mostra uma visão geral do programa em seu estado mais recente de desenvolvimento.

Uma vez definidos os espaços de interação, o processo de construção do autômato foi pensado passo a passo, primeiro com os estados, depois com as transições. Desde o início e

Figura 5 – Captura de tela do protótipo do construtor de autômatos (Visão geral)



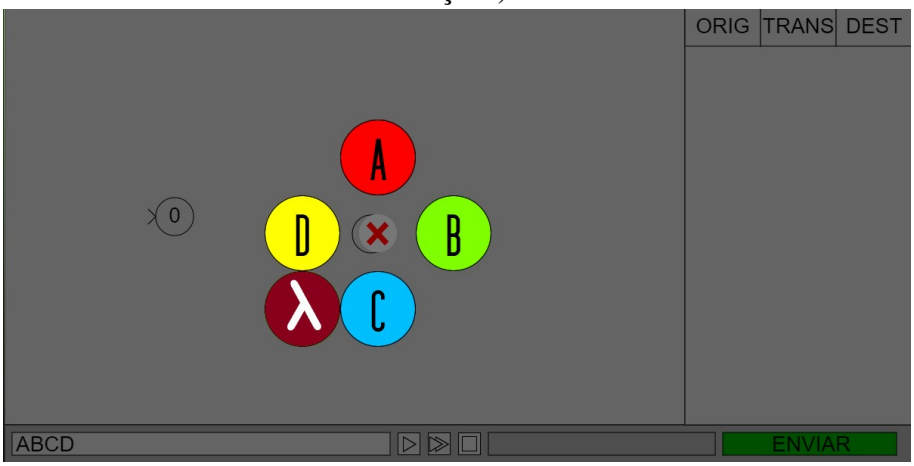
Fonte: o autor

durante cada iteração estável, a usabilidade foi pensada para tornar o programa fácil de usar e com respostas visuais constantes ao usuário, para que ele tenha constante noção do que está fazendo. Nas primeiras etapas, o processo de identificação das transições envolvia o reconhecimento por cor, mas logo se viu que isso poderia ser um problema para portadores de daltonismo. Atualmente, esse é um dos propósitos da tabela de transições: permitir que as transições entre estados possam ser identificadas por algo além de sua cor. Essa tabela pode ser vista à direita da figura 5.

A maioria das funcionalidades foi pensada para ser controlada pelo mouse. Um clique único dentro de qualquer área vazia cria um estado. Clicar em um estado e arrastar o mouse até outro estado qualquer cria uma transição entre dois estados, assim como clicar em um estado e arrastar para fora e para dentro do mesmo estado cria uma transição de mesmo estado. Ao criar uma transição, um menu é exibido para que o usuário possa decidir qual é o símbolo da transição. Na primeira versão, o usuário poderia escolher entre quatro símbolos específicos. Na versão atual, foi adicionada a transição em vazio (lambda). Foram estudadas algumas possibilidades para dinamizar esse processo, permitindo que o usuário pudesse adicionar quantas transições quisesse e nomeá-las como bem entendesse. Isso ficou definido para desenvolvimento futuro. A versão atual do menu de transições pode ser vista na figura 6.

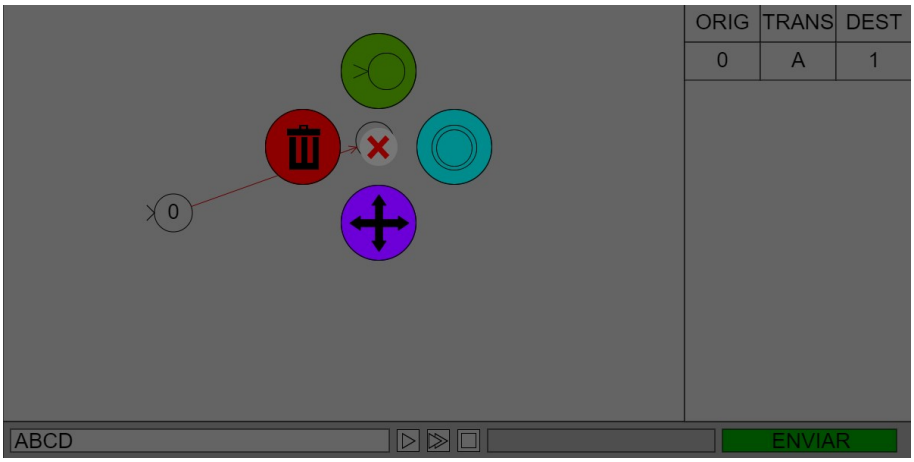
De forma semelhante, o usuário pode clicar e segurar um estado sem arrastá-lo. Isso abrirá o menu de estados. Este menu permite que quatro ações distintas relacionadas ao estado sejam feitas: definir esse estado como o estado inicial, definir esse estado como um estado de aceitação, mover o estado de lugar ou excluir o estado e todas as transições relacionadas à ele. Isso pode ser visto na figura 7.

Figura 6 – Captura de tela do protótipo do construtor de autômatos (com foco no menu de transições)



Fonte: o autor

Figura 7 – Captura de tela do protótipo do construtor de autômatos (com foco no menu de estados)



Fonte: o autor

Além das restrição no uso das transições, causado pela criação do menu citado acima, outro problema surgiu ao implementar os botões desses menus. Como eles não podiam ser criados proceduralmente, devido ao custo de processamento desnecessário que isso causaria, cada botão foi previamente desenhado e importado para o software, mas isso não funcionou, uma vez que os mecanismos de segurança nos navegadores identificaram essa importação como uma brecha de segurança. Para que os arquivos pudessem ser devidamente importados pelo software, permitindo assim sua depuração e teste, as imagens tiveram que ser adicionadas a um repositório público. Essa prática não é recomendada, pois torna o programa dependente de uma plataforma de terceiros. Após diálogo com o colaborador Abraão, percebemos que isso poderia ser resolvido, mas exigiria desenvolvimento futuro.

Uma vez que o processo de construção do autômato estava completo, era necessário

implementar sua lógica de execução. Por ser o método mais fácil de se trabalhar, devido à simplicidade de seu poder computacional e a ausência de restrições complexas, a lógica de execução escolhida para o protótipo foi a do autômato finito não-determinístico. A fundamentação lógica foi criada a partir de controles exibidos no espaço de execução. Esses controles podem ser visualizados na parte inferior da figura 5 e consistem, da esquerda para a direita, em uma caixa de entrada de texto para que o usuário digite uma cadeia de entrada, um botão que permite a execução completa do autômato construído, um botão que permite a execução passo a passo do autômato para fins educacionais e de depuração e uma caixa de texto de saída, que informa se o autômato aceitou ou rejeitou a cadeia informada, ou se houve um erro durante a execução.

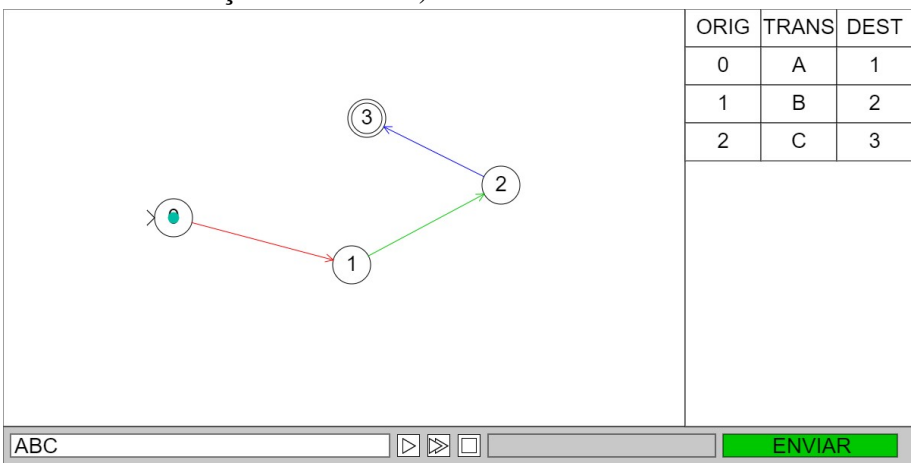
À direita da caixa de texto de saída mencionada acima e visível no canto inferior esquerdo da figura 5, está um botão verde escrito "enviar". Esse botão foi pensado com o propósito de permitir o envio da solução para o servidor, para que ela seja testada, embora essa funcionalidade ainda não tenha sido implementada. Porém, um outro botão na interface de funcionamento, criado por Abraão fora do motor principal, já realiza exatamente a mesma função. Tendo em vista que, obviamente, ter dois botões que façam a mesma coisa é algo contraprodutivo e que gera problemas sérios de usabilidade, o primeiro botão será devidamente removido antes de sua versão final.

Recapitulando cada elemento do painel de execução, ilustrado na parte inferior da figura 5, a caixa de texto de entrada permite que o usuário informe a cadeia de entrada com a qual o autômato finito construído será executado para testá-lo. O usuário pode informar a cadeia usando o teclado, digitando as teclas A, B, C e D para os respectivos símbolos do alfabeto e usando o "BACKSPACE"para corrigir a cadeia quando cometer um erro. Por padrão, a caixa de entrada inicia pré-definida com a cadeia "ABCD", como pode ser visto na mesma figura, mas o usuário pode imediatamente substituí-la por uma de sua preferência.

Logo à direita da caixa de entrada, encontram-se três botões, como podem ser vistos na mesma figura 5. Estes são, respectivamente, da esquerda para a direita, o botão de execução passo-a-passo, o botão de execução completa, e o botão de interromper execução. O primeiro permite que o usuário execute o autômato construído usando a cadeia criada. Pressionar o botão uma vez fará com que um ponteiro, ilustrado por um ponto ciano, surja sobre o estado inicial, dando início ao processo de execução, conforme exibe a figura 8. Pressionar o botão uma segunda vez fará com que a execução "consuma"um elemento da cadeia. Esse elemento aparece na caixa de entrada com a cor ciano e o ponteiro se move através da transição (ou transições) que

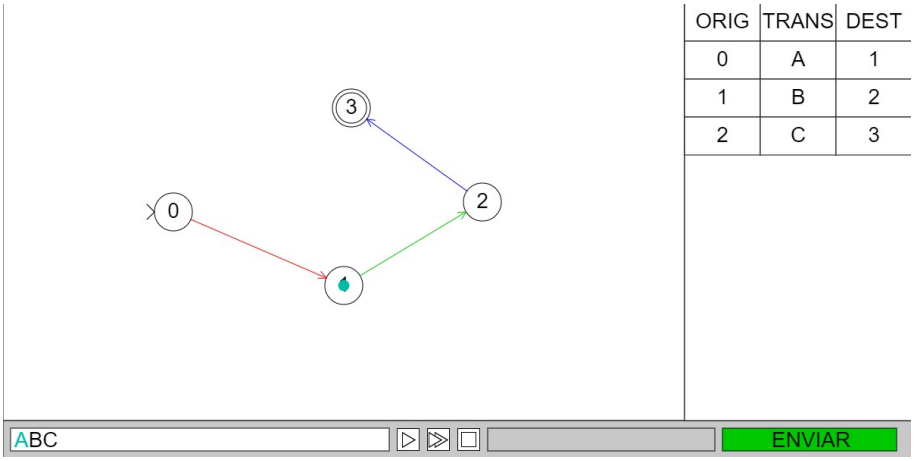
corresponderem ao símbolo consumido. Esse processo é ilustrado na figura 9. Respeitando as regras dos autômatos finitos não-determinísticos, caso haja mais de uma transição com o mesmo símbolo e mesma origem, um ponteiro é criado em cada estado de destino. De forma semelhante, se não houverem transições com esse símbolo, o ponteiro é "perdido". A cada vez que o botão for pressionado novamente, mais um elemento da cadeia é consumido e o(s) ponteiro(s) se move(m) novamente. Se a cadeia de entrada foi completamente consumida, pressionar o botão mais uma vez terminará a execução e exibirá o resultado na caixa de saída, o que é retratado na figura 10. Isso será melhor elaborado nos parágrafos subsequentes.

Figura 8 – Captura de tela do protótipo do construtor de autômatos (a execução foi iniciada)



Fonte: o autor

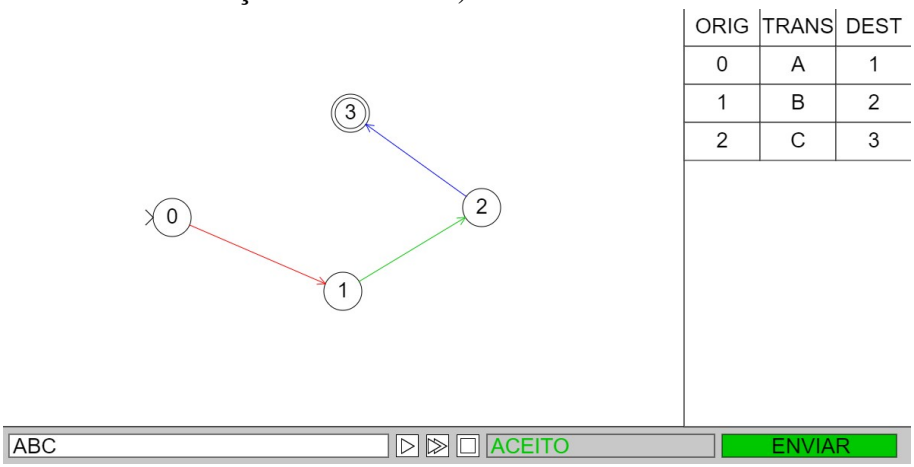
Figura 9 – Captura de tela do protótipo do construtor de autômatos (a execução avançou um passo)



Fonte: o autor

O segundo botão faz a mesma coisa que o primeiro botão, exceto que, enquanto o primeiro botão executa o autômato passo-a-passo e oferece um retorno visual, mostrando cada

Figura 10 – Captura de tela do protótipo do construtor de autômatos (a execução foi concluída)



Fonte: o autor

etapa da execução, o segundo botão executa o autômato completamente de uma vez, retornando o resultado imediatamente na caixa de saída. O terceiro botão interrompe a execução em qualquer estágio e limpa a caixa de saída, permitindo que você realize uma nova execução.

Por fim, a última caixa, à direita dos botões de execução, é a caixa de saída, que informa o resultado da execução. Ela pode retornar três resultados. Se a cadeia foi completamente consumida e algum ponteiro terminou em um estado de aceitação, o resultado é dado como "aceito". Se nenhum ponteiro tiver terminado em um estado de aceitação, o resultado é dado como "rejeitado". Se a cadeia não foi completamente consumida, mas todos os ponteiros foram perdidos (não há nenhuma transição partindo do(s) estado(s) atual(is) com o símbolo sendo lido na cadeia de entrada), o resultado é dado como um "erro", indicando que o autômato precisa ser revisado e corrigido.

Uma vez que a lógica do Autômato Finito Não Determinístico (AFND) estava definida, era necessário trabalhar os requisitos para alcançar a lógica do Autômato Finito Determinístico (AFD). Ambos deveriam ser claramente definidos e separados, pois, uma vez que se trata de um jogo educacional, quando dois jogadores resolvem o mesmo problema usando autômatos finitos, porém um deles usa AFD, este merece uma pontuação extra pelo esforço aplicado, mesmo que ambos possuam o mesmo poder computacional, afinal, os AFD possuem maiores restrições e são mais difíceis de construir do que os AFND. Logo, era necessário que o sistema soubesse distinguir os dois tipos de autômatos. Isso foi possível criando um algoritmo que analisa o autômato construído, levando em consideração algumas restrições:

- Um AFD possui uma, e apenas uma, transição de cada elemento pertencente ao alfabeto partindo de todos os autômatos.

- Um AFD não possui transições em vazio.

Com base nessas restrições, o sistema permite que o usuário selecione o tipo de autômato finito com o qual deseja trabalhar. Se, após concluído, o autômato deixar de cumprir algum requisito, o sistema informará o usuário e mudará o tipo de autômato para o mais adequado. Da mesma forma, se o usuário desenvolver um autômato que atende os requisitos de um tipo mais restrito, o sistema perguntará ao usuário se ele deseja mudar o tipo do autômato de acordo. Essa mudança leva em consideração a hierarquia de Chomsky, embora, no presente momento, a aplicação está preparada para lidar apenas com os tipos AFD e AFND.

Além do que já foi mencionado acima, como um dos responsáveis pelo planejamento e o principal responsável pela lógica e design do "front-end"da aplicação, o autor realizou diversas tarefas ao longo do seu desenvolvimento. Conforme o projeto se tornou cada vez mais robusto, novas ideias e necessidades foram surgindo, o que abriu espaço para novos planejamentos, onde tais ideias eram elaboradas, priorizadas ou simplesmente descontinuadas. O quadro 1 oferece uma visão geral dos recursos e elementos que foram desenvolvidos no decorrer do projeto, bem como os que estão sendo desenvolvidos nesse momento, e os que já foram avaliados e aprovados para desenvolvimento futuro.

Quadro 1 – Recursos e funcionalidades planejadas para o jogo educacional e seus estados de desenvolvimento

Recurso planejado	Estado do desenvolvimento
Criação e manipulação dos estados	Concluído
Criação e manipulação das transições	Concluído
Funcionalidade dos estados de aceitação	Concluído
Implementação do menu de estados	Concluído
Implementação do menu de transições	Concluído
Funcionalidade do estado inicial	Concluído
Funcionalidade de mover um estado de lugar	Concluído
Funcionalidade de remover um estado e as transições relacionadas	Concluído
Implementação do painel de execução	Concluído
Implementação da digitação da cadeia de entrada	Concluído
Implementação do motor de execução, botões de execução e painel de saída	Concluído
Implementação do controle de seleção entre AFD e AFND	Concluído
Procedimento para detectar se um autômato finito é determinístico	Concluído
Implementação da transição em vazio (lambda)	Concluído
Implementação de um mecanismo para informar que o autômato finito informado e o construído são diferentes	Concluído
Implementação da tabela de transições	Concluído
Funcionalidade de remover uma transição específica	Concluído
Melhorias para identificação do estado	Concluído
Implementação de comparador de autômatos finitos	Em andamento
Análise e refatoração do código-fonte para melhor organização e desempenho	Em andamento
Remoção do botão de envio e reajuste das dimensões da barra de execução	A ser feito
Substituição do menu de transições com alfabeto fixo por um menu com alfabeto dinâmico	A ser feito
Implementação de animações procedurais ao executar ações	A ser feito
Embelezamento e melhoria de design	A ser feito
Transferência as imagens importadas para uma base de dados local, retirando a dependência do repositório de terceiros	A ser feito

Fonte: Elaborado pelo autor.

5 CONCLUSÕES

O projeto tem como objetivo levantar uma problemática presente, mas pouco abordada, dentro do ambiente educacional de computação e sua atratividade ao público. O uso da abordagem gamificada como método de ensino e estudo alternativos é relativamente recente e guiado pelas necessidades do mercado de trabalho. Ele aumenta a compreensão de conceitos práticos e da aplicação do que é aprendido no mercado de trabalho, mas, infelizmente, ainda encontra dificuldade para se inserir em disciplinas mais teóricas, indispensáveis para a área. Os livros e a didática dos professores é essencial e útil para os assuntos mais conceituais dessas disciplinas, mas, ao chegar na prática e na resolução de exercícios, a carência de metodologias alternativas para facilitar o aprendizado se torna mais nítida. Por não fixarem devidamente a prática, resolver os problemas apresentados nas avaliações se torna mais difícil do que o necessário, impactando negativamente o desempenho.

E levando em consideração o aumento exponencial do interesse em computação e a lucratividade do mercado atrelado à ela, essa dificuldade gera um efeito desastroso à longo prazo. Alunos que não possuem interesse em se especializar nesse tipo de área, enfrentam dificuldade em compreender a teoria mesmo após a leitura, tem dificuldade para realizar os exercícios e acabam por receber notas ruins nas avaliações, acabam por ver essas disciplinas como obstáculos, e essa má impressão, formentada e corroborada, acaba por levar à criação de preconceitos, esteriótipos e "tabus", que são passados adiante para outros alunos, repelindo-os da área antes mesmo que estes recebam qualquer experiência ou impressão inicial, e quando eles finalmente a recebem, se deparam com a mesma aparente frustração. Afinal, os alunos acabam "vendo o que esperam ver", e os professores, por mais dinâmicos e dispostos a ensinar que estejam, só podem levar o interesse do aluno até certo ponto, ainda mais se ele já foi prejudicado por uma baixa expectativa.

O autor deste artigo utilizou de sua experiência pessoal para inspirá-lo a desenvolver esse projeto. Durante sua experiência no curso de Ciência da Computação, Campus Russas, ao conversar com outros alunos mais avançados, alguns deles levantaram a dificuldade de disciplinas mais teóricas, vistas normalmente na segunda metade do curso, devido à dificuldade de compreensão dos conceitos e as opções limitadas de estudo, que apesar de serem importantes, possuem cada vez menos engajamento devido ao uso cada vez maior de ferramentas digitais. Disciplinas com maior foco na prática possuem ferramentas gamificadas que ajudam e estimulam a prática, mas como essas metodologias ativas são quase inexistentes nas disciplinas teóricas, a

prática, que poderia complementar e aprimorar o conhecimento teórico, acaba sendo limitada.

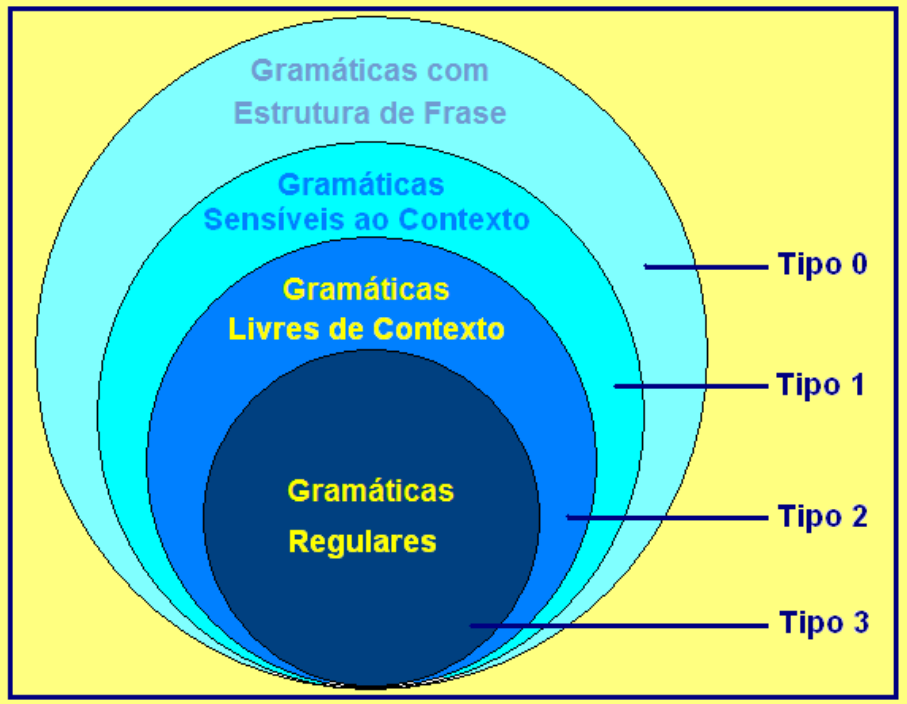
Algo tão sério mostra a relevância deste artigo e traz atenção para essa adversidade importante e pouco observada, agindo como uma porta de entrada para futuros pesquisadores e abrindo a possibilidade de aumentar o interesse, engajamento e satisfação, não apenas na área, mas em toda a ciência da computação. Dito isso, o desenvolvimento da aplicação, apesar de estar longe de sua conclusão, mostrou possibilidades bastante convidativas. De fato, existem muitas abordagens diferentes para o desenvolvimento futuro, e, por serem independentes entre si, elas podem evoluir simultaneamente.

A principal abordagem para seu desenvolvimento futuro envolve a ideia com a qual ele foi inspirado originalmente. Uma vez que uma versão estável com todas as suas funcionalidades básicas tenha sido desenvolvida, serão buscadas oportunidades de testá-la em ambientes reais escaláveis. Por exemplo, o software seria primeiramente testado em uma sala de aula, depois em uma competição entre turmas, depois em um evento universitário. Durante os testes, novos recursos, otimizações e melhorias seriam implementados ao sistema, até que este esteja apto a se tornar um programa utilizado em competições nacionais e mundiais. Tais testes também ofereceriam retorno direto através de *feedback*, auxiliando na detecção de problemas e em análises estatísticas para encontrar o melhor processo de evolução.

A evolução desses recursos levaria em consideração a hierarquia de Chomsky, ilustrada na figura 11 acrescentando a construção e execução de cada tipo gradualmente, com base no poder computacional e complexidade de cada tipo de autômato. Esse desenvolvimento levaria em consideração a estabilidade e usabilidade, exibindo e ocultando os recursos como for conveniente. Além disso, garantir a estabilidade e bom funcionamento é fundamental, por isso, é preciso que os recursos atuais do projeto funcionem adequadamente, antes de adicionar novos.

Outro possível caminho a ser seguido envolve transformar o jogo educacional em algo para estimular crianças e jovens a adquirir interesse e engajar na área. Nesse contexto, o jogo educacional carrega um sistema de regras diferente, voltado para o público-alvo que não possui experiência com autômatos finitos. Assim, ele assumiria um sistema de regras mais próximo de um jogo de quebra-cabeças, com uma introdução às regras, uma dificuldade progressiva e um conjunto de desafios que estimulem a criatividade. Essa versão traria características mais lúdicas e seria mais visualmente atraente, em detrimento da simplicidade e do uso profissional. Nomenclaturas e jargões científicos, por exemplo, seriam substituídos por explicações mais concretas e fáceis de compreender, algo que não poderia ser feito em uma vertente educacional

Figura 11 – Diagrama representando a hierarquia de Chomsky



Fonte: https://pt.wikipedia.org/wiki/Hierarquia_de_Chomsky/media/Ficheiro:Hierarquia_de_Chomsky.PNG

voltada ao público universitário, uma vez que essas terminologias são essenciais para esse ambiente.

Além das possibilidades acima, conforme o desenvolvimento do projeto avança, novas tecnologias e habilidades surgem, o que pode levar a novos recursos e melhorias ao desenvolvimento do software, bem como o próprio software pode contribuir com o estudo e os avanços científicos nas áreas de gamificação e de métodos alternativos de aprendizagem.

Para avaliar melhor qual caminho deve ser seguido, o desenvolvimento deve primeiramente chegar em um estado de funcionalidade e estabilidade que permita testes de campo com possíveis usuários do *software*. Os resultados desses testes poderão indicar o que é mais desejável no sistema e quais carências ele pode preencher, oferecendo uma orientação mais precisa do desenvolvimento futuro.

Diante de novas possibilidades e de novos avanços, muitos elementos presentes dentro da teoria da computação precisam ser revisados e ampliados, e isso só pode ser feito se a área permanecer relevante na sociedade, através de futuros bons cientistas e profissionais que possam expandir o conhecimento. Mas, no momento presente, estes precisam descobrir e adquirir interesse pela teoria da computação, e este projeto busca, principalmente, alcançar essas grandes mentes.

REFERÊNCIAS

ATTLE, S.; BAKER, B. Cooperative learning in a competitive environment: Classroom applications. **International Journal of Teaching & Learning in Higher Education**, Citeseer, v. 19, n. 1, 2007.

BARTH, Z. **Zachtronics**. 2020. Disponível em: <http://www.zachtronics.com>.

BEZ, J. L.; TONIN, N. A. Uri online judge e a internacionalização da universidade. **Disponıvel em https://scratch. mit. edu/Acessado em**, v. 15, 2014.

BURGUILLO, J. C. Using game theory and competition-based learning to stimulate student motivation and performance. **Computers & education**, Elsevier, v. 55, n. 2, p. 566–575, 2010.

CAMPOS, C. P. de; FERREIRA, C. E. Boca: um sistema de apoio a competiç oes de programaç ao. 2004.

CUNHA, G.; BARRAQUI, L.; FREITAS, S. A. A. de. Uso da gamificação nos anos iniciais do ensino fundamental brasileiro. In: **Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)**. [S.l.: s.n.], 2017. v. 28, n. 1, p. 1742.

DIVERIO, T. A.; MENEZES, P. B. **Teoria da Computação–UFRGS: Máquinas Universais e Computabilidade**. [S.l.]: Bookman Editora, 2009.

FOLMER, E. Component based game development–a solution to escalating costs and expanding deadlines? In: SPRINGER. **International Symposium on Component-Based Software Engineering**. [S.l.], 2007. p. 66–73.

KIRRIEMUIR, J.; MCFARLANE, A. Literature review in games and learning. 2004.

MAEKAWA, C.; NAGAI, W.; IZEKI, C. Relato de gamificação da disciplina projeto e análise de algoritmos do curso de engenharia de computação. In: **Anais dos Workshops do Congresso Brasileiro de Informática na Educação**. [S.l.: s.n.], 2015. v. 4, n. 1, p. 1425.

MCCARTHY, L.; REAS, C.; FRY, B. **Getting started with P5. js: Making interactive graphics in JavaScript and processing**. [S.l.]: Maker Media, Inc., 2015.

MCGONIGAL, J. **Reality is broken: Why games make us better and how they can change the world**. [S.l.]: Penguin, 2011.

MCGONIGAL, J. **A realidade em jogo**. [S.l.]: Editora Best Seller, 2017.

MENABREA, L. F.; LOVELACE, A. **Sketch of the analytical engine invented by Charles Babbage**. 1842.

NAVARRO, G. Gamificação: a transformação do conceito do termo jogo no contexto da pós-modernidade. **Biblioteca Latino-Americana de Cultura e Comunicação**, v. 1, n. 1, p. 1–26, 2013.

PIEKARSKI, A. E.; MIAZAKI, M.; HILD, T.; MULATI, M. H.; KIKUTI, D. A metodologia das maratonas de programação em um projeto de extensão: um relato de experiência. In: **Anais dos Workshops do Congresso Brasileiro de Informática na Educação**. [S.l.: s.n.], 2015. v. 4, n. 1, p. 1246.

PIRES, F.; TEIXEIRA, K.; PESSOA, M.; LIMA, P. Desenvolvendo o pensamento computacional através da máquina de turing: o enigma do curupira. In: SBC. **Anais do XXVII Workshop sobre Educação em Computação**. [S.l.], 2019. p. 523–532.

PIRES, F. G. de S.; DUARTE, J. C.; PESSOA, L. da S.; PEREIRA, K. S. dos S.; MELO, R.; FREITAS, R. de. Uma análise cognitiva entre a emergência de padrões em narrativas infantis e elementos do pensamento computacional. In: **Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)**. [S.l.: s.n.], 2018. v. 29, n. 1, p. 1193.

RESNICK, M.; ROBINSON, K. **Lifelong kindergarten: Cultivating creativity through projects, passion, peers, and play**. [S.l.]: MIT press, 2017.

SAVI, R.; ULBRICHT, V. R. Jogos digitais educacionais: benefícios e desafios. **RENOTE-Revista Novas Tecnologias na Educação**, v. 6, n. 1, 2008.

SIGNORI, G. G.; GUIMARÃES, J. C. F. de; CORRÊA, S. Gamificação como método de ensino inovador. In: **XVI Mostra de Iniciação Científica, Pós-graduação, Pesquisa e Extensão**. [S.l.: s.n.], 2016.

SIPSER, M. **Introdução à teoria da computação**. [S.l.]: Thomson Learning, 2007.

TURING, A. M. On computable numbers, with an application to the entscheidungsproblem. **Proceedings of the London mathematical society**, Wiley Online Library, v. 2, n. 1, p. 230–265, 1937.

WING, J. M. Computational thinking. **Communications of the ACM**, ACM New York, NY, USA, v. 49, n. 3, p. 33–35, 2006.

ZANINI, A. S.; RAABE, A. L. A. Análise dos enunciados utilizados nos problemas de programação introdutória em cursos de ciência da computação no brasil. In: **Anais do XXXII Congresso da Sociedade Brasileira de Computação, XX WEI–Workshop sobre Educação em Computação**. [S.l.: s.n.], 2012.