P&D 2022 DESIGN O DESENHO DO CAMPO

14º Congresso Brasileiro de Design
ESDI Escola Superior de Desenho Industrial
ESPM Escola Superior de Propaganda e Marketing

Entre foguetes, estrelas e canetas: relato de experiência de ensino de princípios da Computação para estudantes de graduação em Design

Between rockets, stars, and pens: experience report on teaching Computing principles to undergraduate Design students

SANT'ANNA, Hugo Cristo; Doutor em Psicologia; Universidade Federal do ES

hugo.santanna@ufes.br

Nos últimos 20 anos, as temáticas do design computacional e generativo tornaram-se mais presentes nos interesses e práticas de estudantes de Design. Entretanto, persistem barreiras para a aprendizagem dos princípios, conceitos, práticas e perspectivas da Computação por esses estudantes, tais como a falta de fundamentos teóricos e dificuldades de uso dos ambientes de programação criados para outras áreas. Este artigo apresenta um relato de experiência de ensino de princípios da Computação no Curso de Design da Ufes, utilizando uma abordagem de ensino e ambiente específicos para este público. Por dois semestres, um total de 150 estudantes testaram a proposta e produziram cerca de 2600 programas, cujas estruturas e funcionamentos ilustram processos de resolução de problemas no âmbito da abordagem didática. Os resultados preliminares sugerem que a abordagem promoveu a aprendizagem de conceitos da Computação, fomentou colaboração entre os aprendizes e oportunizou a expressão individual por meio da programação.

Palavras-chave: Ensino; Design Computacional; Computação.

In the last 20 years, computational design and generative design disciplines became more present in undergraduate Design students' interests and practices. Nonetheless, students still face barriers for learning Computing principles, concepts, practices, and perspectives, such as the lack of theoretical foundations, and difficulties to use programing environments created for other areas. This paper presents an experience report on teaching Computing principles at the undergraduate Design course at Ufes, using a specific teaching approach designed for this audience. For two terms, a total of 150 students tested the approach and developed around 2600 programs, whose structures and execution illustrate problem-solving processes within the scope of the didactic approach. Preliminary results suggest that the approach promoted learning of Computing concepts, fostered collaboration between apprentices, and provided opportunities for individual expression through programming.

Keywords: *Teaching; Computational Design; Computing.*

1 Introdução

Nos anos 2000, surgiram diversas iniciativas que culminaram no aumento da presença de temáticas como “design computacional” e “design generativo” nos interesses e práticas de designers. Ambientes de programação como Processing (REAS; FRY, 2007) ofereceram a designers e demais profissionais criativos oportunidades para explorar possibilidades expressivas de linguagens de programação, sem precisarem passar pelo mesmo tipo de formação que cientistas da Computação. Apesar dos diferentes objetivos e aplicações, essas iniciativas tinham em comum a oferta de rica documentação, exemplos, tutoriais e instalação descomplicada, reduzindo as barreiras para que designers pudessem dar os primeiros passos na incorporação dessas possibilidades a seus projetos (cf. SANT’ANNA et al., 2012).

Por design computacional, entende-se práticas de projeto cujos produtos são *algoritmos* que *computam* as soluções para problemas de interesse da projetista. Em vez de especificá-las por meio de ferramentas de desenho e editoração do tipo “apontar e clicar”, encontradas em programas de *desktop publishing*, em design computacional designers *programam* algoritmos e exploram espaços de parâmetros que controlam o comportamento das possíveis soluções para aqueles problemas. Em outros termos, trata-se da diferença entre os usos da computação *no* design e da computação *do* design (SANT’ANNA, 2019). Já a vertente generativa explora distintos graus de autonomia dos algoritmos na obtenção das soluções de projeto, seja pela adição de aleatoriedade aos parâmetros informados pelo projetista, seja pela aplicação de sistemas generativos como autômatos celulares, gramáticas de formas (*shapes*) e gramáticas de linguagens formais no processo de síntese (BUHAMDAN; ALWISY; BOUFERGUENE, 2021).

Algoritmos, indispensáveis às práticas de design computacional e generativo, são conjuntos finitos de regras que fornecem sequências de operações para solucionar tipos específicos de problemas (KNUTH, 1997). Algoritmos têm zero ou mais entradas e uma ou mais saídas, além de propriedades que os diferem do entendimento de “receita” no senso comum, tais como finitude, não-ambiguidade e de preocupações quanto à sua eficácia e tratabilidade.

Na atividade de designers visuais e demais projetistas das vertentes computacional e generativa, algoritmos são empregados em funções gráficas de desenho, animação, interação humano-computador, entrada e saída de dados. Há algoritmos para o traçado de pontos (*pixels*) isolados, desenho de primitivas geométricas (polígonos, curvas, sólidos), texturização e iluminação de superfícies bi ou tridimensionais, carregamento, transformações, amostragem e filtragem de imagens (cf. AZEVEDO; CONCI; LETA, 2008). Ambientes como Processing, em outra direção, oferecem bibliotecas nativas que implementam essas funções de “baixo nível”, permitindo que designers elaborem rapidamente algoritmos combinando recursos nativos da linguagem em desenhos, interfaces, animações, visualizações de dados, jogos, entre outros. Trata-se de importante simplificação dos passos iniciais, removendo obstáculos entre as primeiras linhas de código e a obtenção dos resultados visuais desejados.

Sendo assim, a criação de oportunidades de desenvolvimento de competências e habilidades de design computacional demandaria de educadores e instituições de ensino, no mínimo: 1) a familiarização de aprendizes com ambientes, linguagens e práticas de programação; 2) o ensino de conceitos da Ciência da Computação e suas aplicações à construção de algoritmos de propósito geral e de design visual; e 3) o planejamento de situações didáticas que contribuam para a transferência dos conhecimentos aprendidos entre diferentes situações de projeto, reduzindo gradualmente a dependência de recursos simplificados e aumentando a capacidade expressiva dos aprendizes. Este artigo consiste no relato de uma experiência didática elaborada com o intuito de satisfazer as demandas citadas no contexto do Curso de Design da Universidade Federal do Espírito Santo – Ufes (Vitória, ES).

O restante deste texto está organizado da seguinte forma: na sequência, trataremos das competências e habilidades da Ciência da Computação a serem desenvolvidas por estudantes de Design, considerando o interesse crescente pelas temáticas de design computacional e generativo. Na terceira seção, apresentaremos as origens da abordagem de ensino, denominada RocketSocket, que se propôs a suprir as três demandas supracitadas.

Na quarta seção, caracterizaremos brevemente o Curso de Design da Ufes e a disciplina em que a abordagem foi aplicada, detalhando o conteúdo programático, estratégias de ensino e avaliação. A quinta seção apresenta a implementação Web de RocketSocket e a sexta discorre sobre os resultados da proposta nos dois primeiros semestres letivos da experiência, ilustrando processos de resolução de problemas pelos aprendizes no âmbito da abordagem pedagógica. A última seção discute os resultados articulados às três demandas declaradas, apontando oportunidades para estudos subsequentes e indicando limitações deste relato.

2 Princípios, conceitos, práticas e perspectivas da Computação

Seymour Papert e colaboradores foram precursores da difusão de usos dos computadores entre usuários não especialistas, especialmente crianças e adolescentes (cf. PAPERT, 1972). Estas iniciativas caracterizaram computadores como "ferramentas para pensar" conceitos matemáticos, científicos e sobre o próprio pensamento do aprendiz. Estas ideias estão presentes em pontos fundamentais deste artigo e na abordagem pedagógica proposta.

Embora Papert (1980) tenha sido um dos pioneiros no uso da expressão *Computational Thinking* (pensamento computacional, doravante "PC"), a coluna escrita por Jeannette Wing (2006) foi uma das principais responsáveis pela difusão da conotação atual. Segundo esta autora, PC seria um conjunto de habilidades e competências fundamentais, associadas a profissionais da Ciência da Computação, mas que poderiam ser aprendidas por todos. Wing listou uma série de práticas e conceitos associados ao PC, entre eles o uso de estratégias de incorporação, redução, simulação ou transformação para reformular problemas difíceis em outros que possamos resolver; o uso de abstração e decomposição para lidar com tarefas complexas; uso de raciocínio heurístico para descobrir soluções; e a seleção do nível adequado de representação para tornar um problema tratável.

No entanto, Denning e Tedre (2021) argumentaram que a caracterização corrente do PC seria mais diversa, com relativo consenso sobre as habilidades e conceitos envolvidos, mas diferentes percursos de formação. Haveria interpretações divergentes da relação entre PC e Computação, do vínculo entre PC, computadores e programação, e da interação de PC e outros campos. No mesmo ensaio, os autores definiram PC como (ibid., p. 365, tradução nossa):

> habilidades mentais e práticas para projetar computações que fazem com que computadores realizem tarefas por nós, e para explicar e interpretar o mundo nos termos de processos de informação.

Os diferentes percursos formativos para o desenvolvimento das habilidades relacionadas ao PC e as possibilidades de explicar e interpretar o mundo como processos de informação são centrais para o presente relato. Denning e colaboradores (DENNING, 2004; DENNING; MARTELL, 2015), sintetizando esforços de grupos de trabalho em atividade desde os anos 2000, organizaram "grandes princípios" da Computação. Estas categorias seriam janelas para os conhecimentos da área, a partir das quais um mesmo tema pode ser considerado por diferentes visões não exclusivas. Os princípios foram divididos em *mecânicas*, leis e regularidades dos processos de computação, comunicação, coordenação, memorização e

avaliação, e *sabedoria projetual*, referente aos conhecimentos acumulados ao longo da história sobre a construção de sistemas usáveis, seguros e confiáveis.

Denning e Martell (2015) relataram que a Computação dependeria de *práticas* como programação, uso de sistemas, modelagem e pensamento computacional; profissionais e clientes da área se encontrariam em *domínios* como inteligência artificial, segurança, análise de dados e tecnologia da informação; e haveria *tecnologias centrais*, como linguagens de programação, redes, sistemas operacionais, interação humano-computador e *softwares*. A iniciativa de organização dos princípios teve, entre outras motivações, fornecer uma linguagem comum para a discuti-los com outras áreas, inspirar novas abordagens para o ensino e a aprendizagem da Computação e despertar o interesse das novas gerações.

Tendo em conta as novas gerações, Brennan e Resnick (2012) elaboraram um quadro de referência para a avaliação do desenvolvimento do PC organizado em conceitos, práticas e perspectivas, baseado na linguagem de programação Scratch. Apesar da ênfase na educação básica (K-12 nos EUA), estes autores fizeram recomendações acerca das situações de ensino-aprendizagem que podem ser relevantes para iniciantes de qualquer idade: a) explorar conceitos comuns a muitas linguagens de programação – sequências, laços, condicionais, operadores, dados, eventos e paralelismo; b) desenvolver práticas tais como ser iterativo e incremental, abstrair e modularizar soluções, testar e depurar, reutilizar código próprio e de outros programadores; c) fomentar a expressão individual por meio do PC, a conexão com os pares e a reflexão crítica sobre o potencial e limites das tecnologias.

No contexto do presente trabalho, os “grandes princípios” e o quadro de referência relacionam-se explicitamente às demandas de (1) familiarização com ferramentas e práticas e (2) aprendizagem de conceitos da Computação. Estas propostas são duas entre muitas existentes e, ainda assim, ilustram a articulação necessária entre temas gerais do percurso formativo em PC e a aplicação deles na resolução de problemas específicos de design computacional. Isto é, a demanda de planejamento de situações didáticas (3) requer a articulação das duas demandas anteriores, de modo que mecânicas e sabedoria projetual sejam exploradas por aprendizes em domínios cujos projetos empreguem conceitos, exercitem práticas relacionadas ao PC e incentivem a adoção de perspectivas críticas na interação com a tecnologia.

3 Origens da linguagem e ambiente RocketSocket

As propostas de Denning e Martell (2015) e Brennan e Resnick (2012) formaram as bases da abordagem de ensino elaborada na tese de doutorado do autor do presente artigo (SANT’ANNA, 2014). Naquele estudo, evocações livres sobre os princípios da Computação foram coletadas de 86 estudantes de Design de três IES brasileiras. Após a emissão das evocações, os participantes explicaram como entendiam a relação entre cada princípio e a Computação. Outras questões do instrumento indagaram aos estudantes como os computadores e como a Computação os ajudavam a solucionar problemas de Design, fossem acadêmicos, profissionais ou do cotidiano.

Os dados foram analisados a partir da Teoria do Núcleo Central das Representações Sociais (ABRIC, 1993; SÁ, 1996), gerando uma hipótese de estrutura que organiza as representações compartilhadas pelos participantes sobre os princípios da Computação. As explicações das relações entre os princípios e a Computação passaram por análise de conteúdo, sugerindo, no âmbito da Teoria das Representações Sociais (MOSCOVICI, 2003), quais objetos conferiam materialidade aos princípios no cotidiano dos estudantes e em quais outras ideias os princípios estavam ancorados. Observou-se, por exemplo, nas evocações sobre o termo indutor

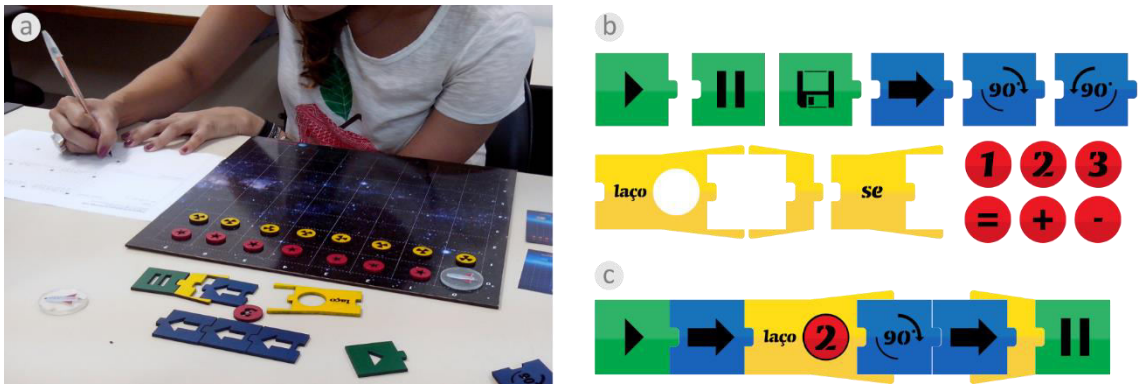
"computação", a centralidade de computadores como meios que materializam e realizam a Computação, mesmo com menções de uso de dispositivos como *smartphones* e videogames entre os participantes.

Os usos predominantes dos computadores informados pelos participantes da pesquisa foram categorizados como *instrumentais*, facilitadores de processos de Design que poderiam ser realizados por outros meios e métodos. Porém, as possibilidades decorrentes dos usos da Computação para pensar os processos de Design, como nas práticas de design computacional e generativo, ainda pareceram incipientes. Com base na hipótese da estrutura de representação dos princípios e na análise de conteúdo, elaborou-se uma estratégia pedagógica que considerou as práticas relatadas pelos participantes como oportunidades para introduzir novas práticas e representações em relação à Computação, superando os usos instrumentais.

No estudo das relações entre representações e práticas sociais, Abric (2001) argumentou que mudanças nas práticas de grupos poderiam promover transformações no conteúdo das representações compartilhadas por seus integrantes. Nesse sentido, a linguagem denominada RocketSocket (SANT'ANNA, 2014, cap. 5) foi elaborada como estratégia pedagógica para o ensino dos princípios da Computação para estudantes de Design, tendo três objetivos: 1) diversificar os objetos que materializam a Computação entre aqueles estudantes; 2) criar oportunidades para o tratamento dos princípios da Computação de forma concreta, com temáticas inicialmente mais familiares aos estudantes, sucedidas por exposições gradualmente mais gerais; 3) combinar a diversificação dos objetos que materializam a Computação e a abordagem dos princípios em processos de resolução de problemas que contribuam para a percepção da utilidade daqueles conhecimentos pelos estudantes em suas práticas de projeto.

A temática familiar escolhida para construir a primeira versão da linguagem RocketSocket foi um jogo de tabuleiro físico (Figura 1-a), com 64 casas (8 linhas por 8 colunas), em que um foguete deveria coletar estrelas sem colidir com asteroides. A opção pela execução física do jogo, e não como *software*, resultou da busca pela diversificação de objetos que materializam a Computação. Ao contrário de uma possível versão digital da linguagem, a suposição foi que todo o "processamento da informação" se daria na imaginação dos aprendizes, no diálogo com os colegas e professor durante a realização das atividades. Retomaremos esta suposição nas ponderações dos testes ao final desta seção.

Figura 1 – RocketSocket: a) tabuleiro, cartas e elementos, b) exemplos de primitivas e c) programa



Fonte: Elaborado pelo autor.

De volta à narrativa proposta para o jogo, o foguete foi dotado de um computador limitado, capaz de ser programado por instruções simples e de servir de modelo para a discussão dos princípios da Computação. Por exemplo, como o foguete computa o trajeto? Que tipo de

informação o foguete precisa armazenar durante a coleta? Como coletar estrelas com menos instruções? Como aproveitar trajetos entre tabuleiros semelhantes? Perguntas como essas poderiam deslocar os debates acerca dos princípios para outros cenários incomuns, porém não totalmente estranhos, de aplicação da Computação.

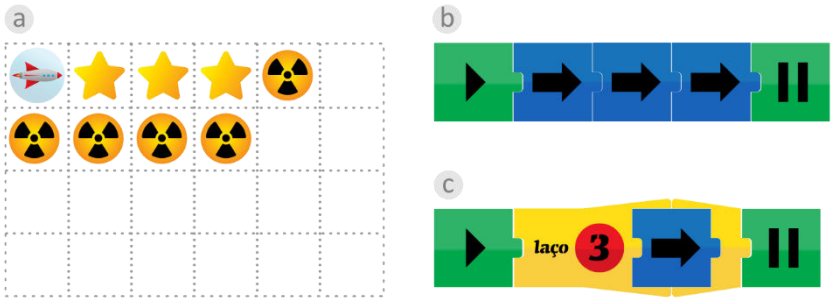
A cada jogada, uma carta sorteada definia a distribuição de estrelas e asteroides no tabuleiro, bem como a posição inicial do foguete. O controle dos movimentos do foguete sobre o tabuleiro durante a coleta era realizado pela combinação de elementos primitivos da linguagem (Figura 1-b), definidos a partir dos conceitos de Brennan e Resnick (2012), exceto paralelismo e eventos. Os elementos primitivos eram peças de encaixe que formavam sentenças correspondentes ao programa ou "plano de voo" que controlava o trajeto do foguete (Figura 1-c).

A iconografia e rotulagem dos elementos primitivos foram definidas considerando limitações físicas das peças, produzidas em MDF por corte a laser, e dos paradigmas imperativo e procedural de programação, como aplicados por Papert (1980) em LOGO. Nesta linguagem, aprendizes "conversam" com agentes computacionais que desejam controlar dando *comandos* a eles. Estes agentes têm condutas espaciais com as quais os aprendizes podem se identificar – andar em certa direção, girar tantos graus, escrever algo e assim por diante – fazendo com que imaginar ou realizar as próprias ações orientaria a escrita do programa de controle do agente.

As regras de composição das sentenças por encaixes igualmente aproveitaram experiências anteriores de linguagens de programação por blocos (cf. MALONEY et al., 2010), em que o desenho e cor de cada peça fornecem pistas sobre as combinações permitidas. Parte das escolhas das informações das peças foi baseada em ícones familiares, como *play* e *pause* para os comandos de ligar e desligar o motor do foguete, e o restante em rótulos não ambíguos em português, como "se", "senão" e "então" para os testes condicionais. O caso de "laço" reflete as dificuldades na tradução de termos de programação, pois seria possível traduzir esta estrutura de controle como "repetição", "repete" ou adotar a forma popular em inglês, *loop*. Em decisões desta natureza, optou-se sempre pela forma mais compacta e em português.

Por fim, o projeto dos elementos primitivos da linguagem buscou contemplar diferentes perfis de aprendizes (HAREL; PAPERT, 1991). Cada átomo da linguagem foi mapeado a ações de comando explícitas para o foguete, de modo que o programa capaz de movê-lo por três casas do tabuleiro (Figura 2-a) seria construído pela combinação da peça *liga*, três peças *move* e da peça *desliga* (Figura 2-b).

Figura 2 – RocketSocket: a) trajeto para coleta de três estrelas, b) sequência de movimentos e c) laço



Fonte: Elaborado pelo autor.

Para aprendizes sem contato prévio com programação, a elaboração da sequência de passos para coletar as estrelas sem colidir com asteroides (o *algoritmo*) consistia inicialmente nos encaixes corretos e ordenados das peças cujos comandos resultariam no trajeto desejado. À

medida em que os conceitos do PC fossem aprendidos, ou nos casos de estudantes com conhecimentos prévios de programação, a dependência das formas e cores das peças seriam gradualmente superadas. Os trajetos obtidos por sequências de instruções recorrentes poderiam ser transformados em laços (Figura 2-c), ou ainda realizados por procedimentos criados pelos aprendizes e que poderiam ser reutilizados em problemas similares.

A primeira versão da linguagem foi testada com dois aprendizes (SANT'ANNA, 2014, cap. 6), um com e outro sem experiência prévia em programação. Após apresentar a narrativa do jogo, os elementos primitivos da linguagem e suas regras de combinação, o pesquisador acompanhou a elaboração dos planos de voo para cinco cartas distintas. Os aprendizes, além de formarem as sentenças por meio dos encaixes das peças, preencheram mapas (ver Figura 1-a) com suas previsões sobre os estados do tabuleiro durante cada etapa da "execução" do plano de voo. Por se tratar de um jogo físico, o pesquisador conduzia a "execução" simulada passo a passo do programa, questionando o aprendiz em caso de inconsistências entre os estados planejados no mapa e aqueles obtidos no tabuleiro. Em caso de erros, o aprendiz elaborava um novo mapa com as previsões dos estados e refazia o programa.

Não há espaço no presente artigo para detalhar a análise qualitativa do funcionamento cognitivo dos aprendizes nos testes da linguagem. O método adotado foi a análise de caso (SAADA-ROBERT, 1997), conduzido por meio da reconstituição pormenorizada dos procedimentos de resolução dos problemas (projeto, meios e solução) pelos aprendizes ao longo do tempo, e baseada em índices sincrônicos (conjunto dos índices pertinentes em um dado momento) e diacrônicos (o mesmo índice considerado em momentos distintos da análise). As duas sessões foram registradas em vídeo e analisadas em conjunto com os mapas dos planos de voo, almejando identificar, em primeiro lugar, como os aprendizes mudavam o significado dos objetos e esquemas de ação empregados, considerando o objetivo de construir o trajeto do foguete adequado para uma dada disposição do tabuleiro.

No que tange à conduta dinâmica dos aprendizes, Saada-Robert (1997) argumentou que pode-se falar em: 1) *rotinas*, esquemas de ação pertinentes ativados para cada situação, familiares ao indivíduo e associados ao contexto imediato – p.ex. como apontar e mover o foguete em uma direção; 2) *primitivas*, esquemas de ação de mais mobilidade, selecionados também conforme seu significado em relação ao objetivo – p.ex. realizar um trajeto em "L" para coletar estrelas envolve direcionar e mover o foguete, mas o controle das ações está subordinado ao objetivo (percorrer o trajeto); e 3) *procedimentos*, unidades significativas resultantes da composição de várias primitivas, que são entendidas como blocos móveis e que poderão ser servir de rotina ou primitiva em outros contextos – p.ex. a compreensão de que partes do trajeto em "U" poderiam ser feitas a partir do trajeto em "L" seguido de sua inversão.

Em segundo lugar, o funcionamento cognitivo dos aprendizes na elaboração dos planos de voo foi investigado quanto às funções de causalidade e finalidade das representações elaboradas sobre o problema (BLANCHET, 1997). A função causal, de sentido ascendente, está relacionada às transformações percebidas como possíveis, legítimas ou válidas de acordo com o sistema de regras que governa a situação-problema. A especificação de instruções locais para a coleta de estrelas, passo a passo, até que se conclua o tabuleiro, ilustra este tipo de atividade *causal-final*. A função final, de tipo descendente, consiste no encadeamento das etapas a serem seguidas, assumindo as transformações possíveis e subordinadas ao objetivo de se alcançar a solução do problema. A concepção global do algoritmo do plano de voo e sua decomposição nos termos de sequências e abstrações das instruções da linguagem, representam o tipo de atividade *final-causal* investigada.

Os resultados da análise de casos sugeriram que a linguagem tinha potencial para exercer o papel de "ferramenta para pensar" princípios e conceitos da Computação. Em certo momento das sessões (SANT'ANNA, 2014, p.146 e seg; p. 155 e seg.), os aprendizes foram convidados a elaborar e utilizar sua própria linguagem de controle do foguete, indicando o entendimento das funções daqueles conceitos para a resolução do problema de coleta das estrelas. No sentido causal-funcional, os aprendizes propuseram átomos mínimos para as suas linguagens, capazes de transformar localmente a situação-problema. No sentido funcional-causal, ponderaram sobre os requisitos da linguagem para que fosse capaz de controlar o foguete, de modo geral, e na coleta das estrelas em distribuições específicas do tabuleiro.

A primeira versão de RocketSocket foi planejada para uso combinado a ambientes como Processing e similares. Estes, além de funcionarem em computadores e dispositivos móveis, possuem recursos de autoria multimídia (síntese de imagens, sons, animações e interatividade) e implementam eventos e paralelismo, conceitos não disponíveis na proposta. A linguagem seria destinada, portanto, às discussões iniciais sobre conceitos da Computação e pensamento computacional em disciplinas de cursos de Design, facilitando a introdução posterior de ambientes e linguagens mais avançadas no conteúdo programático.

A despeito das intenções preliminares para o uso de RocketSocket nas disciplinas, o autor desenvolveu protótipos digitais da linguagem ainda na fase final da tese (SANT'ANNA, 2014, p. 165). O primeiro protótipo foi desenvolvido para dispositivos móveis, aventando a possiblidade de oferecer correções automáticas dos planos de voo e permitindo acesso individual aos tabuleiros-problema. Isto seria útil em turmas numerosas e contextos em que os tabuleiros físicos estivessem indisponíveis. Vale destacar a introdução da versão do tipo *script* para a linguagem, com semântica equivalente às peças de encaixe. A conversa entre aprendiz e foguete se dava pela digitação de comandos como "liga", "move" e "gira 90".

4 A disciplina Design Computacional (DC)

O Curso de graduação em Design da Ufes foi fundado em 1998 e teve sua primeira reforma curricular completa implantada em 2020[1]. Ao longo das duas primeiras décadas de existência, quatro disciplinas obrigatórias ofereciam oportunidades para abordar conteúdos relacionados a design computacional e generativo: Computação Gráfica I (3º período), Computação Gráfica II (4º), Multimídia I (7º) e Multimídia II (8º). Na fundação do curso, essas disciplinas desempenhavam funções mais próximas do conceito de *computação no design*, citado na introdução deste artigo. A partir do final dos anos 2000, projetos que empregavam programas de editoração gráfica e autoria multimídia (inclusive CD-ROMs interativos) perderam espaço para a construção de interfaces digitais utilizando tecnologias Web (HTML, CSS e JavaScript), ferramentas de prototipagem e outras plataformas de design de interação.

O autor do presente relato foi responsável por várias turmas dessas disciplinas durante a vigência da grade curricular anterior, incluindo o período de desenvolvimento da pesquisa que gerou a linguagem RocketSocket (2010-2014). Parte das experiências pedagógicas deste ínterim aspiravam superar os usos instrumentais da Computação nas práticas de projeto – adoção de ambientes para design de jogos baseados em programação simplificada, oficinas de introdução ao pensamento computacional abertas à comunidade, mapeamento de ambientes e linguagens de programação para designers e artistas visuais (SANT'ANNA, 2010; SANT'ANNA et al., 2012; SANT'ANNA; NEVES, 2012).

[1] Disponível em <https://design.ufes.br/historia-do-curso>. Acesso: 14 abr. 2022.

Com a retomada das discussões sobre a reforma curricular do Curso de Design da Ufes em 2014, e sob forte influência dos resultados da tese de doutorado recém-defendida, a disciplina de "Design Computacional" (DC) foi planejada pelo autor deste artigo (Quadro 1). A revisão de literatura da tese ofereceu subsídios para a definição da ementa, conteúdo programático e bibliografia. Recomendações da Sociedade Brasileira de Computação (SBC)[2] e Computer Science Teachers Association (CSTA, EUA)[3] para ensino do pensamento computacional na educação básica foram consideradas na elaboração da disciplina.

Quadro 1 – Programa da disciplina Design Computacional.

Unidade	**Semana**	**Conteúdo**	**Bibliografia sugerida**
I - Pensamento Computacional e Design	1	Introdução ao Pensamento Computacional	Wing (2006), Denning e Martell (2015)
	2	Apresentação dos ambientes e linguagens	Reas e Fry (2007), Marji (2014), Iepsen (2018), Ierusalimschy (2014)
	3	Princípios da Computação	Denning e Martell (2015)
	4	Abstração e Automação	Denning e Martell (2015), Wing (2008)
	5	Conceitos, práticas e perspectivas do Pensamento Computacional	Brennan e Resnick (2012)
II – Aplicações	6 a 11	Conceitos: sequências, laços, condicionais, dados, operadores, eventos e paralelismo	Brennan e Resnick (2012), Marji (2014), Reas e Fry (2007), Iepsen (2018)
	12 a 14	Práticas: iterações, abstração e modularização, testes e depurações, remixes	Brennan e Resnick (2012)
	15	Revisão e discussão de perspectivas do PC	

Fonte: Elaborado pelo autor.

No projeto novo pedagógico[4], a disciplina de DC foi definida como obrigatória, integrando o 1º período do ciclo básico, com carga horária total de 60h (4h/semana), 30 vagas para estudantes ingressantes e a seguinte ementa: "Relações entre Pensamento Computacional e Design. Princípios da Computação aplicados às Artes, Design e Arquitetura. Computação Criativa". Os objetivos definidos para a disciplina foram: 1) oferecer oportunidades de aprendizado dos princípios da Computação; 2) discutir conceitos, práticas e perspectivas da Computação no Design; 3) aplicar conceitos da Computação a tarefas de domínio específico; 4) apresentar

[2] Disponível em <https://www.sbc.org.br/documentos-da-sbc/send/203-educacao-basica/1220-bncc-em-itinerario-informativo-computacao-2>. Acesso: 14 abr. 2022.
[3] Disponível em <https://www.csteachers.org/page/reports>. Acesso: 14 abr. 2022.
[4] Disponível em <https://design.ufes.br/organizacao-curricular-ppc-2020>. Acesso: 14 abr. 2022.

ambientes e linguagens de programação em contextos de projeto. Quanto ao último objetivo, a proposta da disciplina previa atividades com a versão física da linguagem RocketSocket, além de Processing, Scratch, JavaScript e Lua.

Contudo, antes mesmo da implantação do novo projeto pedagógico, o mundo foi impactado pela pandemia de Covid-19. O semestre letivo iniciado em 02 de março de 2020, e interrompido 16 dias depois, realizaria a experiência-piloto do conteúdo programático de DC, ainda na disciplina de Computação Gráfica II do currículo anterior. Havia expectativas factíveis de início da nova grade curricular no segundo semestre daquele ano, de maneira que diversas disciplinas do curso optaram por testar e avaliar as mudanças iminentes.

Na Ufes, a modalidade de ensino remoto emergencial temporário, denominada Earte, teve início em setembro de 2020. A instituição aprovou a oferta de disciplinas por meio de uma combinação de cargas horárias síncronas (até 25% do total semanal, via conferências Web) e assíncronas. Ponderando sobre as dificuldades de ministrar disciplinas novas em um período de tantas incertezas e limitações, o autor deste relato decidiu postergar os testes da nova disciplina para o ano letivo seguinte, iniciado em junho de 2021.

Por razões óbvias, a utilização da versão física da linguagem RocketSocket seria impossível no Earte. A utilização de várias linguagens representava outro problema, uma vez que haveria diferenças entre as configurações dos computadores pessoais dos estudantes e das habilidades necessárias para baixar e instalar cada ambiente. As únicas certezas para os professores durante o período eram a capacidade de conexão à Internet e o acesso das turmas às aulas e materiais didáticos por meio de navegadores Web, via dispositivos móveis ou computadores. A Ufes forneceu chips de Internet e auxílio financeiro para estudantes que não dispunham de equipamentos ou conexão em casa, visando garantir condições mínimas para que todo o corpo discente pudesse cursar as disciplinas.

5 A implementação Web de RocketSocket

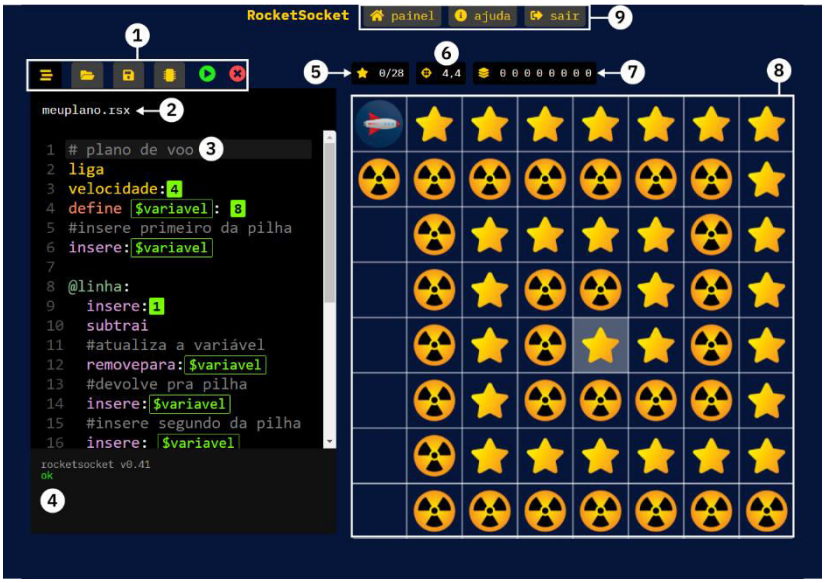
O conjunto das dificuldades do Earte enunciadas, somado à demanda de ofertar a disciplina de DC pela primeira vez, levaram o autor a retomar o desenvolvimento da versão digital da linguagem RocketSocket. Mais do que implementar o tabuleiro digital e as cartas com a disposição de estrelas e asteroides no tabuleiro, a versão para o Earte tinha os seguintes requisitos: 1) ser acessível por navegadores de computadores e dispositivos móveis com diferentes capacidades de memória e processamento; 2) permitir a edição dos planos de voo diretamente no navegador, incluindo armazenamento, recuperação e compartilhamento; 3) verificar o sucesso dos planos de voo para a coleta das estrelas de modo automático, dispensando a conferência síncrona dos programas pelo professor; 4) disponibilizar todo o material para a aprendizagem da linguagem e uso do ambiente de modo integrado – manual, documentação e exemplos de código; 5) substituir as atividades que seriam realizadas nos demais ambientes e linguagens, diversificando o tipo e complexidade dos projetos que exploram os princípios da Computação e os conceitos, práticas e perspectivas do pensamento computacional (PC); 6) apoiar a realização da ementa, objetivos e conteúdo programático da disciplina (Quadro 1).

A implementação Web de RocketSocket utiliza tecnologias Web (HTML5, CSS e JavaScript) na parte frontal (*front-end*), *scripts* em linguagem PHP na retaguarda (*back-end*), banco de dados MySQL e hospedagem em servidor Apache para sistema operacional Linux. A geração de

gráficos dinâmicos via JavaScript e HTML5 Canvas pelo ambiente é baseada em padrões suportados por navegadores desde 2011[5].

A interface (Figura 3) é composta por duas áreas: à esquerda, abas para a recuperação, salvamento e compartilhamento, execução e interrupção da execução dos planos de voo (1), identificação do plano de voo (2), editor de código (3), console para mensagens de alerta e erros (4). À direita, o total de estrelas coletadas e remanescentes (5), as coordenadas (coluna e linha) da casa do tabuleiro selecionada pelo cursor do mouse (6), o mostrador da pilha do processador do foguete (7), e o tabuleiro digital (8), que pode ter entre 16 e 32 casas. Na parte superior da interface (9), há *links* de acesso ao painel com todos os planos do usuário, à documentação e ao encerramento da sessão atual. As instruções dos planos de voo em códigos *script* implementam os comandos das peças de encaixe da primeira versão.

Figura 3 – Visão do ambiente RocketSocket no navegador (modo coleta).



Fonte: Elaborado pelo autor.

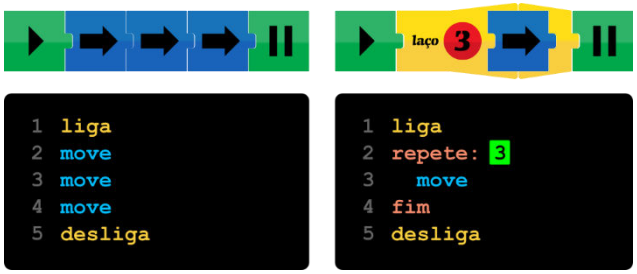
No momento de finalização deste artigo (abril/2022), há 23 instruções básicas na linguagem, organizadas em sete grupos de acordo com suas funções.

- *Controle*: partida, parada e velocidade do motor do foguete;
- *Voo*: mover, girar e teleportar o foguete;
- *Laços*: controle de repetições de blocos de instruções;
- *Condicionais:* testes se o valor fornecido como argumento é maior, menor ou igual ao valor no topo da pilha;
- *Abstrações:* comentários, criação e execução de procedimentos, definição de variáveis, criação de bibliotecas do usuário;
- *Pilha:* inserção, sorteio e remoção de valores na pilha;
- *Operações*: soma, subtração, multiplicação e divisão de valores na pilha.

[5] Disponível em <https://caniuse.com/canvas>. Acesso: 14 abr. 2022.

A implementação *script* da linguagem segue o paradigma imperativo e procedural de LOGO, nomeando as instruções nos termos de comandos que controlam a conduta do foguete. Os encaixes das peças em sentenças foram substituídos pela digitação de instruções em linhas, informando argumentos adicionais quando necessário. A Figura 4 compara planos de voo em sentenças com peças de encaixe e *scripts* da versão Web. Como na versão física, o editor codifica instruções por cor, facilitando a leitura do código e identificação dos grupos de funções.

Figura 4 – Comparação de dois planos de voo na versão física e Web



Fonte: Elaborado pelo autor.

O computador "imaginado" do foguete na versão original foi implementado como uma máquina virtual que pode ser inspecionada na aba CPU (Figura 3, área 1). Durante a execução dos planos de voo, é possível acompanhar em tempo real o processamento das instruções pelo "processador fictício", verificar as variáveis declaradas e o uso de memória pelo programa. A pilha (Figura 3, área 7) consiste em estrutura de dados do tipo *LIFO*[6], usada para realizar operações aritméticas, testes condicionais e endereçamentos da instrução "teleporta" da linguagem. Esta instrução permite realizar saltos instantâneos do foguete para qualquer casa vazia do tabuleiro, mediante indicação do endereço de destino (0-63 no tabuleiro de 8x8).

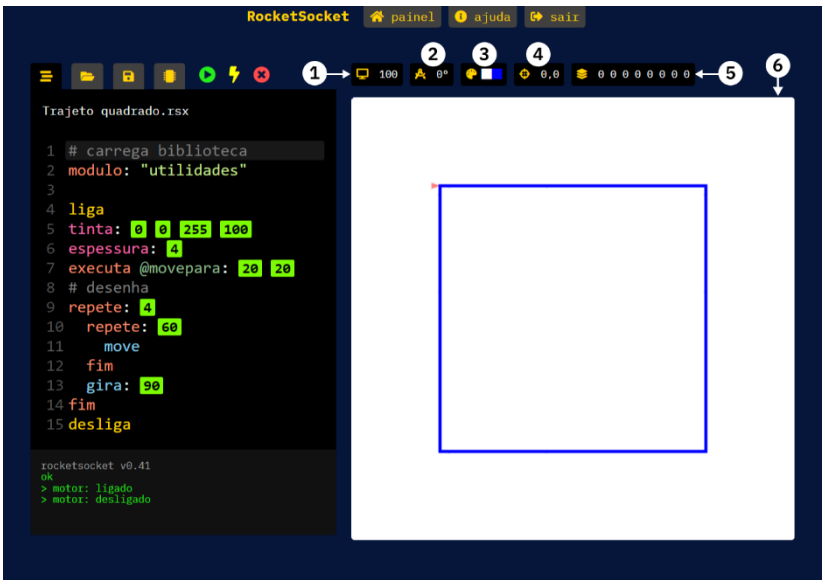
O modo inicial do ambiente corresponde à coleta de estrelas, como na versão física. Todavia, considerando o requisito de substituir outras linguagens durante o Earte, a versão Web de RocketSocket conta com um segundo modo de uso, destinado a aplicações de design computacional e generativo. Batizado de "Papert", este modo foi inspirado na "geometria de tartaruga" desenvolvida pelo autor de mesmo nome e colaboradores na linguagem LOGO (PAPERT, 1980). Em termos práticos, este modo substitui o foguete por uma caneta e o tabuleiro por uma área de desenho em branco. As mesmas instruções de controle dos movimentos do foguete se aplicam à caneta, com a diferença de que esta "risca" a área de desenho enquanto se move. Em vez de se deslocar entre as casas do tabuleiro, a caneta marca pontos (*pixels*) endereçáveis em áreas de desenho de 33x33 a 580x580 pontos, combinando as instruções básicas da linguagem a outras 11 específicas para desenhar:

- *Área*: definição do tamanho da área de desenho e exibição ou ocultação da grade;
- *Caneta*: abaixar (riscar durante o movimento) e levantar a caneta (não riscar), definir espessura do traçado, mostrar e esconder a posição da caneta;
- *Cores*: definição da mistura RGB do fundo da área de desenho e da tinta da caneta;
- *Tipografia*: seleção da família tipográfica e escrita na área de desenho.

A interface (Figura 5) apresenta poucas alterações em relação ao modo coleta: mostrador das dimensões da área de desenho (1), ângulo atual da caneta (2), cor de fundo e cor da tinta atuais (3), posição do ponteiro do mouse na área de desenho (4), mostrador da pilha (5) e área de desenho (6). A caneta é representada por um triângulo isósceles transparente que aponta para a direção de movimento, assim como o foguete.

[6] A operação da pilha do tipo LIFO requer que o último valor inserido seja o primeiro a ser removido (*Last In, First Out)*, permitindo a realização de operações aritméticas com pares de valores situados no topo. A manipulação de valores na pilha é fundamental para a discussão de variáveis e operadores.

Figura 5 – Visão do modo “Papert” de RocketSocket



Fonte: Elaborado pelo autor.

O modo “Papert” parte do pressuposto de que aprendizes seriam capazes de realizar, no mínimo, transferências próximas dos conhecimentos construídos nas atividades de coleta de estrelas para as atividades de desenho. Este tipo de transferência ocorreria entre contextos percebidos como semelhantes pelos aprendizes e que demandariam habilidades similares (cf. PERKINS; SALOMON, 1992). Por exemplo, os procedimentos (no sentido de Saada-Robert, 1997) utilizados para desenhar o quadrado na Figura 5 poderiam ser construídos a partir de primitivas ou procedimentos pertinentes para programar trajetos isomórficos do foguete.

Já a ocorrência de transferências distantes, nas quais aprendizes transfeririam conhecimentos a contextos e habilidades distintos, estão diretamente relacionados aos potenciais benefícios do desenvolvimento do PC. Enquanto habilidades fundamentais para todos, que permitiriam explicar e interpretar o mundo nos termos de processos de informação (cf. WING, 2006; DENNING; TEDRE, 2021), o PC extrapola problemas de coletar de estrelas e de desenhar com código. A promessa do PC é que o aprendizado dos princípios da Computação ampliaria as capacidades analíticas e projetuais em muitos domínios e, no caso de designers, pode-se sugerir que qualificaria e diversificaria as práticas de projeto.

A investigação das possíveis contribuições de RocketSocket para se atingir a promessa do PC ainda é incipiente e exigirá a condução de análises de casos como aquelas das sessões de testes da versão física da linguagem. Não obstante, os exemplos a seguir, selecionados entre as atividades da disciplina, fornecem indícios plausíveis dos processos de transferência em curso.

6 Entre foguetes, estrelas e canetas

Nos semestres letivos 2021/1 e 2021/2, 150 estudantes cursaram a disciplina na modalidade Earte. Este quantitativo foi dividido em três turmas, sendo uma em 2021/1 e duas em 2021/2. Todas as atividades eram individuais e efetuadas assincronamente, com correção automática das objetivas e avaliação das discursivas. Monitores apoiaram a disciplina tirando dúvidas fora dos horários de aulas e verificando as correções automáticas. Em 2021/1, os monitores eram alunos que haviam cursado Computação Gráfica II anteriormente. Em 2021/2, houve monitores na mesma condição e outros que cursaram a primeira turma de DC.

Até a quinta semana (ver Quadro 1), as turmas realizaram atividades objetivas e discursivas, subsidiando os debates dos encontros síncronos. Na Unidade I, dez atividades com RocketSocket abordaram conceitos, práticas e perspectivas do PC por meio da coleta de estrelas. A partir da Unidade II, as atividades no modo "Papert" tinham apenas temas sugeridos (Figura 6): iniciais dos nomes dos estudantes (a), reprodução de desenho de escolha livre (b), reprodução de identidades visuais brasileiras (c), criação de padrões para ladrilhos hidráulicos (d), reprodução de pictogramas de jogos olímpicos (e) e criação de cartazes de jogos olímpicos (f). As atividades foram acompanhadas de referências audiovisuais, demonstrações de códigos avançados e resolução de dúvidas nos encontros síncronos. Cerca de 2600 programas, entre coletas de estrelas e desenhos, foram produzidos nos dois semestres letivos.

Figura 6 – Exemplos de atividades de desenho livre de 2021/1 e 2021/2

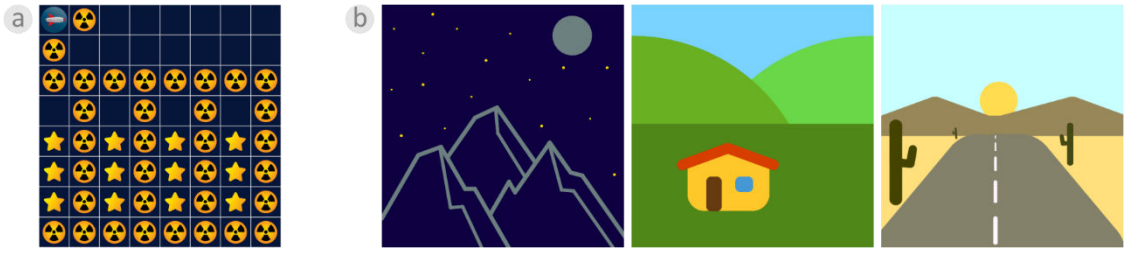
Fontes: Estudantes a) S.M.F., b) V.H.A.F., c) B.F.I., d) C.S.F.C., e) S.A.O.L., f) D.N.S.

Para ilustrar o percurso das turmas na interação com a linguagem e o conteúdo programático, consideraremos a seguir a transição do sentido causal-final para o final-causal na construção de algumas atividades. Em situações com muitas estrelas (p.ex. Figura 3), esquemas de ação predominantemente causais tendem a ser tediosos, marcados pelo sequenciamento de blocos de instruções "move" e "gira" para coletar cada estrela individualmente, até a finalização do tabuleiro. Analiticamente (cf. SAADA-ROBERT, 1997, p. 161), a subordinação dos esquemas ao objetivo global da fase promoveria o reconhecimento de padrões de movimentos (rotinas, primitivas ou procedimentos), capazes de solucionar etapas ou todo o tabuleiro. Atentando-se à progressão do conteúdo programático, esses padrões transformariam sequências de instruções em laços, depois em procedimentos dos planos, para finalmente atingirem a condição de procedimentos em módulos criados pelo aprendiz e reutilizados em diferentes atividades.

Em fases com "becos sem saída", em que o foguete está preso entre asteroides (Figura 7-a), ou em desenhos com diferentes camadas (Figura 7-b), a instrução "teleporta" tem papel essencial. Por meio dela, aprendizes podem deslocar o foguete ou caneta instantaneamente entre casas do tabuleiro e coordenadas da área de desenho, sem precisar percorrer a distância que separa a posição atual e a desejada. Esta operação exige a inserção do endereço destino na pilha, representado por um número específico obtido por dois métodos: 1) contagem manual do número da casa desejada, iniciando em zero (0) no canto superior esquerdo do tabuleiro e avançando pelas colunas de cada linha até atingir a última casa no canto inferior

direito; ou 2) algoritmos que calculem o endereço da casa em função das coordenadas *x*, *y* e número de colunas (ou largura da área de desenho em pontos)[7].

Figura 7 – Exemplos de a) tabuleiro com "becos sem saída" e b) desenhos com camadas



Fontes: a) elaborado pelo autor e b) estudantes D.R.M., S.M.F. (2021/1) e G.S.B. (2021/2)

A contagem manual produz o mesmo resultado do algoritmo, embora seja mais trabalhosa em tabuleiros com muitas casas e impraticável em áreas de desenho. Por exemplo, tabuleiros de 32x32 casas têm 1024 endereços possíveis e desenhos de 100x100 pontos têm 10 mil. Um único salto durante o plano de voo pode ser realizado pelo bloco de código indicado na Figura 8-a, mas a realização de sucessivos saltos incentivaria a construção de um procedimento capaz de calcular o endereço de destino a partir das coordenadas *x* e *y* (Figura 8-b).

Figura 8 – Progressão de sequências de instruções a procedimentos

```
# Figura 8-a

# coords: 50,50
# d=50+(50*100)
insere: 5050
teleporta
```

```
# Figura 8-b

# algoritmo
@movepara: $x $y
  insere: 100
  insere: $y
  multiplica
  insere: $x
  soma
  teleporta
volta
```

```
# Figura 8-c

# linha+ângulo
@linha: $a $m
  gira: $a
  repete: $m
    move
  fim
volta
```

```
# Figura 8-d

# arco
@arco: $a $m
   repete: $m
    move
    gira: $a
  fim
volta
```

Fonte: Elaborado pelo autor a partir das atividades (2021/1 e 2021/2)

Este procedimento, rotulado "movepara", foi apresentado pelo professor nos encontros síncronos para ilustrar abstrações na linguagem. Em seguida, variações dele surgiram nas atividades entregues pela turma, acompanhadas de construções similares para desenhar linhas em ângulos específicos (Figura 8-c) e arcos (8-d). Mesmo assim, houve estudantes que mantiveram a prática de repetir o bloco de código da Figura 8-a diversas vezes no plano de voo, indicando preferências por esquemas de ação causais-finais.

Com o avanço da disciplina em 2021/1, o professor apresentou a galeria dos projetos, onde era possível visualizar desenhos dos colegas e respectivos códigos-fonte, desde que compartilhados publicamente em opção específica do ambiente. Esta possibilidade incentivou a exploração e troca de trechos de código para a solução de problemas semelhantes, gerando a adoção massiva de alguns procedimentos pelas turmas. Em 2021/2, uma estudante descobriu o procedimento "sanfona" na galeria (Figura 9-a), criado por uma estudante de outra turma com o objetivo de variar a espessura do traçado no movimento da caneta. A

[7] Formalmente: endereço = x + (y * largura).

atividade de reprodução dos pictogramas de jogos olímpicos demandou esta solução para representar a variação nas proporções de membros inferiores e superiores no desenho (Figura 9-b).

Figura 9 – Procedimento “sanfona” e aplicações ao desenho de pictogramas

Fontes: a) procedimento “sanfona” da estudante P.L.G.; b) estudantes D.N.S. e M.C.R. (2021/2)

A estudante que identificou a pertinência daquele procedimento revelou a descoberta durante os encontros síncronos e a solução se espalhou pelos planos de voo e bibliotecas criadas pelos colegas. Isto foi observado mesmo entre aqueles que optaram por repetir a sequência de instruções quando necessário, sem abstraí-la como procedimento. Casos similares foram observados com procedimentos para preencher o fundo dos cartazes olímpicos com frases sobre as modalidades esportivas representadas (p.ex., Figura 9-b).

7 Discussão

A abordagem pedagógica RocketSocket foi elaborada com o intuito de satisfazer demandas de aprendizagem de conceitos da Computação por estudantes de Design e da aplicação deles a práticas de design computacional e generativo. Iniciativas anteriores sugeriram que este processo seria condicionado, por um lado, pela necessidade de familiarização dos aprendizes com ambientes, linguagens e práticas de programação e, por outro, pelo planejamento de situações didáticas que contribuíssem para a transferência dos conhecimentos aprendidos entre diferentes situações de projeto.

As condições do ensino remoto, decorrentes da pandemia de Covid-19, complexificaram os requisitos originais da abordagem: acesso simplificado via diferentes configurações de dispositivos; edição, execução e correção de códigos diretamente no navegador; disponibilização de ambiente integrado, incluindo documentação e materiais de apoio; diversificação das atividades, somando o modo de design generativo ao de coleta de estrelas. Além disso, havia o compromisso com o cumprimento da ementa, objetivos e conteúdo programático definidos para a disciplina no novo projeto pedagógico.

Os exemplos selecionados neste relato sugerem ocorrências de transferências próximas de conceitos da Computação entre os modos de coleta e “Papert”, e indícios de transferências distantes na elaboração dos desenhos livres. Quanto a esta hipótese, a ser investigada em estudos subsequentes, os esquemas de ação ativados pelos estudantes pareciam estar sob o controle do desenho idealizado, cada vez menos associados ao funcionamento do ambiente e regras da linguagem. Em outras palavras, as preocupações iniciais dos aprendizes quanto às instruções, sintaxe, deslocamento da caneta e mecânica dos procedimentos de desenho tornaram-se secundários frente ao desenvolvimento gradual de seus objetivos criativos.

As possibilidades de compartilhar e trocar soluções para problemas similares pareceram desempenhar papel importante na redução de barreiras para alcançar tais objetivos, talvez em razão das condições adversas do ensino remoto. O retorno ao ensino presencial em 2022/1 introduzirá, entre outras variáveis, dinâmicas de ensino e aprendizagem síncronas que podem não repetir os resultados observados nesta primeira experiência.

Finalmente, restará avaliar se os achados acerca da conduta dos aprendizes na versão original, de natureza física, permanecem válidos na versão Web, especialmente quanto aos objetivos de diversificação dos objetos que conferem materialidade à Computação.

8 Referências

ABRIC, J. C. Central System, Peripheral System: their functions and role in the dynamics of Social Representations. **Papers on Social Representations**, v. 2, n. 2, p. 75–78, 1993.

ABRIC, J. C. Prácticas Sociales, Representaciones Sociales. Em: ABRIC, J. C. (Ed.). . **Prácticas sociales y representaciones**. D.F. México: Ediciones Coyoacán, 2001.

AZEVEDO, E.; CONCI, A.; LETA, F. R. **Computação Gráfica: Geração de Imagens**. Rio de Janeiro: Campus, 2008. v. 1

BLANCHET, A. As unidades procedimentais, causais e teleonômicas no estudo dos processos cognitivos. Em: **O percurso das descobertas das crianças: pesquisas sobre as microgêneses cognitivas**. Lisboa: Instituto Piaget, 1997.

BRENNAN, K.; RESNICK, M. **New frameworks for studying and assessing the development of computational thinking**. Proceedings of the 2012 annual meeting of the American educational research association, Vancouver, Canada. **Anais**...2012.

BUHAMDAN, S.; ALWISY, A.; BOUFERGUENE, A. Generative systems in the architecture, engineering and construction industry: A systematic review and analysis. **International Journal of Architectural Computing**, v. 19, n. 3, p. 226–249, 1 set. 2021.

DENNING, P. J. **Great Principles in Computing Curricula**. Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education. **Anais**...: SIGCSE '04.New York, NY, USA: ACM, 2004. Disponível em: <http://doi.acm.org/10.1145/971300.971303>

DENNING, P. J.; MARTELL, C. H. **Great Principles of Computing**. Cambridge: MIT Press, 2015.

DENNING, P. J.; TEDRE, M. Computational Thinking: A disciplinary perspective. **Informatics in Education**, v. 20, n. 1, p. 361–390, 2021.

HAREL, I.; PAPERT, S. **Constructionism**. New York: Ablex Publishing, 1991.

IEPSEN, E. F. **Lógica de Programação e Algoritmos com JavaScript**. São Paulo: Novatec, 2018.

IERUSALIMSCHY, R. **Programming in Lua**. 2. ed. Rio de Janeiro: Lua.org, 2014.

KNUTH, D. **The Art of Computer Programming**. 3. ed. Massachusetts: Addison-Wesley, 1997. v. 1

MALONEY, J. et al. The scratch programming language and environment. **ACM Transactions on Computing Education (TOCE)**, v. 10, n. 4, p. 16, 2010.

MARJI, M. **Aprenda a Programar com Scratch**. São Paulo: Novatec, 2014.

MOSCOVICI, S. **Representações Sociais: investigações em psicologia social**. Petrópolis: Vozes, 2003.

PAPERT, S. **On making a theorem for a child**. Proceedings of the ACM annual conference-

Volume 1. **Anais**...ACM, 1972.

PAPERT, S. **Mindstorms: Children, computers, and powerful ideas**. New York: Basic Books, 1980.

PERKINS, D. N.; SALOMON, G. Transfer of learning. Em: **International Encyclopedia of Education**. Oxford: Pergamon Press, 1992. v. 2p. 6452–6457.

REAS, C.; FRY, B. **Processing: A Programming Handbook for Visual Designers and Artists**. Cambridge: MIT Press, 2007.

SÁ, C. P. **Núcleo Central das Representações Sociais.** Rio de Janeiro: Vozes, 1996.

SAADA-ROBERT, M. A construção microgenética de um esquema elementar. Em: **O percurso das descobertas da criança: pesquisa sobre as microgêneses cognitivas**. Lisboa: Instituto Piaget, 1997.

SANT'ANNA, H. C. **A construção de narrativas multimídia na perspectiva da Zona de Desenvolvimento Proximal de Vygotsky e da Pós-Produção de Bourriaud como apoio ao processo de aprendizagem digital**. Anais Eletrônicos do 3º Simpósio Hipertexto e Tecnologias na Educação. **Anais**... Em: 3º SIMPÓSIO HIPERTEXTO E TECNOLOGIAS NA EDUCAÇÃO. Recife: NEHTE, 2010.

SANT'ANNA, H. C. et al. **Da Arte Generativa ao Pensamento Computacional - Uma análise comparativa das plataformas de aprendizagem**. Anais do 11º Encontro Internacional de Arte e Tecnologia. **Anais**... Em: 11º ART. Brasília: Departamento de Artes Visuais / UnB, 2012.

SANT'ANNA, H. C. **Ação, Computação, Representação: uma investigação psicogenética sobre o desenvolvimento do Pensamento Computacional**. Doutorado em Psicologia—Vitória: Universidade Federal do Espírito Santo, 2014.

SANT'ANNA, H. C. **Revisão crítica das aplicações de Aprendizado de Máquina no Design Visual: bases teóricas, desempenho dos modelos e novos paradigmas de projeto**. Anais do SIIMI/2019 VI Simpósio Internacional de Inovação em Mídias Interativas. **Anais**... Em: SIIMI/2019 VI SIMPÓSIO INTERNACIONAL DE INOVAÇÃO EM MÍDIAS INTERATIVAS. Buenos Aires: Medialab/UFG, 2019.

SANT'ANNA, H. C.; NEVES, V. B. **Scratch Day UFES: oficina itinerante de introdução à programação para professores**. Anais Eletrônicos. **Anais**... Em: 4º SIMPÓSIO HIPERTEXTO E TECNOLOGIAS NA EDUCAÇÃO. Recife: NEHTE, 2012.

WING, J. M. Computational Thinking. **Communications of the ACM**, v. 49, n. 3, p. 33–35, mar. 2006.

WING, J. M. Computational thinking and thinking about computing. **Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences**, v. 366, n. 1881, p. 3717–3725, 28 out. 2008.

9 Agradecimentos

A linguagem RocketSocket é resultado do apoio incondicional da minha orientadora de doutorado e supervisora de pós-doutorado, Profª. Drª. Maria Cristina Smith Menandro.