UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO

Gabriele Pessoa do Nascimento

Programinó: um Jogo para Auxílio ao Aprendizado do Assunto de Tipos de Dados na Programação

Recife

2019

Gabriele Pessoa do Nascimento

Programinó: um Jogo para Auxílio ao Aprendizado do Assunto de Tipos de Dados na Programação

Monografia apresentada ao Curso de Bacharelado em Sistemas de Informação da Universidade Federal Rural de Pernambuco, como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Universidade Federal Rural de Pernambuco – UFRPE

Departamento de Estatística e Informática

Curso de Bacharelado em Sistemas de Informação

Orientadora: Taciana Pontual

Coorientador: Pablo Sampaio

Recife

2019

Dados Internacionais de Catalogação na Publicação
Universidade Federal Rural de Pernambuco
Sistema Integrado de Bibliotecas
Gerada automaticamente, mediante os dados fornecidos pelo(a) autor(a)

244p Nascimento, Gabriele
Programinó: um Jogo para Auxílio ao Aprendizado do Assunto de Tipos de Dados na Programação / Gabriele Nascimento. - 2019.
50 f. : il.

Orientadora: Taciana Pontual.
Coorientadora: Pablo Sampaio.
Inclui referências e apêndice(s).

Trabalho de Conclusão de Curso (Graduação) - Universidade Federal Rural de Pernambuco, Bacharelado em Sistemas da Informação, Recife, 2020.

1. introdução à programação. 2. tipos de dados. 3. virtualização de jogos. 4. inteligência artificial. 5. minimax. I. Pontual, Taciana, orient. II. Sampaio, Pablo, coorient. III. Título

CDD 004

GABRIELE PESSOA DO NASCIMENTO

Programinó: um Jogo para Auxílio ao Aprendizado do Assunto de Tipos de Dados na Programação

Monografia apresentada ao Curso de Bacharelado em Sistemas de Informação da Universidade Federal Rural de Pernambuco, como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Recife, 10 de Novembro de 2020.

BANCA EXAMINADORA

_____

Profa. Taciana Pontual
Departamento de Computação
Universidade Federal Rural de Pernambuco

_____

Prof. Pablo Sampaio
Departamento de Computação
Universidade Federal Rural de Pernambuco

_____

Prof. Gabriel Alves
Departamento de Estatística e Informática
Universidade Federal Rural de Pernambuco

À minha família,
Aos meus amigos,
Aos meus mestres.

Agradecimentos

Cada pessoa presente nestes agradecimentos teve sua parcela de amor e força doados a mim para que eu pudesse chegar até aqui. Acredito que já consegui externar pessoalmente a minha gratidão a todos os citados, mas permanecerá, aqui, o registro para que outras pessoas vejam a minha eterna gratidão.

Começo agradecendo a Deus, ao universo e a todas as energias boas e positivas por me darem forças para continuar, mesmo quando tudo parecia apenas névoa.

Agradeço amorosamente à minha mãe Girlania, meu pai Gustavo, minhas irmãs Grasiele e Gisele, minha avó Mariléa e meus sobrinhos Vi, Gui e Tety, por sempre acreditarem no meu potencial e crescimento.

Agradeço à minha amiga-irmã Thallytha pelo amor de sempre e ao meu afilhado Thalles, que ela e a vida me deram de presente para iluminar muito mais os meus dias.

Também agradeço ao anjo da minha vida, Cristina Ramos, que sempre foi muito mais que uma tia, mãe, amiga e espelho. Tenho muito orgulho de dizer que você é a minha estrela-guia em todos os âmbitos da vida. Obrigada!

Agradeço imensamente ao melhor presente que esta graduação poderia me trazer, meu companheiro Daniel Cândido. Obrigada por ter estado ao meu lado, me apoiando e ajudando sempre que necessário. Além de ser a melhor dupla que poderíamos ser, foi profundamente essencial para que este trabalho fosse pensado, amadurecido e desenvolvido como gostaria desde o começo.

Agradeço a todos da família Cândido e Souza, principalmente à Ceça, Luiz, Darlan e David, e aos amigos do grupo Tente Outra Vez, por todo cuidado e pelas energias boas durante boa parte da graduação.

Agradeço às amigas Lisandra, Raissa, Stefany e Brona, maravilhosas e empoderadas que foram essenciais na luta diária durante o curso e que saio levando-as da universidade para a vida. E aos demais queridos amigos conquistados na universidade que também levo no coração. Em especial, Raffael Vieira, Maria Luíza, Demis, Tancicleide, Brunna, Marta, Ícaro, Daivid, François, Matheus Uehara, Gustavo, Ricardo, Yuri, Robson Ugo, Rodolfo, Edvan, Jorge, João, Delando, José Fernando, Júnior e Victor Leuthier.

Também agradeço aos queridos que, de alguma forma, sempre incentivaram muito a minha contínua evolução como pessoa e profissional, alguns antes mesmo da minha entrada na universidade. Em especial, Isa, Débora, Diego, Laise, Pedro, Serginho, Adriel, Priscila Araújo, Flaviano, Jackson, Marcolino, Ricardo e Felipe.

Agradeço demais aos queridos professores e orientadores que não só me ajudaram a guiar este trabalho, mas também a não surtar além da conta. Obrigada por toda paciência, carinho e inspiração! Em especial, Taciana, Gabriel, Ceça, Roberta, Silvana, Jones, Filipe e Pablo.

Também agradeço aos queridos mentores do CESAR e CESAR School por toda inspiração, carinho e empatia durante meu processo de término de curso e construção deste trabalho. E a todos que fazem ou fizeram parte da família ETE Porto Digital. Em especial, Maurício, Danielle, Luiz, Anderson, aluninhos e aluninhas, Felipe, Andrea, Cloves, Laura e Marcos.

Agradeço especialmente a mim mesma, por continuar e evoluir em busca de uma vida e um mundo melhor. É gratificante poder olhar para trás e ver o quanto aquela garotinha se empoderou e se fortaleceu, mesmo diante de tantas dificuldades, e poder continuar passando essa mensagem adiante.

Por fim, mas não menos importante, agradeço a Universidade Federal Rural de Pernambuco, a Ruralinda, por todo seu acolhimento e empatia com os estudantes.

“Depois de todas as tempestades e naufrágios, o que fica de mim
e em mim é cada vez mais essencial e verdadeiro.”
(Caio Fernando Abreu)

“Eu não sou um ser humano, sou uma ideia.
E não adianta tentar acabar com as ideias.”
(Luiz Inácio Lula da Silva)

“Liberdade é pouco. O que desejo ainda não tem nome.”
(Clarice Lispector)

“Welcome to the real world! It sucks. You’re gonna love it!”
(Monica Geller)

“Onde não puderes amar, não se demores.”
(Frida Kahlo)

“I’ll be there for you.”
(FRIENDS)

“Lugar de mulher é onda ela quiser!”

Resumo

A era digital em que vivemos faz com que nós estejamos sempre imersos em tecnologias cada vez mais ubíquas. Para que este contato com a tecnologia permaneça de forma saudável, é preciso aprender a consumi-la de forma consciente, e além disso, aprender a desenvolvê-la em diferentes contextos; pois, desta forma, teremos soluções cada vez mais inclusivas. Sobre desenvolvimento de soluções, por mais que tenhamos diversos artefatos facilitadores, o processo de ensino-aprendizagem de programação ainda é um desafio, principalmente para estudantes iniciantes. Lidar com tantos estímulos, concorrentes e constantes e ainda ter a capacidade de abstrair e assimilar conceitos de programação que não é trivial e nem é trabalhado desde a infância, por isso, artefatos lúdicos, como os jogos digitais, são essenciais para facilitar os primeiros contatos com a programação. Neste contexto, este trabalho traz para a sociedade um jogo educacional digital que trabalha o assunto de tipos de dados na programação, o Programinó, para que estudantes iniciantes possam praticar e consolidar o conteúdo através de uma ferramenta lúdica. O jogo foi desenvolvido com três níveis de dificuldade, um fácil, um médio e um difícil. O difícil aplica o algoritmo minimax adaptado, enquanto o fácil usa o mesmo minimax adaptado de maneira invertida. Já o nível médio utiliza um algoritmo aleatório. Como forma de validar os níveis de dificuldades, foram realizados experimentos comparativos que comprovaram que o minimax perdeu em apenas 5,6% das vezes; ganhando em 49,7% ou empatando nas demais partidas.

Palavras-chave: introdução à programação, tipos de dados, jogos, virtualização, ludicidade, inteligência artificial, minimax.

Abstract

The digital age we live in means that we are always immersed in ever more ubiquitous technologies. For this contact with technology to remain healthy, it is necessary to learn to consume it consciously, and to learn to develop it in different contexts; therefore, we will have increasingly inclusive solutions. Regarding solution development, as much as we have several facilitating artifacts, the process of programming teaching and learning is still a challenge, especially for beginning students. Dealing with so many competing and constant stimuli and still having the ability to abstract and assimilate programming concepts that is not trivial and has not been worked on since childhood, so playful artifacts such as digital games are essential to facilitate first contacts with the schedule. In this context, this work brings to society a digital educational game that deals with the subject of data types in programming, Programinó, so that beginning students can practice and consolidate the content through a playful tool. The game was developed with three levels of difficulty, one easy, one medium and one hard. The hard one applies the adapted minimax algorithm, while the easy one uses the same minimax adapted in an inverted way. The middle level uses a random algorithm. As a way to validate the difficulty levels, comparative experiments were performed that showed that the minimax lost only 5.6% of the time; winning at 49.7% or drawing in the remaining matches.

Keywords: Introduction to programming, data types, games, virtualization, playfulness, artificial intelligence, minimax.

Lista de ilustrações

Lista de abreviaturas e siglas

ETE	*Escola Técnica Estadual*
ETE PD	*Escola Técnica Estadual Porto Digital*
GB	*Gigabyte*
HD	*Hard Disk* ou Disco rígido
IA	*Inteligência Artificial*
IDE	*Integrated Development Environment*
RAM	*Random Access Memory*
SBIE	*Simpósio Brasileiro de Informática na Educação*
SSD	*Solid-State Drive*
WIE	*Workshop de Informática na Escola*

Sumário

1 Introdução

Estamos vivendo uma era digital, que exige competências e habilidades tecnológicas para se manter no mercado de maneira bem-sucedida. Difícil imaginar um dia-a-dia sem usar tecnologias digitais, mas as competências que têm ganhado importância vão além disso. O estudo de programação tem se tornado fundamental, independente da área, pois, além de estudantes não se limitarem ao consumo de tecnologias, eles ainda desenvolvem um conjunto de conhecimentos capazes de resolver problemas reais, estando mais preparados para as exigências do século XXI (COMPUTING IN THE CORE, 2018). Segundo (WING, 2006), o pensamento computacional é uma habilidade fundamental para todos, não somente para cientistas da computação. Entretanto, durante o processo de ensino-aprendizagem de algoritmos é possível observar, em uma parte significativa dos estudantes (GOMES et al., 2015), muitas dificuldades em compreender e aplicar conceitos abstratos. A autora deste trabalho compartilha de tal constatação, por ser professora de Introdução à Programação no ensino médio-técnico, daí a motivação pessoal para o desenvolvimento deste trabalho. A revisão sistemática de (SILVA et al., 2015) mostra que 60% dos artigos sobre ensino-aprendizagem de programação focam em educação superior, apenas 22% no nível médio e 3% no ensino técnico. Isso nos revela a importância da preocupação deste trabalho ter surgido a partir de um cenário médio-técnico. Além disso, os autores mostram também que 20% dos artigos sugerem jogos como uma boa ferramenta de apoio ao ensino-aprendizagem de programação, ficando atrás apenas de ferramentas de softwares em geral.

1.1 Motivação

Os jogos surgem como uma oportunidade de trazer o assunto de forma lúdica. Dessa forma, o conteúdo se apresenta de forma indireta, trazendo junto a diversão e os desafios, atendendo às novas gerações de estudantes, cada vez mais conectados. (SILVA et al., 2015) ainda complementa que jogos, especificamente digitais, também são destaques no ensino-aprendizagem de programação e que, em sua maioria, são voltados a melhoria de rendimento de estudantes iniciantes.

O jogo de dominó, por exemplo, é um importante recurso didático no ensino de conteúdos de matemática (SANTOS, 2013; GODOY et al., 2015), permitindo construir definições, propriedades, raciocínio e operações matemáticas, português (BRASIL ESCOLA, 2018) no ensino de sinônimos e antônimos, química (FIALHO, 2008) no ensino de elementos e símbolos da tabela periódica, biologia (SILVA; MORAES, 2011) para o ensino de morfologia vegetal, (CANDIDO et al., 2012) para ensino de artrópodes, entre

outros.

Para tentar superar as dificuldades no ensino-aprendizagem, educadores e pesquisadores buscam alternativas através de ferramentas e metodologias. Conforme (ARAUJO et al., 2013), o hábito de estudo influencia diretamente no desempenho do estudante iniciante de programação, principalmente se este for um hábito diário. Além disso, encontram-se artefatos lúdicos e atividades desplugadas (CANDIDO et al., 2017) que ajudam a desenvolver o raciocínio lógico através de atividades que refletem o cotidiano, podendo, inclusive, ser utilizadas para ensino de conceitos de computação em locais que tenham pouca ou nenhuma infraestrutura tecnológica.

Com isso, este trabalho propõe o Programinó, um jogo virtualizado (SANTOS; NETO; JUNIOR, 2015) que tem como base o tradicional jogo de dominó, e vem como um facilitador ao aprendizado do conteúdo de tipos de dados na programação. O Programinó traz uma possibilidade de criar um hábito de praticar o conteúdo de tipos de dados, unindo princípios pedagógicos de tipos de dados na programação com o jogo tradicional de dominó. Foi realizado um estudo bibliográfico a respeito dos trabalhos existentes na academia e verificou-se a falta de pesquisas voltadas ao conteúdo de tipos de dados na programação, mostrando ser uma oportunidade de contribuição à sociedade.

1.2 Objetivos

Este trabalho tem como objetivo geral apoiar o aprendizado de tipos de dados, em disciplinas de Introdução a Programação, através de um jogo digital virtualizado baseado no tradicional jogo de dominó.

Para chegarmos ao objetivo geral deste trabalho, foram definidos alguns objetivos específicos:

- Desenvolver um jogo digital para utilização por estudantes de Introdução à Programação;
- Adaptar soluções de Inteligência Artificial (IA) para aplicação no modo *single player* do jogo;
- Validar os níveis de dificuldades desenvolvidos no Programinó através de simulações entre máquinas.

1.3 Estrutura do Trabalho

Este trabalho está organizado em 7 capítulos, descritos abaixo:

- No Capítulo 1, encontra-se a introdução, com a apresentação das motivações, da justificativa e dos objetivos;
- No Capítulo 2, é apresentado um compilado de trabalhos relacionados sobre ensino-aprendizagem de programação;
- O Capítulo 3 está a motivação deste trabalho de forma mais abrangente, explicando a importância de jogos digitais como alternativa para vencer as dificuldades no aprendizado de programação, contém os estudos e explicações necessários para temas como inteligência artificial, busca em profundidade, minimax, entre outros;
- O Capítulo 4 explana o método utilizado neste trabalho para que o Programinó fosse desenvolvido;
- No Capítulo 5 são mostrados os experimentos realizados para validar o algoritmo desenvolvido;
- No Capítulo 6 é apresentada a conclusão deste trabalho; junto com trabalhos futuros e dificuldades encontradas.

2 Trabalhos Relacionados

Neste capítulo serão abordados os estudos relacionados à proposta deste trabalho, que busca na literatura o que as pessoas desenvolvem como alternativa à dificuldades no processo de ensino-aprendizagem no geral e, especificamente, no ensino de tipos de dados na programação.

2.1 Contexto Geral

A indústria de jogos e entretenimento cresce a cada ano. No mundo, somente em 2019, deve movimentar US$ 152 bilhões. No Brasil, este valor chega a US$ 1,5 bilhão por ano[1]. As pessoas permanecem durante longas datas buscando os desafios e fantasias dos jogos. Mas, por que isso também não acontece com jogos educacionais? Procurando responder essa pergunta, pesquisadores e educadores buscam, cada vez mais, estudar e desenvolver jogos educacionais que sejam atrativos e motivacionais, mas sem perder seu viés pedagógico.

(SAVI; ULBRICHT, 2008) cita em seu trabalho muitos benefícios que os jogos educacionais podem trazer, entre eles, temos o efeito motivador, que incentiva o aprendizado por ambientes interativos e dinâmicos; facilitador do aprendizado tornando o estudante um protagonista solucionador de problemas; desenvolvedor de habilidades cognitivas ao ultrapassar desafios e elaborar estratégias; socializador ao ser estimulado à competitividade e colaboração; entre outros. Pensando nisso, jogos educacionais são desenvolvidos em todas as áreas, seja para crianças que ainda não sabem ler, como o Discovery Kids[2], que ensina cores, números e palavras; adolescentes que estudam para o vestibular[3] ou para quem já faz uma graduação de engenharia[4], por exemplo.

2.2 Ensino de Programação

Mais especificamente no ensino de programação, a revisão sistemática de literatura de (AURELIANO; TEDESCO, 2012) traz uma análise de artigos sobre o processo de ensino-aprendizagem de Introdução à Programação, publicados no período

[1] https://www.diariodepernambuco.com.br/noticia/economia/2019/10/industria-de-games-se-profissionaliza-para-ganhar-mercado.html
[2] https://www.discoverykidsplus.com.br/jogos
[3] http://extras.ig.com.br/infograficos/jogoindependencia/
[4] https://ufal.br/ufal/noticias/2019/6/professor-desenvolve-jogo-didatico-para-dimensionamento-asfaltico

de 2002 a 2011, em dois renomados eventos nacionais: Simpósio Brasileiro de Informática na Educação (SBIE) e o Workshop de Informática na Escola (WIE). A análise mostrou que a quantidade de artigos na área cresceu significativamente a partir de 2009, indicando o crescente interesse de pesquisadores sobre o tema, desde então. Além disso, mostrou que a maior parte das pesquisas aconteceu nas regiões Sul e Sudeste com estudantes de ensino superior.

Por outro lado, a revisão sistemática de (MEDEIROS; SILVA, 2013) mostra que pesquisas sobre jogos digitais no ensino de programação estão focando em estudantes de ensino médio. Os autores destacam que tais jogos proporcionam elementos motivadores para o processo de ensino-aprendizagem de programação, trazendo o ato de jogar para o contexto educacional. Os estudos selecionados apresentam, em sua maioria, oficinas e experimentos em escolas públicas, como por exemplo, os estudantes criam jogos e robôs sobre quaisquer temas, mas no processo de criação eles aprendem programação. Entre os resultados, os estudos apontam que o ensino de programação deve começar no ensino médio, mas também indicam que o uso de jogos para o ensino de programação é uma alternativa de aprendizado para universitários, para que o grande número de reprovações diminua. Por fim, verificou-se que o ensino de programação com jogos digitais foi muito bem recebido entre os os estudantes de escolas públicas e graduação.

Para entender as dificuldades enfrentadas no ensino de programação é preciso que entendamos, também, quais as atividades desenvolvidas, quem é o público alvo e suas respectivas limitações. Em sua revisão sistemática (SOUZA; BATISTA; BARBOSA, 2016) apresentam dificuldades comuns principalmente para os iniciantes, relacionadas a abstração e compreensão dos problemas, tratamento de erros, especificação e codificação de soluções. Os resultados (Figura 1) mostram que as dificuldades vão desde entender conceitos de programação (presentes em 39% dos artigos analisados) até a aplicação deles em busca de resolver problemas (24% dos artigos). Em 4 anos (2010-2014), esses dois problemas citados estavam presentes em mais da metade dos artigos analisados pelos pesquisadores; por outro lado, essas pesquisas também indicam uma crescente tendência em buscar inovações metodológicas e ferramentais para contornar essas limitações.

Outros dados bastante chamativos da pesquisa são que 78,5% dos trabalhos que apresentaram problemas motivacionais dos estudantes também apresentaram problemas na aprendizagem; e 77,4% dos trabalhos que citam dificuldades em aplicar os conceitos relacionam tais dificuldades com o processo de aprendizagem deles (Figura 2), ou seja, aparentemente, como esperado, há uma correlação entre a dificuldade de aprender o conceito e aplicá-lo. Os autores expuseram também que a quantidade de trabalhos que citaram o problema de assimilação dos conceitos foi a maior

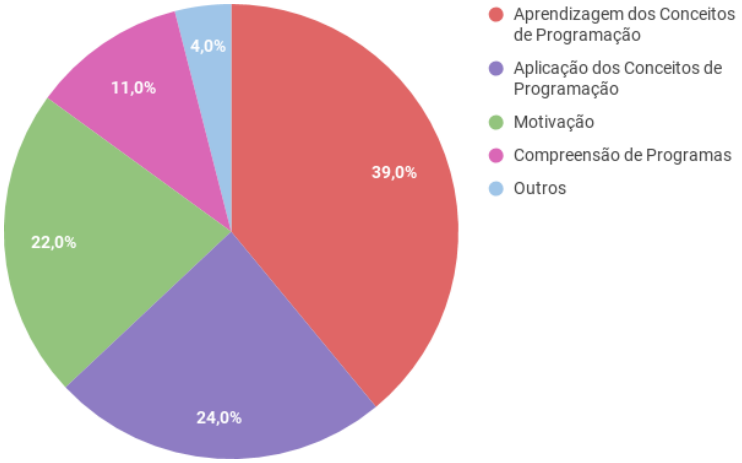

Figura 1 – Porcentagem de Artigos Classificados em cada Categoria de Problemas

Fonte: (SOUZA; BATISTA; BARBOSA, 2016)

durante 3 anos, igualando com motivação apenas em 2014. Além disso, foi possível visualizar as soluções relatadas, como por exemplo, que o uso de jogos pode ajudar na concentração e diminuir o número de desistência em disciplinas de programação como um aliado importante, ficando atrás apenas da utilização de visualização de programas e algoritmos (PRICE; BAECKER; SMALL, 1998).

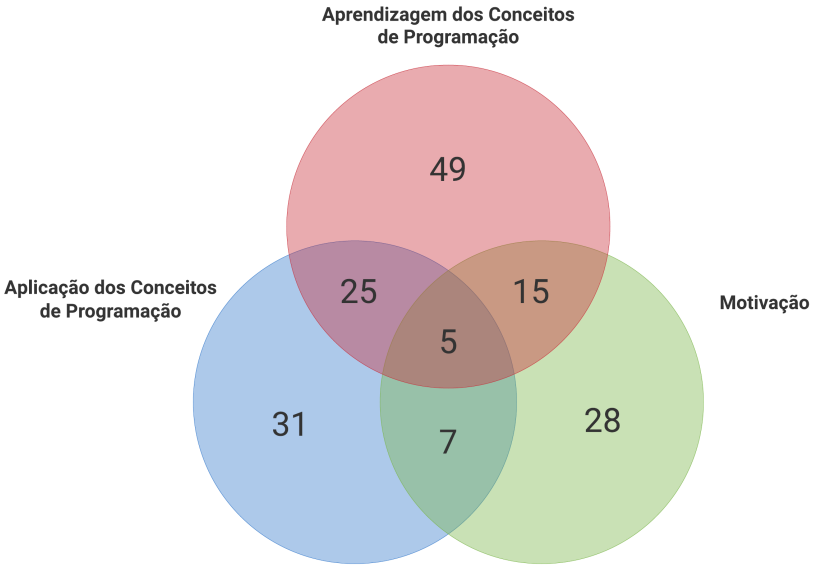

Figura 2 – Interseção entre as Categorias de Problemas

Fonte: (SOUZA; BATISTA; BARBOSA, 2016)

(MEDEIROS, 2019) realizou uma revisão sistemática da literatura sobre dificuldades que iniciantes enfrentam na aprendizagem de programação. Um dos problemas observados pelo autor, quando iniciantes começam a programar, é a dificuldade em traduzir determinada tarefa em linguagem de programação, um problema conhecido como pragmática da programação; onde o estudante iniciante não consegue olhar para

o problema como um todo e pensar em como resolvê-lo, partindo diretamente para a resolução linha a linha. Por pular a parte do entendimento, aparece, em seguida, um novo problema: saber quais instruções o computador precisa.

Na pesquisa de (CANDIDO et al., 2017), os autores utilizaram o jogo digital *Lightbot*[5] e atividades de simulação corporal em duas turmas da educação básica, aplicando em ordem diferentes (a primeira turma jogou o *Lightbot* e em seguida fez a simulação corporal e a segunda turma fez na ordem inversa), para analisar se a ordem interferiria no nível de aprendizado do assunto de funções na programação. Embora os resultados tenham sido inconclusivos, a pesquisa mostrou um aumento eficaz de conhecimento em todas as turmas, disponibilizando mais uma alternativa inovadora para educadores de programação.

Mais relacionado à este trabalho, a pesquisa de (SILVA, 2018) apresenta a virtualização de um jogo físico que aborda operações básicas matemáticas, conhecido como Cubra Doze. No processo de virtualização, um jogo físico é transformado em formato digital, permanecendo com sua essência pedagógica, e agregando os benefícios da interatividade (SANTOS; NETO; JUNIOR, 2015). Na virtualização do Cubra Doze, foram preservados conceitos abordados no jogo físico e acrescentados alguns níveis de dificuldade, além de elementos de design e interação com o usuário, no intuito de aumentar o engajamento de alunos de ensino fundamental e médio. O jogo foi desenvolvido no Ionic[6] e, por conta do público participante da pesquisa (todos do 1º ano do ensino médio de uma escola técnica estadual) utilizar *tablets* na escola, foi disponibilizado através do navegador Google Chrome no sistema operacional Windows. A pesquisa mostrou que os educandos participantes dos testes se sentiram motivados e tiveram interesse na utilização de atividades lúdicas computacionais, assim como em praticar determinados assuntos através de jogos. Por fim, a pesquisa conclui que ferramentas lúdicas educacionais podem e devem ser inseridas no contexto da aprendizagem.

Uma outra abordagem, também relacionada a este trabalho, no trabalho de (FREITAS et al., 2018), realizou-se um estudo qualitativo sobre um jogo digital para o auxílio no ensino de tipos de dados na programação. Este estudo foi dividido em duas fases: a primeira foi o desenvolvimento do jogo Invade The System[7], usando a *engine* Unity[8] 3D. No jogo, o jogador deve relacionar os tipos de dados com afirmações cotidianas, com o objetivo de acertar o maior número de correlações no menor tempo possível. Por exemplo: o jogador deve ligar a frase “Henrique está contando quantos centavos faltam para comprar uma ficha de sinuca” ao tipo de dado *Float*,

[5] https://lightbot.com/
[6] https://ionicframework.com/
[7] https://github.com/paulorcmendes/InvadeTheSystem/releases
[8] https://unity.com/pt

pois os centavos são números decimais. A segunda fase foi uma avaliação qualitativa com a aplicação deste jogo em uma turma de 27 alunos de primeiro período de graduação, em uma disciplina de introdução à programação. Após o uso, os estudantes responderam perguntas de forma a concordar ou discordar de afirmações, como: “Eu precisei aprender poucas coisas para poder começar a jogar o jogo” ou “O jogo é um método de ensino adequado para esta disciplina”. Através da análise dessas respostas, os autores concluíram que o jogo apresentado trouxe motivação o suficiente para que os estudantes se engajassem, e que os estudantes conseguiram identificar o assunto abordado na disciplina. Por outro lado, foi um consenso que o jogo não substitui a aula.

Já no trabalho de (OLIVEIRA; FARIAS, 2018), foi desenvolvido um jogo educacional 2D chamado de projetoÉden abordando os assuntos de variáveis, constantes e tipos de dados. Ele tem por objetivo facilitar o aprendizado dos conteúdos em disciplinas introdutórias de programação de cursos técnicos e graduações. O jogo, desenvolvido através da *engine* Construct2[9], é do gênero plataforma e demanda que o jogador utilize comandos envolvendo codificação para cumprir missões, passando por uma narrativa de estratégias. É possível visualizar o aprendizado quando o jogador interage com uma tela de codificação contendo, por exemplo, uma missão para matar uma lesma através da declaração de variável correta com sua nomenclatura, tipo e valor. O projeto prevê o uso deste jogo como uma ferramenta complementar ao ensino. Como trabalhos futuros, o autor deseja validá-lo em turmas de um curso técnico, visando observar o grau de motivação, experiência e o ganho de aprendizagem.

2.3 Conclusões

Analisando os trabalhos relacionados, percebe-se que existe um crescente interesse em soluções digitais que apoiem o ensino e aprendizagem de programação. Em particular, o uso de jogos eletrônicos é uma alternativa que tem sido bem aceita, pela motivação e engajamento que gera nos estudantes. Entretanto, embora a inserção do ensino de programação nas escolas seja uma tendência mundial, no Brasil ainda prevalecem os estudos voltados para a graduação (SILVA et al., 2015). A proposta deste trabalho é desenvolver um jogo digital, baseado na mecânica do jogo físico de dominó, que auxilie a aprendizagem de tipos de dados de estudantes iniciantes.

[9] https://www.construct.net/en

3 Referencial Teórico

Neste capítulo, será apresentada uma visão geral sobre as teorias e técnicas da computação utilizadas neste trabalho. Com uma visão geral das dificuldades no processo de ensino-aprendizagem e da área de Inteligência Artificial e suas aplicações na resolução de jogos, na seção 3.2. Em seguida, serão explicadas as teorias e tipos dos algoritmos de busca e sua integração com a Teoria de Jogos, com a descrição detalhada do principal algoritmo utilizado neste trabalho, o Minimax, na seção 3.3.

3.1 Processo de Aprendizagem

3.1.1 Processo Educacional

Para se manter relevante, o processo educacional precisa se atualizar constantemente, onde novas formas de trabalhar o conteúdo são indispensáveis de serem adotadas. Com isso, a reflexão e a observação de como o ser humano aprende e como deve ser avaliada sua inteligência são fundamentais.

(GARDNER, 1995) fala que todo ser humano possui múltiplas inteligências e cada uma delas pode ser evoluída ou enfraquecida de acordo com as práticas diárias. Entre as múltiplas inteligências, é possível citar 7 tipos: Inteligência verbal ou linguística (facilidade em usar a fala e a escrita), Inteligência lógico-matemática (facilidade em trabalhar com números, lógica e reconhecer padrões e conexões de conteúdos), Inteligência musical (permite a organização de sons de forma criativa), Inteligência visual ou espacial (preferem aprender através de recursos visuais como ilustrações, gráficos, mapas, etc), Inteligência corporal ou cinestésica (habilidade de usar o próprio corpo para resolver problemas - cirurgiões, malabaristas, etc.), Inteligência interpessoal (comunicar-se bem, verbalmente ou não), Inteligência intrapessoal (reconhecer os próprios limites, medos e aspirações para utilizá-los a seu favor). O autor desenvolveu sua teoria baseada em estudos com diversas pessoas e comprovou que o ensino pode ser melhorado se considerar esses diferentes tipos de inteligência na individualidade de cada aluno.

Levando em consideração essa teoria, entende-se que da mesma forma que existem muitas maneiras do ser humano se expressar, provavelmente também há inúmeras formas de adquirir conhecimento e de avaliá-lo. Por fim, é possível considerar habilidades intelectuais como algo mais amplo, ou seja, desenhar, dançar, atuar, ouvir música, cantar, jogar etc, se tornam tão importantes quanto escrever e realizar cálculos matemáticos. Com isso, o principal desafio da educação é perceber essas inteligências

e diferenças intelectuais dos alunos para poder desenvolvê-los.

3.1.2 Dificuldades de Aprendizagem no Geral

Dentro do contexto geral no processo de ensino-aprendizagem, entender dificuldades no aprendizado é uma tarefa árdua, tendo em vista que elas não começam na graduação, nem são algo exclusivo da computação. É preciso considerar que para além de um novo assunto, o estudante tem vida pessoal, familiar e profissional para gerenciar. O problema não é o fato dessas dificuldades existirem, são as consequências que elas provocam: desistência da disciplina, escola, curso e até mesmo desmotivação de vida causada por frustrações e questões de saúde mental (XENOS; PIERRAKEAS; PINTELAS, 2002).

Manter o equilíbrio do nível de conhecimento entre os estudantes também não é uma tarefa fácil, tendo em vista que cada um tem o seu histórico e ritmo de aprendizado, principalmente se tratando de alunos do ensino médio, muitos dos quais não têm fácil acesso a internet e computador (CHAVES; BEZERRA et al., 2019).

Um outro ponto interessante para se observar é como, e se, o diálogo acontece nas disciplinas entre os alunos ou com o professor. (GIRAFFA; MORA, 2013) consideram que muitos dos formatos atuais de aula não favorecem essa comunicação, diminuindo a eficiência na construção do conteúdo e, consequentemente, desmotivando alunos e professores, podendo levar, inclusive, à desistência dos alunos e evasão dos cursos.

Ainda falando sobre relação aula, professor e aluno, existe uma preocupação quanto ao método de ensino: tendo em vista os inúmeros estímulos em que os estudantes estão expostos (redes sociais, mensagens instantâneas, noticiário de fácil acesso, etc.), o método tradicional de aula onde o professor é o provedor exclusivo do conhecimento deixou de ser efetivo. Entretanto, muitos professores não parecem estar preocupados por acreditarem que é natural que cursos de programação tenham taxas elevadas de evasão (BENNEDSEN; CASPERSEN, 2007).

3.1.3 Dificuldades na Aprendizagem de Programação

Analisando o contato dos estudantes com a tecnologia, percebe-se que muitas vezes, se resume apenas ao consumo, mas quando desenvolvida, a Ciência da Computação consegue despertar habilidades do pensamento crítico e computacional, mostrando aos alunos que eles também podem criar tecnologias, ao invés de apenas usá-las. Nesse contexto, a literatura comprova que a atividade de programar é complexa, e o ensino de programação é particularmente desafiador e lento, pois, lidar com

as abstrações da programação não é uma atividade trivial, principalmente para iniciantes (SILVA; MEDEIROS; ARANHA, 2014).

Os conceitos de programação (variáveis, estruturas de controle, repetição, seleção, condicionais, tipos de dados, estrutura de dados, funções, parâmetros, listas) também são amplamente citados quando se fala de dificuldades. Onde mais uma vez, iniciantes, pela sede em programar algo, pulam tais conceitos - que por si já não são triviais, tendo em vista a não habitual abordagem desses assuntos - assim como acontece no português, matemática, física, etc., para irem direto aprender uma linguagem de programação. Um dos problemas citados por (LAHTINEN; ALA-MUTKA; JÄRVINEN, 2005), em uma pesquisa feita com professores e alunos ligados com o ensino-aprendizagem de programação, foi a dificuldade em entender as abstrações necessárias para as declarações de variáveis e seus respectivos tipos de dados. Tal dificuldade vai desde entender qual nome adequado, de acordo com padrões de nomenclatura da variável, até qual valor será armazenado para então decidir seu tipo; tal problema é onde o presente trabalho busca auxiliar diretamente, facilitando o relacionamento de valores aos seus respectivos tipos de dados.

Dadas todas as dificuldades citadas acima, o uso de ferramentas lúdicas que venham a auxiliar no aprendizado são necessárias e estão sendo muito discutidas no Brasil e no mundo, pois, mesmo que estudantes ainda apresentem dificuldades para resolver problemas de programação utilizando novas ferramentas de ensino-aprendizagem, pesquisas revelam que o uso de jogos, por exemplo, estimula a motivação durante os desafios propostos, uma maior retenção da atenção e o despertar do interesse na área (PIRES; PRATES, 2019).

A literatura dá alguns indícios de que pesquisadores buscam contornar esses problemas propondo o ensino de programação ainda no ensino fundamental. Entretanto, isso ainda não é uma realidade nos currículos oficiais de escolas públicas brasileiras (LIMA; VIEIRA; BRANDÃO, 2019), contando atualmente com iniciativas extracurriculares como Pernambucoders[1], Code.Org[2], Programaê[3], PyLadies[4], Programa, essa menina![5], CinTIa[6] e M.I.N.A.S[7], que buscam tornar esse acesso ao desenvolvimento da programação ainda mais fácil para todos.

[1] https://www.cesar.org.br/index.php/portfolio/pernambucoders
[2] https://code.org/international/about
[3] http://programae.org.br/
[4] http://brasil.pyladies.com/about/
[5] https://sites.google.com/view/programaessamenina/
[6] https://sites.google.com/cin.ufpe.br/cintia/quem-somos
[7] https://www.instagram.com/portodigitalminas/

3.2 Inteligência Artificial e Jogos

Não é por acaso que os seres humanos são conhecidos como *homo sapiens*, a inteligência proveniente desta classe é muito importante para nós, pois, ela quem nos faz raciocinar, compreender e tomar decisões. A Inteligência Artificial (IA) é uma área, que como o próprio nome já diz, estuda formas de desenvolver computadores capazes de realizar tarefas de forma independente e inteligente. A área vem crescendo muito, mostrando que é possível que a máquina opere sem a ajuda do ser humano, em tarefas rotineiras, como em carros autônomos[8], por exemplo. Artefato este, também chamado de agente, que age de forma inteligente e deve ser capaz de não apenas executar uma tarefa específica, mas também de ter a capacidade de perceber seu ambiente através de sensores e atuadores, ter controle autônomo, adaptar-se a mudanças e tomar decisões (NORVIG; RUSSELL, 2014).

Os primeiros experimentos científicos com o objetivo de desenvolver máquinas inteligentes ocorreram em meados da década de 50 (MAŃDZIUK, 2007). Entre os pioneiros, estão Claude Shannon, conhecido como o "pai da Teoria da Informação", com o seu trabalho *Programming a computer for playing chess* (1950), e Alan Turing, "pai da computação", com o seu artigo *Digital computers applied to games* (1953), em que aborda busca em árvore para jogos. A partir de então, jogos populares de tabuleiro, como Damas, Xadrez, Poker, Go e Dominó, são objetos de pesquisas em Inteligência Artificial, com o principal objetivo de superar o desempenho humano.

Sendo assim, a aplicação em jogos foi uma das primeiras iniciativas experimentadas em Inteligência Artificial (NORVIG; RUSSELL, 2014) e, ao longo da história da IA, tem se apresentado como um tópico interessante, rendendo grandes avanços no desempenho em jogos clássicos de tabuleiro nas últimas décadas (MAŃDZIUK, 2008). Porém, segundo Mańdziuk, ainda existem alguns pontos, na área de IA aplicada aos jogos, que demandam atenção e melhorias, sendo eles:

- A melhoria do desempenho do agente baseado na experiência obtida através de rodadas/jogadas anteriores;
- O desenvolvimento de agentes com a capacidade de descoberta de conhecimento a fim de criar novas estratégias de jogo;
- A implementação efetiva de agentes que simulem, de fato, a intuição humana.

De forma geral, segundo Mańdziuk, o objetivo principal do uso de técnicas de Inteligência Artificial em jogos é a concepção de agentes jogadores de vários jogos, verdadeiramente autônomos, como o ser humano.

[8] https://exame.abril.com.br/negocios/arm-faz-parceria-com-gm-e-toyota-para-criar-sistemas-de-carros-autonomos/

De acordo com as características e regras de cada jogo, existem técnicas específicas da Inteligência Artificial que são mais indicadas para atingir um melhor desempenho. Por exemplo, levando em consideração a classe dos jogos com dois jogadores, em que um faz o primeiro movimento, seguido do outro jogador, e assim se revezam até o término do jogo, (NORVIG; RUSSELL, 2014) definem este tipo de jogo como um problema de busca, pois apresentam os seguintes componentes:

I. **Estado Inicial** - que estabelece o jogador que fará o primeiro movimento e a posição das peças no tabuleiro;

II. **Função Sucessor** - que retorna uma lista de ações (jogadas) válidas e o respectivo estado resultante de cada ação escolhida;

III. **Teste de Término** - que indica quando não há mais ações possíveis, ou seja, o jogo finalizou;

IV. **Função Utilidade** - que retorna o resultado final do jogo, podendo ser vitória, derrota ou empate.

Como o Dominó e, consequentemente, o Programinó, se enquadra como um jogo deste tipo de problema, os Algoritmos de Busca são amplamente indicados e utilizados no desenvolvimento de agentes inteligentes para a sua solução. Na seção seguinte, será explicado o funcionamento dos Algoritmos de Busca em geral e detalhado o algoritmo utilizado neste trabalho, o Minimax.

3.3 Algoritmos de Busca

Nesta seção, são explanados os principais conceitos dos algoritmos de busca, clássicos na aplicação em jogos. Na subseção 3.3.1, é feita uma explicação da lógica básica desta classe de algoritmos. Já na subseção 3.3.2, explica-se a contribuição da Teoria dos Jogos para a evolução dos algoritmos de busca. E, por fim, o algoritmo Minimax é descrito na subseção 3.3.3.

3.3.1 Busca Clássica

Na solução de muitos problemas de Inteligência Artificial, pode ser necessária e relevante a busca, no universo de estados possíveis do problema, do seu específico estado objetivo, por parte do agente inteligente. Isto é, uma vez fornecido o modelo completo do universo do problema para o agente, este pode realizar uma busca por uma sequência de ações que acabem levando ao estado final desejado, antes mesmo da necessidade de efetuar sua ação atual (BONET; GEFFNER, 2001).

No ambiente dos jogos, significa que o agente inteligente, em posse do espaço de todos os estados possíveis do jogo, pode decidir, com base em um objetivo (normalmente a vitória) qual deve ser sua próxima jogada.

Com relação aos tipos de buscas, os mais clássicos são os representáveis por grafos, onde o agente empenha-se em encontrar um caminho condizente com a especifidade do problema ao qual foi projetado para resolver, como buscar o caminho menos custoso, mais rápido ou mais curto, para um nó objetivo.

A depender da estrutura de dados que armazena os estados possíveis do problema, o tipo de busca também pode mudar. Por exemplo, as árvores são uma categoria especial de grafos, em que cada nó pode ter 1 ou mais filhos, porém todos os nós só podem ter necessariamente um pai. Uma especifidade das próprias árvores são as Árvores Binárias, em que existe um limite de 2 filhos por nó. Para qualquer tipo de grafo, existem dois algoritmos de busca bastante comuns, são as buscas em profundidade e em largura. Na primeira, a busca percorre o máximo possível em cada ramo antes de retornar ao nó original. Já na busca em largura, essa exaustão é feita por níveis, que pode ser todos os nós filhos de um nó de origem e em sequência todos os nós filhos dos filhos do nó de origem.

Na busca em profundidade, o algoritmo considera várias sequências de ações possíveis em uma árvore para encontrar uma solução. A árvore é composta pela raiz (estado inicial), ramos (ações) e nós (estados do problema). Conforme ilustra a Figura 3, o seu funcionamento começa no nó raiz (A) e explora, o quanto possível, cada um dos seus ramos (nó B, em seguida D) até o nó folha (nó H, não possui filhos) antes de retroceder e acessar um novo ramo (I). Isso se repete sucessivamente.

Mesmo não sendo um algoritmo cuja solução seja ótima, de complexidade tempo/espaço exponencial, tendo em vista os caminhos redundantes, este apresenta vantagens no quesito de armazenamento, precisando armazenar apenas um único caminho da raiz até um nó folha. A partir do momento que este nó é expandido e todos os seus descendentes foram explorados, ele pode ser excluído da memória. Tal fato faz com que essa busca seja muito usada na IA (NORVIG; RUSSELL, 2014).

As buscas explicadas até agora levam em consideração apenas os estados visíveis e as jogadas do próprio agente inteligente. Ou seja, no contexto de 2 jogadores existem buscas mais específicas. Uma adaptação da busca em profundidade para este tipo de problema será explanada na próxima seção.

3.3.2 Busca Competitiva e Teoria dos Jogos

Quando falamos de jogos, a grande maioria destes, possuem como formato a modalidade com múltiplos jogadores, inclusive os Jogos Digitais, em que o “outro joga-

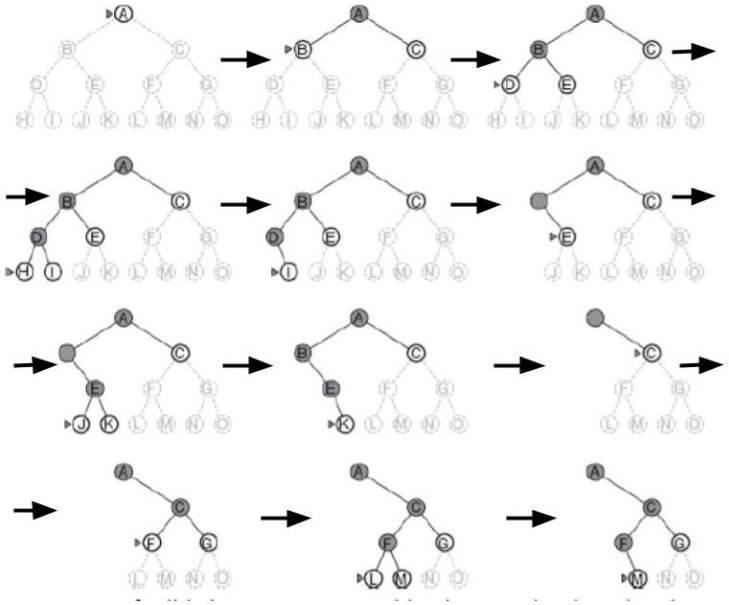

Figura 3 – Ilustração de uma árvore de busca em profundidade

Fonte: Adaptação de NORVIG, RUSSELL, (2014)

dor" não necessariamente é um humano, podendo ser uma máquina. Nestes cenários, a busca clássica, discutida na seção anterior, não consegue modelar o comportamento dos demais jogadores. Assim, para se adaptar a esta nova visão, os algoritmos de buscas foram integrados a uma área classicamente conhecida por considerar a interação entre agentes concorrentes, a Teoria dos Jogos.

A Teoria dos Jogos é uma ferramenta poderosa para analisar situações nas quais as ações de vários agentes afetam a decisão do seguinte, lidando assim, com problemas de otimização interativa. John von Neumann e Oskar Morgenstern são formalmente creditados como os pais da Teoria dos Jogos moderna, através do clássico livro *Theory of games and economic behavior* (1944), que resume os conceitos básicos existentes da época. Desde então, a Teoria dos Jogos teve uma explosão de desenvolvimentos e aplicações nas mais diversas áreas, tendo maior relevância em Economia e Matemática.

Na Inteligência Artificial, a Teoria dos Jogos tem uma relevante contribuição nos algoritmos de sistemas multiagentes. Particularmente, para este trabalho e para os algoritmos de Busca Competitiva, as maiores contribuições desta área de estudo são alguns de seus principais conceitos, sendo eles: os Jogos de Soma-Zero, Jogos de Perfeita e Imperfeita Informação e o Equilíbrio de Nash.

Jogos de Soma-Zero são aqueles em que a vitória de um jogador significa, necessariamente, a derrota de outro. De maneira mais específica, a soma dos ganhos e perdas dos jogadores deve sempre resultar em zero. No jogo do Dominó, por exemplo, existe um vencedor, um perdedor ou um empate. Sendo assim, considerando a vitória

com valor $+1$, a derrota com valor -1 e o empate como 0, tem-se que ao final a soma dos resultados será 0.

Nos jogos de perfeita informação todos os jogadores conhecem todos os movimentos disponíveis dos demais jogadores. Já no de informação imperfeita, ao menos um jogador desconhece o movimento dos outros jogadores (THOMAS, 2012). Por exemplo, o xadrez é um jogo de perfeita informação, visto que todos os jogadores sabem o que tem no tabuleiro e o que cada peça é capaz de fazer. Já o dominó é o contrário, pois, os jogadores não sabem a peça que seus oponentes têm.

O Equilíbrio de Nash (NASH et al., 1950) estabelece um conceito de solução, ou seja, uma previsão de como o jogo será jogado. No contexto de jogos entre 2 jogadores, ambos jogam a melhor resposta em relação à jogada do outro. Sendo assim, ninguém pode melhorar nada unilateralmente, mas pode ser que melhore mudando juntos.

Na próxima seção, será explicado o conceito e funcionamento do algoritmo utilizado neste trabalho, o Minimax, e como ele realiza sua busca competitiva aplicando conceitos da Teoria dos Jogos.

3.3.3 Algoritmo Minimax

O algoritmo Minimax é um clássico de busca competitiva, que realiza a busca em profundidade em árvores não-binárias, tendo como inspiração os conceitos citados na seção anterior da Teoria dos Jogos. O seu princípio é percorrer os nós da árvore até chegar no fim do jogo e identificar, de forma antecipada, se o jogador perdeu, empatou ou ganhou. Dessa forma, ele explora todas as possíveis ações dos jogadores a partir do estado atual. É utilizada uma recursão simples (Figura 4) dos valores MIN e MAX e calculada a melhor decisão no momento, levando em consideração que o adversário sempre vai escolher a melhor peça.

Um oponente é o MIN (jogador adversário) e o outro é o MAX (jogador principal), eles jogam alternadamente e ambos têm uma árvore de números em comum, equivalente à árvore de movimentos. Cada jogador considera a melhor jogada para si em cada rodada, fazendo com que cada jogador maximize suas chances de ganhar (maximizando o *score* no pior caso).

De forma geral, o agente percebe o ambiente no qual está inserido, e simula as possíveis jogadas futuras (dele e do oponente) para escolher a sua. Ou seja, o agente tem acesso ao estado completo do ambiente atual. Dentro dessa árvore das jogadas simuladas, o MIN tem que achar a melhor jogada a partir da raiz e decidir qual o ramo ele deve seguir com o menor valor. Já o MAX tem que fazer o contrário.

Conforme ilustra a Figura 5, este é um exemplo de uma árvore construída no jogo da velha com tantos ramos quanto o número de ações possíveis. Observa-se que

```
function MINIMAX-DECISION(state) returns an action
  return arg max a ∈ ACTIONS(s) MIN-VALUE(RESULT(state, a))

function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
      v ← MAX(v, MIN-VALUE(RESULT(state, a)))
  return v

function MIN-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← ∞
  for each a in ACTIONS(state) do
      v ← MIN(v, MAX-VALUE(RESULT(state, a)))
  return v
```

Figura 4 – Pseudocódigo do algoritmo minimax

Fonte: (NORVIG; RUSSELL, 2014)

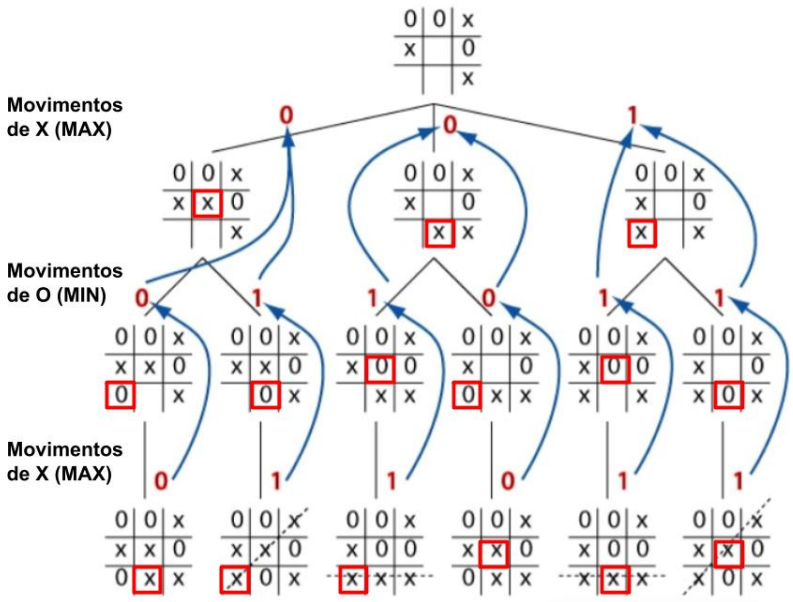

Figura 5 – Exemplo de uma árvore gerada a partir de um estado do Jogo da Velha

Fonte: Adaptação de ORGÂNICA NATURAL MARKETING (2019)

no estado atual do jogo (raiz), existem 3 espaços vazios, logo, os ramos posteriores serão o preenchimento de cada um desses espaços para encontrar a escolha ideal. O retorno dos valores guardados depende do status da jogada (menos 1 quando a máquina perde, 0 quando empata e mais um quando ganha) e acontece de baixo pra cima, onde o MAX (neste caso, movimento do “X”) retorna sempre o maior valor e o MIN (neste caso, movimento do “O”) o menor. Ainda neste exemplo, é possível verificar que o “X” deve seguir por qualquer opção da direita, tendo em vista que ele ganha em

todas.

Particularmente, para este trabalho e para os algoritmos de Busca Competitiva, as maiores contribuições desta área de estudo são alguns de seus principais conceitos, sendo eles: os Jogos de Soma-Zero, o Equilíbrio de Nash e Jogos de Perfeita e Imperfeita Informação.

3.4 Conclusões

Com base nos conceitos discutidos acima, pode-se concluir que o Programinó apresenta uma natureza comum à aplicação da Busca Competitiva do Minimax, com algoritmos de busca e Teoria dos Jogos, pois, assim como o Dominó, é um jogo de Soma-Zero e que pode ter o Equilíbrio de Nash aplicado.

Assim como o Programinó, muitas outras iniciativas utilizam de algoritmos de Inteligência Artificial para o ensino-aprendizagem através de jogos digitais. Por exemplo, (AMARAL; MACHADO; BRAGA, 2013) utilizou o algoritmo Minimax para dar dicas para o jogador aprender sobre administração rural. Um outro exemplo, inclusive clássico, de uma adaptação do Minimax é em um jogo de xadrez, onde a máquina conhecida como *Deep Blue* foi a primeira IA capaz de vencer um campeão humano (CAMPBELL; JR; HSU, 2002).

4 Método de Desenvolvimento

Neste capítulo será explanado o método aplicado na construção do Programinó. Na seção 4.1, é explicado como surgiu a ideia do jogo. A seção 4.2 conta com um resumo das regras do Dominó levadas em consideração neste trabalho. A seção 4.3 elucida o potencial auxílio pedagógico do Programinó no ensino de programação. A seção 4.4 detalha todo o processo de desenvolvimento e implementação do jogo Programinó.

4.1 Surgimento do Programinó

A ideia do Programinó (Figura 6) surgiu durante a aplicação de uma atividade de computação desplugada (ensino de computação sem a necessidade do computado) para o aprendizado do conteúdo de tipos de dados na programação, na disciplina de Lógica e Pensamento Computacional, lecionada no curso técnico em Desenvolvimento de Sistemas da Escola Técnica Estadual Porto Digital (ETE PD).

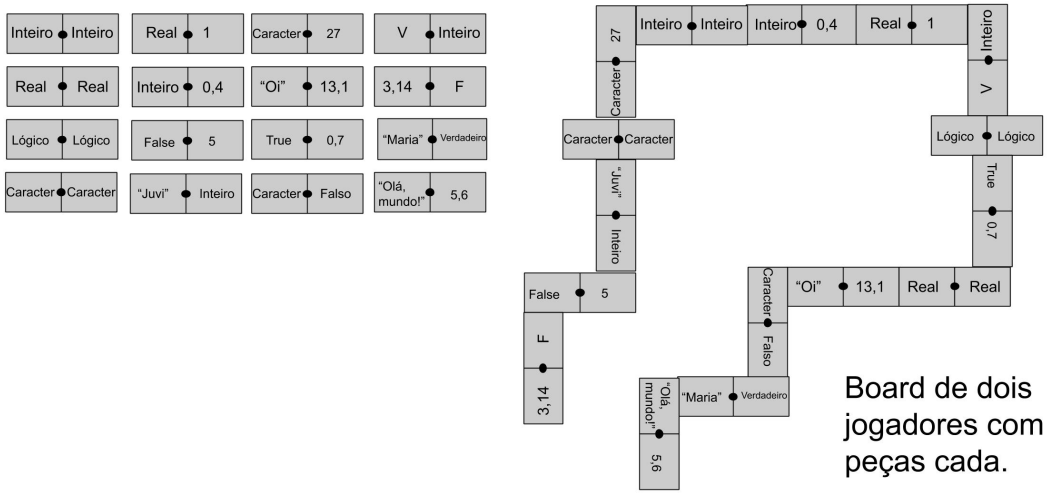

Figura 6 – Partida de um jogo de Programinó

Durante o desenvolvimento da atividade, que solicitava um artefato lúdico físico que trabalhasse o conceito de tipos de dados na programação, os estudantes criaram um jogo com materiais reciclados seguindo o mesmo design de um dominó tradicional, conforme a Figura 7. Após o engajamento visto durante a atividade aplicada em sala de aula, decidiu-se por virtualizá-lo, neste trabalho, para que os estudantes (ou qualquer estudante iniciante em programação) pudessem jogar contra a máquina, exercitando o conteúdo mesmo quando não houvesse colegas, aumentando as possibilidades de uso. Além disso, seria possível que mais estudantes tivessem acesso, sem as limita-

ções da necessidade de estar presente fisicamente. Após pesquisa na literatura, foram encontrados jogos digitais que abordassem o conteúdo de tipos de dados (FREITAS et al., 2018) e (OLIVEIRA; FARIAS, 2018), mas nenhum como o proposto neste trabalho, integrando tipos de dados à mecânica de um jogo tradicional de dominó.

Figura 7 – Jogo de dominó com tipos de dados feito por alunos da ETE PD

4.2 Regras do Dominó

Jogado em diversos países do mundo, o jogo de Dominó (Figura 8) tornou-se popular e tradicional principalmente na América Latina. No Brasil, o jogo foi trazido pelos portugueses no século XVI. Porém, não existe consenso com relação a sua origem. Sua primeira menção é datada da China no século XI, onde teria sido inventado por um funcionário do imperador Hui Tsung.

Com origens diversas e adaptações a cada cultura em que foi inserido, o jogo apresenta inúmeras regras e formatos. Porém, a versão mais popular no ocidente é o duplo-seis (COSTA, 2016), cujas regras foram escolhidas como base para este trabalho. Nesta forma, o conjunto do jogo contém 28 peças, ou pedras.

O jogo é formado por 4 jogadores individuais, podendo-se também jogar entre duplas (4 jogadores, 2 contra 2). O jogo contém 28 peças divididas em duas partes conhecidas como ”pontas”ou ”cabeças”que são marcadas por números de 1 a 6, ou deixadas em branco. Para o jogo, as peças são embaralhadas e distribuídas uniformemente, sendo 7 ou 6 peças para cada jogador (na opção com 6, 4 peças ficam fora do jogo e só podem ser reveladas ao final do jogo, chamado de ”dorme”). O jogo começa com a peça de valor mais alto (geralmente, a que contém o número 6 em cada “cabeça” da peça do dominó) e segue no sentido-horário até que o jogo acabe, ou seja,

Figura 8 – Partida de um jogo de Dominó[1]

Fonte: Foto de Fares Hamouche em Unsplash

até que alguém não tenha mais nenhuma peça nas mãos ou até que não haja mais movimentos válidos. O objetivo é colocar todas as peças na mesa primeiro e ganhar pontos. Caso algum jogador não tenha alguma peça durante as rodadas, ele passa a vez para o próximo jogador.

4.3 Base Pedagógica

Santos e Alves (2000) realizou uma análise da aplicação de 3 tipos de dominós diferentes e observou que, mesmo para quem nunca tivesse jogado algum dominó, o erro por tipo de dominó era inferior a 12% do total de jogadas possíveis, mostrando que os saberes exigidos para tal ação são bastante acessíveis. Isso indica que o dominó é um bom aliado para ensino de conceitos em diversas áreas. Assim, ter o dominó como base para o Programinó não prejudica quem não sabe os conceitos de tipos de dados e nunca jogou dominó.

(PIAGET, 2004) classificou os jogos em 3 categorias: jogos de exercício, simbólicos e de regra. Os jogos de regras, como no caso do dominó e do Programinó, envolvem o seguimento obrigatório de regras e trabalham o aprendizado pela repetição (*drill and practice*) como recurso de aprendizagem para criação de hábitos. No Programinó, a aprendizagem pela repetição foi pensada como uma forma de revisão do que foi ensinado em sala de aula, ou até para praticar antes mesmo da exposição

[1] https://unsplash.com/photos/UMNVul8jnFk

do conteúdo (sala de aula invertida[2]). Outra abordagem escolhida para o Programinó foi o *scaffolding*, que (Wood, 1976 apud Teixeira, 2015) define da seguinte forma:

> "Scaffolding refere-se a uma variedade de técnicas de instrução usadas para mover os alunos progressivamente em direção a uma maior compreensão, em última análise, uma maior independência no processo de aprendizagem."

Com isso, estudantes obtêm ajuda apenas quando há necessidade de uma nova habilidade, exatamente como um "impulso" a continuar aprendendo. Da mesma forma acontece no Programinó, pois, inicialmente as jogadas acontecem baseadas nas movimentações válidas. Ou seja, na primeira fase o jogador não vai errar, pois não será possível escolher uma peça incorreta para posicionar no tabuleiro.

4.4 Implementação do Programinó

Para o desenvolvimento do Programinó foi utilizada a linguagem Python (versão 3.7.2). A autora utilizou um *Notebook* DELL, com processador Intel *Core i7* (7ª geração), memória *RAM* de *8GB*, *250GB* de *HD* e *flash (SSD)* e Sistema Operacional *Windows* 10. O desenvolvimento do código se deu através da IDE *Sublime*[3] e *Visual Studio Code*[4], com versionamento de todo o código feito via *GIT*. Os desenhos foram criados através da ferramenta *Draw.io*[5].

4.4.1 Biblioteca *dominoes*

Foi utilizada a linguagem de programação Python para a implementação deste trabalho por afinidade da autora, por ter encontrado a biblioteca *dominoes*[6] e pela linguagem já ser amplamente adotada no campo da Inteligência Artificial.

A *dominoes* é uma biblioteca Python para o jogo de Dominó, que fornece interações através de uma interface via linha de comando. Na biblioteca, os jogadores são chamados de 0 e 1. O tabuleiro do jogo consiste em uma corrente de peças encaixadas de ponta a ponta, conectadas por valores iguais. Anteriormente ao início do jogo, é decidido qual peça deve começar, assim como o duplo 6 no jogo tradicional do dominó. Esta peça então aparece, automaticamente, no tabuleiro, como primeira jogada (do jogador que a recebeu). Após esta jogada inicial, os jogadores se alternam na vez de jogar.

[2] http://www.futura.org.br/trilhas/o-que-e-sala-de-aula-invertida/
[3] https://www.sublimetext.com
[4] https://code.visualstudio.com
[5] https://www.draw.io
[6] https://github.com/abw333/dominoes

Durante o jogo, se o jogador da vez tiver uma peça com o valor requerido do tabuleiro ele deve jogá-la, caso contrário, a rodada passa e o jogador seguinte deverá jogar seguindo a mesma lógica. O jogo termina quando um jogador fica sem nenhuma peça na mão ou quando não há mais jogadas possíveis com as peças que estão nas mãos dos jogadores (situação de jogo fechado). É possível que os jogadores ganhem, individualmente, ou que o jogo empate.

4.4.2 Adaptações das Regras para o Programinó

Para o desenvolvimento do Programinó, foram necessários alguns ajustes de regras e lógica na biblioteca *dominoes*, a saber:

1. Por ser desenvolvida para o dominó, a *dominoes* apresenta como valores das "cabeças" das peças números (tipo *int* do Python) de 0 a 6. Assim, foi necessário mudar as "cabeças" das peças para valores e tipos de dados primitivos do Python (*int*, *double*, *str* e *bool*) (Figura 9);

2. Tendo em vista a regra do Programinó, em que tipos de dados e valores que correspondem a cada tipo podem ser "encaixados", foi necessária a criação da classe *head.py*, que abstraísse a representação destes valores, de forma a fazer a linguagem Python entender que valores como uma *string* "Maria" e o tipo *string str* deveriam ser equivalentes. Esta foi uma parte fundamental para o prosseguimento deste trabalho, pois, a partir dela que o Dominó se transformou em Programinó. Ou seja, a lógica do algoritmo toda trabalhada em números de 0 a 6 se modificou para valores e tipos de dados primitivos;

3. Ao invés de 28 peças como o Dominó, o Programinó tem 16 peças que contém valores e tipos de dados;

4. Considerando que serão dois jogadores apenas (humano X humano ou humano X máquina) e 16 peças no total, a distribuição é feita com 8 peças para cada jogador, em vez de 7 como acontece no dominó;

5. Na *dominoes* era realizada uma contagem de pontos, com a soma dos valores das peças do dominó para determinar a pontuação de cada jogador. Este recurso foi retirado por não fazer mais sentido no modelo atual, uma vez que são utilizados diversos tipos de dados, não podendo ser realizada a operação de soma (a ideia é pensar em outra forma de contagem de pontos nos trabalhos futuros);

6. Também foi reduzido o escopo da jogabilidade em equipes (era possível fazer 2x2 também), para que desta forma os jogos 1x1 - humano X máquina e má-

quina X máquina - pudessem acontecer;

7. Em substituição à peça inicial citada anteriormente, para o Programinó, foi escolhida a peça cujo valor era ”Float”em cada “cabeça” da peça. A escolha desta peça aconteceu por nenhum motivo especial;

8. Por fim, foi criada a classe *ia_programino.py* responsável pela inteligência artificial em três níveis de dificuldade para o humano poder jogar contra a máquina (descritos na subseção 4.4.3).

```
12      all_dominoes = [
13          programino.Domino(Head(int), Head(int)),
14          programino.Domino(Head(float), Head(1)),
15          programino.Domino(Head(str), Head(27)),
16          programino.Domino(Head('V'), Head(int)),
17          programino.Domino(Head(float), Head(float)),
18          programino.Domino(Head(int), Head(0.4)),
19          programino.Domino(Head('Oi'), Head(13.1)),
20          programino.Domino(Head(3.14), Head('F')),
21          programino.Domino(Head(bool), Head(bool)),
22          programino.Domino(Head('False'), Head(5)),
23          programino.Domino(Head('True'), Head(0.7)),
24          programino.Domino(Head('Maria'), Head('Verdadeiro')),
25          programino.Domino(Head(str), Head(str)),
26          programino.Domino(Head('Juvi'), Head(int)),
27          programino.Domino(Head(str), Head('Falso')),
28          programino.Domino(Head('Olá, Mundo!'), Head(5.6)),
29      ]
```

Figura 9 – Dados cadastrados no código para as peças do Programinó

O Programinó foi criado como um *fork* (bifurcação) do repositório da *dominoes*, através da plataforma de hospedagem e versionamento de código GitHub. Assim, todas as alterações citadas e demais pequenas modificações podem ser acessadas no repositório do projeto[7].

4.4.3 Minimax para o Programinó

Foram desenvolvidos três níveis de dificuldade para a máquina do Programinó, sendo um nível aleatório (médio), outro mais difícil utilizando o algoritmo Minimax (ver Figura 10) e um último de nível fácil aplicando o Minimax de maneira invertida (escolhendo sempre a pior peça). A ideia é que a partida seja disputada entre jogadores humanos ou entre o jogador humano e a máquina - agente inteligente (NORVIG; RUSSELL, 2014)).

[7] https://github.com/gabrielepessoa/programino

```python
def play_max(self, game, is_root=False):
    self.run += 1
    game = copy.deepcopy(game)
    if not game.valid_moves:
        if game.turn == 0:
            return -1
        elif game.turn == 1:
            return 1
        else:
            return 0
    vmax = -math.inf
    max_move = None

    for move in game.valid_moves:
        gcopy = copy.deepcopy(game)
        gcopy.make_move(*move)
        v = self.play_min(gcopy)
        if v > vmax:
            vmax = v
            max_move = move

    return max_move if is_root else vmax
```

```python
def play_min(self, game):
    self.run += 1
    game = copy.deepcopy(game)
    if not game.valid_moves:
        if game.turn == 0:
            return -1
        elif game.turn == 1:
            return 1
        else:
            return 0
    vmin = math.inf
    min_move = None

    for move in game.valid_moves:
        gcopy = copy.deepcopy(game)
        gcopy.make_move(*move)
        v = self.play_max(gcopy)
        if v < vmin:
            vmin = v
            min_move = move

    return vmin
```

Figura 10 – Código dos métodos de MAX e MIN

No modo aleatório, a máquina escolhe qualquer peça entre as jogadas válidas, sem análise aprofundada e nem prever algo. No Minimax, o algoritmo de Inteligência Artificial é acionado. Este fluxo se repete apenas quando a máquina joga e até que algum jogador ganhe ou empate. O objetivo é escolher a próxima ação, no entanto, o algoritmo simula todas as ações possíveis do próximo jogador e assim sucessivamente. A árvore gerada no Programinó (Figura 11) tem profundidade máxima de 15 níveis; a raiz dela é o tabuleiro inicial (estado atual); cada nó equivale ao estado atual do tabuleiro e pode ser referente ao MIN ou MAX; cada nó filho é um tabuleiro resultante da jogada e cada nó folha (sem filho) é um fim possível do jogo.

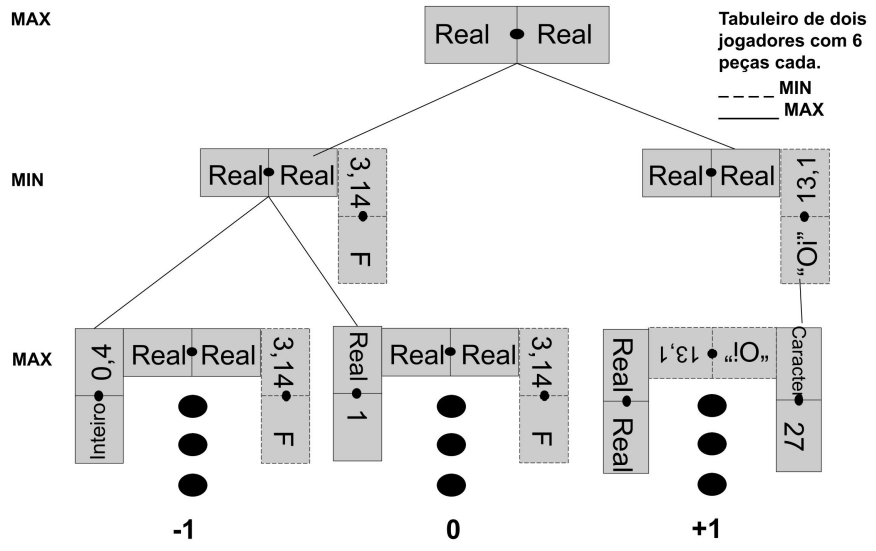

Figura 11 – Exemplo de execução do *minimax* no Programinó

Considerando que cada folha (o estado final de cada rodada) contenha um valor de avaliação de tal jogada e seja um valor alto, este significa vantagem competitiva para o agente e o valor baixo, consequentemente, desvantagem. Para o Programinó, este valor varia entre -1, 0 e 1, sendo vitória do oponente humano, empate e vitória do agente, respectivamente. No nível fácil, esta etapa de decisão foi modificada, com o valor -1 significando a vitória do agente e o valor 1 a vitória do oponente humano, forçando assim a escolha da pior jogada possível para o agente.

4.4.4 Exemplo de Jogadas do Programinó

Buscando ilustrar o cenário atual do Programinó, segue um exemplo de jogadas realizadas, mas, antes de mostrá-lo, é preciso entender o fluxo do algoritmo (Figura 12) e alguns comandos realizados via linha de comando:

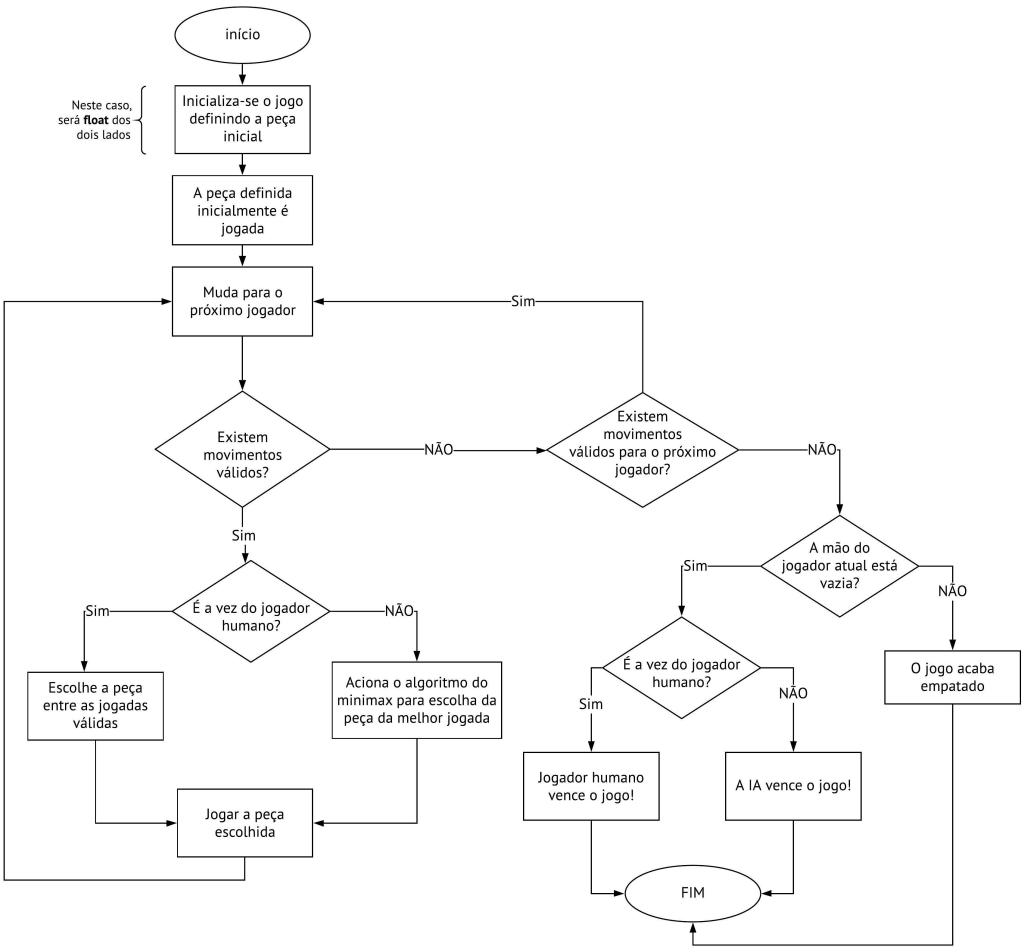

Figura 12 – Fluxograma referente ao passo a passo do algoritmo do Programinó

1. Inicialmente, considera-se que a chamada do *python* seja feita em uma pasta antes de onde está o código do Programinó, conforme mostra a primeira linha da Figura 13, onde, a pasta do Programinó está dentro;

```
C:\Users\gpn\Desktop\TCC>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import programino
>>> from programino import Head
>>> d = programino.Domino(Head(float),Head(float))
>>> g = programino.Game.new(starting_domino=d)
>>> g
Board: [<class 'float'>|<class 'float'>]
Player 0's hand: [<class 'float'>|1][True|<class 'int'>][<class 'str'>|<class 'str'>][False|5][True|0.
7]
Player 1's hand: [3.14|F][Maria|True][<class 'int'>|<class 'int'>][<class 'str'>|False][Juvi|<class 'i
nt'>][Olá, Mundo!|5.6]
Player 1's turn
```

Figura 13 – Configuração inicial de jogo humano contra humano no Programinó

2. O *d* define a peça inicial a ser jogada. No caso do Programinó é a peça cujo valor tem "*float*" nas duas cabeças;

3. O *g* mostra todas as configurações do cenário atual do jogo: quais as peças estão no tabuleiro, quais as peças constam nas mãos do jogador 1 e do jogador 2 e de quem é a vez de jogar;

4. O *g.valid_moves* retorna uma lista contendo as possíveis jogadas. Ou seja, mostra as peças que se encaixam de acordo com o tipo de dado correto e onde elas devem ser inseridas na cabeça da esquerda ou da direita;

5. O *g.make_move(*g.valid_moves[0])* escolhe qual a peça (posição da lista) deve ser usada de acordo com o resultado obtido no comando de *valid_moves*. Obs: a lista começa na posição zero, mas no lugar do zero pode ser qualquer número de acordo com o tamanho da lista retornada;

6. O *g.hands[0]* retorna uma lista com todas as peças que o jogador 1 tem em mãos. No lugar do 0 (zero), presente no comando, também pode ser 1 (um), caso a necessidade seja saber as peças das mãos do outro jogador.

4.4.4.1 Exemplo de uma partida humano contra humano

Após a configuração inicial presente na Figura 13, a Figura 14 mostra o jogador 2 verificando quais as peças válidas e escolhendo a primeira opção presente no índice 0 da lista retornada.

Em seguida, a Figura 15 ilustra o jogador 1 verificando as peças presentes em suas mãos, quais as jogadas válidas e escolhendo a opção presente no índice 2, ou seja, a peça com os valores "*false*" e "*5*" será encaixada, de forma invertida, no lado esquerdo do tabuleiro.

Por fim, a Figura 16 mostra que o jogador 1 joga sua última peça e vence o jogo.

```
>>> g
Board: [<class 'float'>|<class 'float'>]
Player 0's hand: [<class 'float'>|1][True|<class 'int'>][<class 'str'>|<class 'str'>][False|5][True|0.
7]
Player 1's hand: [3.14|F][Maria|True][<class 'int'>|<class 'int'>][<class 'str'>|False][Juvi|<class 'i
nt'>][Olá, Mundo!|5.6]
Player 1's turn
>>> g.valid_moves
(([3.14|F], True), ([Olá, Mundo!|5.6], True))
>>> g.make_move(*g.valid_moves[0])
>>> g
Board: [False|3.14][<class 'float'>|<class 'float'>]
Player 0's hand: [<class 'float'>|1][True|<class 'int'>][<class 'str'>|<class 'str'>][False|5][True|0.
7]
Player 1's hand: [Maria|True][<class 'int'>|<class 'int'>][<class 'str'>|False][Juvi|<class 'int'>][Ol
á, Mundo!|5.6]
Player 0's turn
```

Figura 14 – Exemplo onde jogador 2 escolhe uma peça entre as jogadas válidas para jogar

```
>>> g
Board: [False|3.14][<class 'float'>|<class 'float'>]
Player 0's hand: [<class 'float'>|1][True|<class 'int'>][<class 'str'>|<class 'str'>][False|5][True|0.
7]
Player 1's hand: [Maria|True][<class 'int'>|<class 'int'>][<class 'str'>|False][Juvi|<class 'int'>][Ol
á, Mundo!|5.6]
Player 0's turn
>>> g.hands[0]
[<class 'float'>|1][True|<class 'int'>][<class 'str'>|<class 'str'>][False|5][True|0.7]
>>> g.valid_moves
(([<class 'float'>|1], False), ([True|<class 'int'>], True), ([False|5], True), ([True|0.7], True), ([
True|0.7], False))
>>> g.make_move(*g.valid_moves[2])
>>> g
Board: [5|False][False|3.14][<class 'float'>|<class 'float'>]
Player 0's hand: [<class 'float'>|1][<class 'str'>|<class 'str'>][False|5][True|0.7]
Player 1's hand: [Maria|True][<class 'int'>|<class 'int'>][<class 'str'>|False][Juvi|<class 'int'>][Ol
á, Mundo!|5.6]
Player 1's turn
```

Figura 15 – Exemplo onde jogador 1 escolhe uma peça entre as jogadas válidas para jogar

```
>>> g
Board: [Juvi|<class 'int'>][5|False][True|Maria][<class 'str'>|<class 'str'>][Olá, Mundo!|5.6][<class
'float'>|1][<class 'int'>|<class 'int'>][5|False][False|3.14][<class 'float'>|<class 'float'>]
Player 0's hand: [True|0.7]
Player 1's hand: [<class 'str'>|False]
Player 0's turn
>>> g.make_move(*g.valid_moves[0])
Result(player=0, won=True, points=1)
```

Figura 16 – Exemplo de finalização do jogo, tendo como vencedor o jogador 1

5 Experimento

Nesta seção constam os testes comparativos realizados entre os algoritmos de Inteligência Artificial implementados em seus diferentes níveis de dificuldades, buscando demonstrar a real eficiência do Minimax com algoritmos menos efetivos. Ou seja, verificar o quão perto do objetivo (ganhar) esses testes chegaram.

5.1 Metodologia do Experimento

A infraestrutura aqui utilizada contou com a mesma configuração citada no início da seção 4.4. Para realizar os experimentos e comparar os níveis de dificuldades implementados, foram executados 1000 testes em cada um dos cenários:

1. Algoritmo aleatório x Algoritmo aleatório;
2. Algoritmo aleatório x Algoritmo Minimax;
3. Algoritmo aleatório x Algoritmo Minimax invertido.

Para tal, foi desenvolvido o módulo "*__main__.py*", em que entende-se por módulo todo código que estiver fora da definição de classes e funções e que será executado automaticamente ao inicializá-lo (Python Brasil[1], 2019). Isto possibilitou que os experimentos fossem realizados de forma automatizada, armazenando em memória o resultado de cada execução (jogo) e, ao final, salvando todos os resultados dos experimentos, e a somatória destes, em arquivos *CSV* separados para análise posterior.

Além disso, foi desenvolvido um comando (ver Figura 17) para parametrização e execução dos testes através de linha de comando.

```
C:\Users\gpn>python -m programino 1000 1000 1000
```

Figura 17 – Comando para executar os experimentos

Em que o módulo *__main__.py* espera o comando com a quantidade de simulações de cada um dos três experimentos.

Os resultados e análise dos experimentos estão descritos na seção seguinte. Com o experimento 1 descrito na subseção 5.2.1, o experimento 2 na subseção 5.2.2 e o experimento 3 na subseção 5.2.3.

[1] https://wiki.python.org.br/ModulosPacotes

5.2 Resultados

5.2.1 Máquina algoritmo aleatório x Máquina algoritmo aleatório

Neste cenário, foram simulados 1000 jogos com ambos os jogadores (1 e 2) utilizando o algoritmo aleatório, ou seja, escolhendo qualquer uma das peças válidas. Foi possível observar uma equivalência entre os valores, tendo em vista que eles foram muito próximos (conforme Figura 18). O jogador 1 ganhou 313 das 1000 vezes, o jogador 2 ganhou 323 e ocorreram 364 empates.

Figura 18 – Resultado do experimento Aleatório x Aleatório do Programinó

Comparando este resultado com o trabalho de (COSTA, 2016), que aplica agentes inteligentes no jogo de Dominó, é possível verificar que foram próximos, mas com algumas observações importantes a fazer:

- Um resultado esperado se tratando de um nível aleatório;
- (COSTA, 2016) mostra o resultado do experimento Aleatório x Aleatório na perspectiva do jogador 1, enquanto aqui é possível ver os resultados de ambos os jogadores;
- A porcentagem de empate no Programinó é em torno de 33%, enquanto no trabalho de (COSTA, 2016) fica em torno de 1%, pois, no dominó tradicional são raras as vezes em que um jogo realmente termina empatado. Na maioria

das vezes em que o jogo, teoricamente, empata, alguém ganha na contagem dos pontos (Ex: Mesmo que o jogo tenha “fechado” e ninguém tenha a peça adequada para colocar no tabuleiro, nas mãos do jogador 1 tem uma peça com 6 pontos, enquanto na mão do jogador 2 tem uma peça com 5 pontos. Com isso, o jogador 2 vence por ter menos pontos). Já no caso do Programinó, se tratando de valores e tipos de dados, não tem como dizer que um tipo de dado é “maior” ou “menor” que o outro para tal contagem de pontos, logo, o Programinó é mais propício ao empate.

5.2.2 Máquina algoritmo aleatório x Máquina algoritmo Minimax

Neste cenário, também foram simulados 1000 jogos entre o jogador 1, utilizando o algoritmo aleatório, e o jogador 2, utilizando o algoritmo Minimax. Foi possível observar uma grande vantagem entre os ganhos do Minimax comparado ao outro nível (Figura 19). O jogador 1 ganhou 56 das 1000 vezes, o jogador 2 ganhou 497 partidas e foram 447 empates. Isso demonstra a eficiência do Minimax, pois neste experimento ele perdeu em apenas 5,6% das vezes, ganhando ou empatando nas demais.

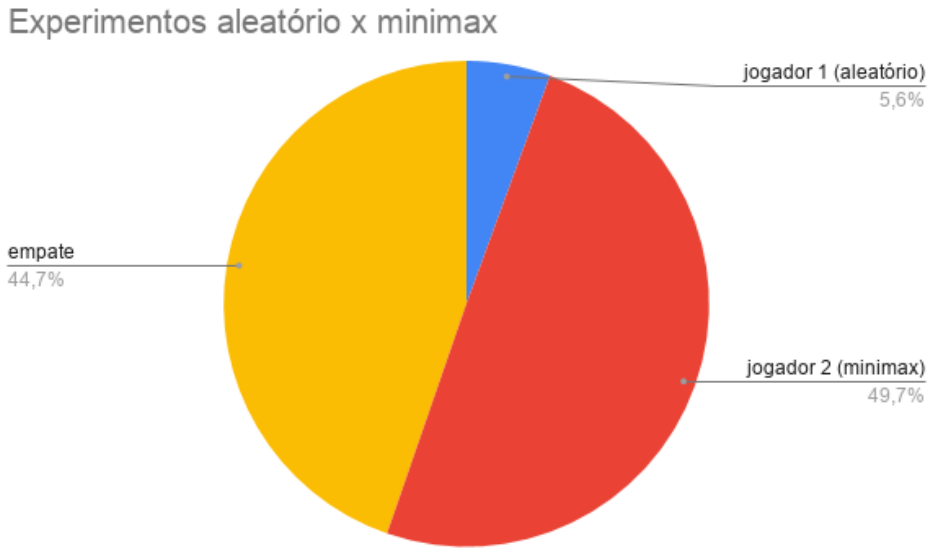

Figura 19 – Resultado do experimento Aleatório x Minimax

Assim como no primeiro experimento, houve uma grande quantidade de empates como resultado dos jogos. Isto, novamente, se justifica pelo fato do Programinó não ter um recurso adicional ao “fechamento”, como existe no jogo de Dominó original. Provavelmente, o percentual de vitórias da máquina utilizando o Minimax seria maior caso tal recurso existisse.

Por fim, estudou-se o desenvolvimento e aplicação nos experimentos da conhecida evolução do algoritmo Minimax com a poda Alpha-Beta, para comparação entre

os dois demais algoritmos desenvolvidos. Porém, como afirmado em (NORVIG; RUSSELL, 2014), o algoritmo otimizado encontra comprovadamente as mesmas jogadas do algoritmo original, apenas com a execução em um tempo consideravelmente inferior. Assim, optou-se pelo foco no término do desenvolvimento do projeto e pela sugestão a esta implementação no último capítulo, como trabalhos futuros.

5.2.3 Máquina algoritmo aleatório x Máquina algoritmo Minimax invertido

Neste cenário de testes também foram executados seguindo as mesmas configurações dos demais, mas agora o jogador 1 jogou de forma aleatória e o jogador 2 com o minimax invertido, ou seja, adaptado para escolher sempre a pior jogada.

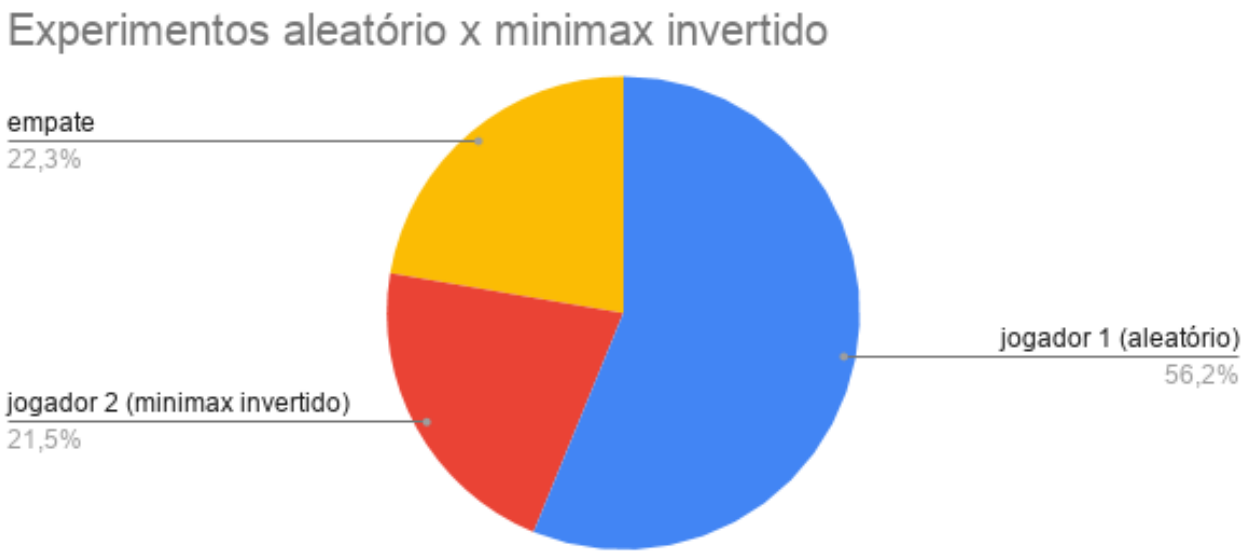

Figura 20 – Resultado do experimento Aleatório x Minimax Invertido

Conforme mostra a Figura 20, os testes demonstram que o jogador 1 (aleatório) ganhou em 56,2% das vezes, empatou em 22,3% e perdeu apenas em 21,5% das vezes. Desta forma, o minimax invertido pode se categorizar como um nível fácil para jogar contra um jogador humano. Mesmo que o humano jogue de forma aleatória, as chances dele ganhar são bem grandes.

6 Considerações Finais

Estudos comprovam que jogos digitais auxiliam a educação de forma efetiva, desenvolve o raciocínio lógico, motiva, engaja, trabalha a compreensão conceitual e aproxima alunos e professores. Porém, existe uma carência desses jogos com base pedagógica e para além da matemática (DIAS et al., 2017). Neste contexto, o presente trabalho traz para a sociedade acadêmica um artefato lúdico educacional que auxilia o aprendizado de tipos de dados na programação. O algoritmo desenvolvido utiliza de estratégias e jogadas clássicas do jogo tradicional do dominó, o que facilita a jogabilidade ao primeiro contato com o Programinó, e implementa o algoritmo de Inteligência Artificial com busca em profundidade e Teoria dos Jogos, Minimax. Espera-se que o Programinó, como um jogo digital educacional, possa ser utilizado em sala de aula, por educadores e estudantes de computação, trazendo motivação o suficiente para praticar e consolidar o assunto de tipos de dados de forma lúdica. E, para além dos muros da escola, que o Programinó esteja acessível à todas as pessoas interessadas em aprender sobre o assunto.

Conforme resultado explanado no Capítulo 5, foi verificado que o nível aleatório é inferior ao Minimax quanto à sua efetividade. Foram simulados jogos de testes e verificou-se que em apenas 5,6% das vezes ele perde e, nas demais, ganha ou empata. O algoritmo implementado foi testado e validado, comprovando a eficiência de dois níveis de dificuldade (fácil e difícil), mas, precisa ser adaptado antes de chegar ao usuário final (sugestões na seção 6.2). Além disso, espera-se colaborar, motivar e deixar uma reflexão para professores sobre a sua metodologia de ensino ainda ancorada em métodos tradicionais, sem a utilização de recursos lúdicos e interativos. Por fim, é importante ressaltar que, assim como citado por (FREITAS et al., 2018), jogos não substituem a aula.

6.1 Dificuldades Encontradas

Sobre o desempenho do Minimax, ao testá-lo, foi visualizado que o tempo que levaria para executar uma única partida era exponencial (já era esperado tendo em vista que o algoritmo tem tempo de execução exponencial já comprovado pela literatura (NORVIG; RUSSELL, 2014)), pois a árvore pode atingir profundidade 15 com uma imensa quantidade de ramos, se tornando inoportuno durante uma partida. Com isso, foi pensada na estratégia de diminuir de 8 para 6 peças distribuídas para cada jogador na execução dos experimentos.

6.2 Trabalhos Futuros

Para trabalhos futuros, fica a sugestão da otimização do algoritmo Minimax, através da poda Alpha-Beta; aplicação de Aprendizado de Máquina para que o algoritmo adquira novas habilidades e possa atualizar seus próprios movimentos; criação de uma interface gráfica para poder testar com estudantes iniciantes; funcionalidade de escolha de qualquer peça da mão com um *feedback* construtivo sobre o erro, para que dessa forma, o aprendizado seja construído; adicionar mais peças para cada tipo de dado para que não sejam sempre os mesmos valores e os estudantes não “decorem” que tal valor é de um tipo de dado; avaliação pedagógica com estudantes e desenvolvimento de um nível médio de dificuldade para o Programinó.

Referências

AMARAL, H. F.; MACHADO, A. F.; BRAGA, J. L. Um sistema de apoio a decisão baseado em minimax para um jogo de estratégia em administração rural. *CEP*, v. 36180, p. 000, 2013. Citado na página 29.

ARAUJO, E. C. de et al. O papel do hábito de estudo no desempenho do aluno de programaç ao. 2013. Citado na página 13.

AURELIANO, V.; TEDESCO, P. Ensino-aprendizagem de programação para iniciantes: uma revisão sistemática da literatura focada no sbie e wie. *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação - SBIE*, v. 23, 2012. Citado na página 15.

BENNEDSEN, J.; CASPERSEN, M. E. Failure rates in introductory programming. *AcM SIGcSE Bulletin*, ACM, v. 39, n. 2, p. 32–36, 2007. Citado na página 21.

BONET, B.; GEFFNER, H. Planning as heuristic search. *Artificial Intelligence*, Elsevier, v. 129, n. 1-2, p. 5–33, 2001. Citado na página 24.

BRASIL ESCOLA. *DOMINÓ PEDAGÓGICO PARA O ENSINO DE SINONÍMIA E ANTONÍMIA*. 2018. Disponível em: <https://educador.brasilescola.uol.com.br/estrategias-ensino/domino-pedagogico-para-ensino-sinonimia-antonimia.htm>. Acesso em: 01 dez. 2019. Citado na página 12.

CAMPBELL, M.; JR, A. J. H.; HSU, F.-h. Deep blue. *Artificial intelligence*, Elsevier, v. 134, n. 1-2, p. 57–83, 2002. Citado na página 29.

CANDIDO, C. et al. Recursos de ensino e aprendizagem: elaboração de um material didático sobre o tema artrópodes destinado a alunos do ensino fundamental e médio. *Cadernos da Pedagogia*, v. 5, n. 10, 2012. Citado na página 12.

CANDIDO, D. et al. Estudo comparativo de abordagens referentes ao desenvolvimento do pensamento computacional. *Anais do Workshop de Informática na Escola*, v. 23, 2017. Citado 2 vezes nas páginas 13 e 18.

CHAVES, L. S.; BEZERRA, C. I. M. et al. Ensino de programação em escolas públicas: Relato de uma ação do pet-ti. In: *Anais do Workshop de Informática na Escola*. [S.l.: s.n.], 2019. v. 25, n. 1, p. 667. Citado na página 21.

COMPUTING IN THE CORE. *What is Computer Science and What do People Do Once They Know It?* 2018. Disponível em: <https://code.org/files/computer_science_is_foundational.pdf>. Acesso em: 01 dez. 2019. Citado na página 12.

COSTA, J. A. S. d. Inteligência artificial em jogos de tabuleiro: proposição de uma heurística para o jogo de dominós. 2016. Citado 2 vezes nas páginas 31 e 41.

DIAS, J. L. et al. O uso de jogos digitais na educac ao basica: uma revisao sistematica da literatura. 2017. Citado na página 44.

FIALHO, N. N. Os jogos pedagógicos como ferramentas de ensino. In: *Congresso nacional de educação*. [S.l.: s.n.], 2008. v. 6, p. 12298–12306. Citado na página 12.

FREITAS, P. et al. Proposta e avaliação de um jogo digital para o auxílio no ensino de tipos de dados. *VII Jornada de Informática do Maranhão*, v. 1, 2018. Citado 3 vezes nas páginas 18, 31 e 44.

GARDNER, H. *Inteligências múltiplas*. [S.l.]: Porto Alegre: Artes Médicas, 1995. Citado na página 20.

GIRAFFA, M. M.; MORA, M. da costa. Evasão na disciplina de algoritmo e programação: um estudo a partir dos fatores intervenientes na perspectiva do aluno. In: *Congresos CLABES*. [S.l.: s.n.], 2013. Citado na página 21.

GODOY, M. C. J. et al. Ensino de equivalência monetária por meio de um jogo de dominó adaptado. *Acta Comportamentalia: Revista Latina de Análisis de Comportamiento*, Universidad Veracruzana, v. 23, n. 2, p. 117–135, 2015. Citado na página 12.

GOMES, M. et al. Um estudo sobre erros em programação-reconhecendo as dificuldades de programadores iniciantes. In: *Anais dos Workshops do Congresso Brasileiro de Informática na Educação*. [S.l.: s.n.], 2015. v. 4, n. 1, p. 1398. Citado na página 12.

LAHTINEN, E.; ALA-MUTKA, K.; JÄRVINEN, H.-M. A study of the difficulties of novice programmers. *Acm Sigcse Bulletin*, ACM, v. 37, n. 3, p. 14–18, 2005. Citado na página 22.

LIMA, P.; VIEIRA, P.; BRANDÃO, L. Ensino de algoritmos, programação e matemática: panorama e estudo de caso para estudantes de escolas públicas brasileiras. In: *Anais do Workshop de Informática na Escola*. [S.l.: s.n.], 2019. v. 25, n. 1, p. 697. Citado na página 22.

MAŃDZIUK, J. Computational intelligence in mind games. In: *Challenges for Computational Intelligence*. [S.l.]: Springer, 2007. p. 407–442. Citado na página 23.

MAŃDZIUK, J. Some thoughts on using computational intelligence methods in classical mind board games. In: IEEE. *2008 IEEE International Joint Conference on Neural Networks*. [S.l.], 2008. p. 4002–4008. Citado na página 23.

MEDEIROS, R. P. Hello, world: uma análise sobre dificuldades no ensino e na aprendizagem de introdução à programação nas universidades. Universidade Federal de Pernambuco, 2019. Citado na página 17.

MEDEIROS, T.; SILVA, T. Ensino de programação utilizando jogos digitais: uma revisão sistemática da literatura. *CINTED-UFRGS*, v. 1, 2013. Citado na página 16.

NASH, J. F. et al. Equilibrium points in n-person games. *Proceedings of the national academy of sciences*, USA, v. 36, n. 1, p. 48–49, 1950. Citado na página 27.

NEUMANN, J. V.; MORGENSTERN, O. Theory of games and economic behavior. Princeton University Press, 1944. Citado na página 26.

NORVIG, P.; RUSSELL, S. *Inteligência Artificial: Tradução da 3a Edição*. [S.l.]: Elsevier Brasil, 2014. v. 1. Citado 8 vezes nas páginas 23, 24, 25, 26, 28, 35, 43 e 44.

OLIVEIRA, Y.; FARIAS, C. projetoéden jogo sério sobre variáveis e tipos de dados. *Proceedings of SBGames 2018*, v. 1, 2018. Citado 2 vezes nas páginas 19 e 31.

ORGÂNICA NATURAL MARKETING. *Algoritmo Minimax - Introdução à Inteligência Artificial*. 2019. Disponível em: <https://www.organicadigital.com/blog/algoritmo-minimax-introducao-a-inteligencia-artificial/>. Acesso em: 01 dez. 2019. Citado na página 28.

PIAGET, J. *A formação do símbolo na criança: imitação, jogo e sonho, imagem e representação*. [S.l.]: Livros Técnicos e Científicos, 2004. 1–227 p. Citado na página 32.

PIRES, A.; PRATES, J. Uma contribuição ao ensino de programação na educação básica. *Anais do Workshop de Informática na Escola*, v. 25, n. 1, p. 1274, 2019. ISSN 2316-6541. Citado na página 22.

PRICE, B.; BAECKER, R.; SMALL, I. *An introduction to software visualization*. [S.l.]: Mit Press, 1998. 3–27 p. Citado na página 17.

SANTOS, J. E. B. d. O dominó no contexto do ensino e aprendizagem da matemática. 2013. Citado na página 12.

SANTOS, J. G. W.; ALVES, J. M. O jogo de dominó como contexto interativo para a construção de conhecimentos por pré-escolares. SciELO Brasil, 2000. Citado na página 32.

SANTOS, W. O. d.; NETO, S.; JUNIOR, C. S. Processo de virtualização de jogos para uso como mecanismo de apoio ao processo de ensino e aprendizagem da disciplina de matemática. *Anais do Seminário de Jogos Eletrônicos, Educação e Comunicação*, v. 1, 2015. Citado 2 vezes nas páginas 13 e 18.

SAVI, R.; ULBRICHT, V. R. Jogos digitais educacionais: benefícios e desafios. *Renote*, v. 6, n. 1, 2008. Citado na página 15.

SHANNON, C. E. Xxii. programming a computer for playing chess. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, Taylor & Francis, v. 41, n. 314, p. 256–275, 1950. Citado na página 23.

SILVA, A. B. V.; MORAES, M. G. Jogos pedagógicos como estratégia no ensino de morfologia vegetal. 2011. Citado na página 12.

SILVA, A. C. S. Virtualização de jogos tradicionais para aprendizagem de matemática: uma avaliação do jogo cubra doze em versão digital. *Universidade Federal Rural de Pernambuco*, v. 1, 2018. Citado na página 18.

SILVA, T. R. d.; MEDEIROS, T.; ARANHA, E. Jogos digitais para ensino e aprendizagem de programação: uma revisão sistemática da literatura. *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação - SBIE)*, v. 25, n. 1, 2014. Citado na página 22.

SILVA, T. R. da et al. Ensino-aprendizagem de programação: uma revisão sistemática da literatura. *Revista Brasileira de Informática na Educação*, v. 23, n. 1, 2015. Citado 2 vezes nas páginas 12 e 19.

SOUZA, D. M.; BATISTA, M. d. S.; BARBOSA, E. F. Problemas e dificuldades no ensino e na aprendizagem de programação: Um mapeamento sistemático. *Revista Brasileira de Informática na Educação*, v. 24, n. 1, p. 39–52, 2016. Citado 2 vezes nas páginas 16 e 17.

TEIXEIRA, H. *O que significa o termo "Scaffolding" em educação?* Hélio Teixeira blog, 2015. Acesso em 07 de dezembro de 2019. Disponível em: <http://www.helioteixeira.org/ciencias-da-aprendizagem/o-que-significa-o-termo-scaffolding-em-educacao/>. Citado na página 33.

THOMAS, L. C. *Games, theory and applications*. [S.l.]: Courier Corporation, 2012. Citado na página 27.

TURING, A. M. Digital computers applied to games. *Faster than thought*, Pitnan & Sons, 1953. Citado na página 23.

WING, J. M. Computational thinking. *Communications of the ACM*, v. 49, n. 3, p. 33–35, 2006. Citado na página 12.

WOOD, D.; BRUNER, J. S.; ROSS, G. The role of tutoring in problem solving. *Journal of child psychology and psychiatry*, Wiley Online Library, v. 17, n. 2, p. 89–100, 1976. Citado na página 33.

XENOS, M.; PIERRAKEAS, C.; PINTELAS, P. A survey on student dropout rates and dropout causes concerning the students in the course of informatics of the hellenic open university. *Computers & Education*, Elsevier, v. 39, n. 4, p. 361–377, 2002. Citado na página 21.