Dominoops: Um Jogo de Cartas para Ensino-Aprendizagem de Conceitos de Programação Orientada a Objetos

Antônio Lucas Ferreira de Souza
Dep. de Ciência da Computação
Universidade Federal da Bahia
Salvador, Brasil
alfsouza708@gmail.com

Rodrigo R. G. e Souza
Dep. de Ciência da Computação
Universidade Federal da Bahia
Salvador, Brasil
rodrigorgs@ufba.br

***Resumo*—A aprendizagem da programação orientada a objetos (POO) é um desafio para estudantes, uma vez que envolve uma mudança de paradigma e diversos novos conceitos. Apesar de jogos terem sido usados com sucesso no ensino de programação em geral, existem poucas iniciativas especificamente para POO. Para preencher esta lacuna, desenvolvemos o Dominoops, um jogo de cartas que contempla conceitos de POO, em particular associação, herança e compatibilidade de tipos, e avaliamos o jogo através de sessões de *playtesting* com estudantes e profissionais. Os participantes consideraram o jogo divertido e defenderam seu uso em sala de aula, mas demonstraram dificuldade de entender as regras apenas com a leitura do manual.**

***Index Terms*—programação orientada a objetos, jogo de cartas, ensino-aprendizagem**

I. Introdução

A programação orientada a objetos é um paradigma de linguagem de programação que possui raízes na linguagem Simula e é amplamente utilizado e ensinado em cursos da área de computação [1]. O aprendizado desse paradigma representa um desafio, uma vez que o estudante precisa entender e aplicar corretamente conceitos como herança [2], polimorfismo [3], construtores, modificadores de acesso [4], dentre outros.

Jogos têm sido usados para apoiar o processo de ensino-aprendizagem em diversos níveis de instrução, sobretudo por sua capacidade de motivar aprendizes [5]. Na área da computação, há numerosos jogos voltados para o ensino inicial de programação, englobando conceitos como sequenciamento, desvios condicionais, repetição e abstração procedural, mas poucos jogos voltados especificamente para a programação orientada a objetos, como pode ser observado em um mapeamento sistemático recente [6].

Neste artigo apresentamos o Dominoops, um jogo de cartas que apoia o ensino-aprendizagem de alguns conceitos essenciais da programação orientada a objetos. Especificamente, o jogo trata de conceitos como herança, associação entre objetos e compatibilidade de tipos.

O jogo foi avaliado em três sessões de *playtesting*, incluindo como participantes estudantes e profissionais da área da computação e de outras áreas. Em geral, os participantes consideraram o jogo divertido e promissor no ensino de programação orientada a objetos e, ao mesmo tempo, indicaram oportunidades de melhoria para o jogo.

II. Trabalhos relacionados

Greenfoot [7], Alice [8] e MUPPETS [9] são ambientes de desenvolvimento com finalidade educacional, no qual estudantes criam jogos usando conceitos de orientação a objetos. Uma desvantagem dessa abordagem é o alto investimento de tempo por parte do estudante, que precisa se familiarizar com um ambiente relativamente complexo porém limitado, e portanto inadequado para uso profissional [10].

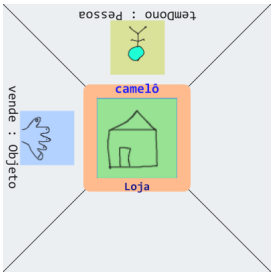
ZTECH é um jogo estilo RPG que testa o conhecimento de estudantes sobre estruturas básicas de programação e conceitos de programação orientada a objetos [11]. Ele consiste de *minigames* que incluem quizzes e quebra-cabeça de deslizar. Diferentemente do ZTECH, o Dominoops inclui elementos de programação orientada a objetos nas próprias regras do jogo, exigindo do jogador não apenas a compreensão desses elementos a nível conceitual, mas também sua aplicação.

Livovský e Porubän [12] desenvolveram um jogo 2D visto de cima no qual o jogador avança de fase ao encontrar a saída da fase atual. O jogador pode visualizar e interagir com o modelo de objetos do jogo — seja identificando a classe de algum elemento do mapa, alterando atributos ou criando instâncias —, e essa interação é necessária para concluir algumas das fases. Diferentemente desse jogo, Dominoops é multijogador, incentivando que estudantes compartilhem experiências e aprendam uns com os outros.

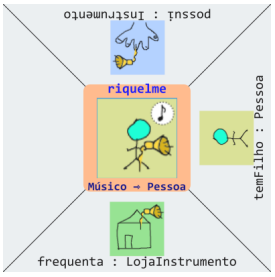
III. Dominoops

Dominoops é um jogo de cartas voltado para o ensino-aprendizagem de conceitos de programação orientada a objetos (POO). Seu nome é uma mistura de dominó (um jogo de mesa do qual Dominoops empresta algumas mecânicas) com OOP (sigla em inglês para programação orientada a objetos).

Para alcançar o objetivo educacional, o jogador deve conhecer três conceitos básicos da POO: classe, objeto e atributo. O jogo em si introduz, de maneira informal, os conceitos de herança e de compatibilidade de tipos. Por essa razão,



(a) Carta camelô, do tipo Loja.



(b) Carta riquelme, do tipo Músico.

Fig. 1. Duas cartas do deque original de Dominoops.

espera-se que o jogo possa ser jogado por estudantes de cursos de POO antes das aulas sobre herança, como uma forma de preparação para o aprendizado futuro [13].

O deque de cartas traz temas do mundo real com os quais a maior parte dos adultos tem familiaridade. Além disso, os conceitos de POO são apresentados no manual com outros nomes, mais comumente usados em situações cotidianas. Assim, espera-se que o jogo possa ser jogado também por pessoas sem conhecimento prévio em programação, ainda que nesse caso o jogo deixe de ter propósito educacional.

A. *Manual do jogo*

Em Dominoops, de 2 a 4 jogadores alternam-se em turnos, buscando conectar uma das cartas na mão com uma das cartas da mesa. O primeiro jogador a ficar sem cartas vence a partida. A seguir o jogo é explicado em termos de componentes, preparação, movimentação e condição de vitória.

Componentes. O jogo consiste de um deque com 30 cartas e um conjunto de 29 setas, usadas para conectar cartas. Cada carta representa uma entidade do mundo real, seja uma coisa, uma pessoa, um lugar, ou até elementos abstratos como números e cores. Cada carta possui um nome, um tipo, e de 0 e 4 atributos, apresentados nas laterais das cartas. Há 3 cartas de cada tipo, e cada tipo é representado por uma figura. Cada atributo possui um nome e um tipo, que determina com que cartas cada carta pode se conectar. Se duas cartas possuem o mesmo tipo, elas também possuirão os mesmos atributos. Por exemplo, a Fig. 1(a) representa a carta *camelô*, do tipo *Loja*. Essa carta possui dois atributos: *temDono*, do tipo *Pessoa*, e *vende*, do tipo *Objeto*.

No lugar do tipo de uma carta, podem vir dois tipos, ligados por uma seta, como é o caso da carta *riquelme* (Fig. 1(b)), onde aparece *Músico* → *Pessoa*. Isso significa que *riquelme* é do tipo *Músico*, e que o tipo *Músico* é um subtipo do tipo *Pessoa*. A influência dos subtipos nas regras é explicada na seção *Movimentação*.

Preparação. No início do jogo o deque é embaralhado e cada jogador recebe 4 cartas. O restante das cartas forma o monte, com a face voltada para baixo. A carta do topo do monte é então colocada, virada com a face para cima, no centro da mesa. As setas são colocadas sobre a mesa, ao alcance de todos os jogadores.

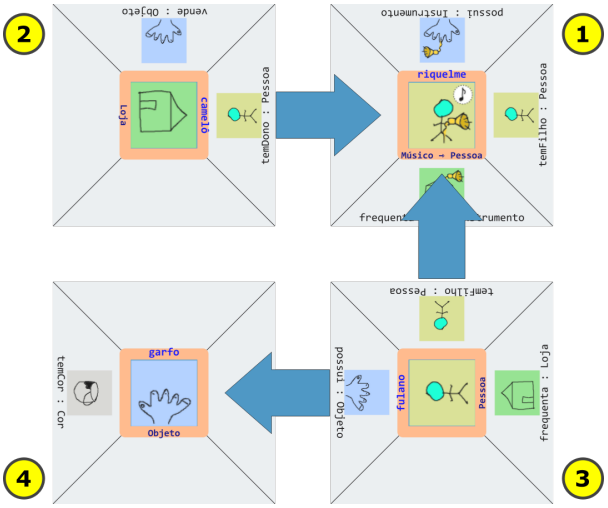


Fig. 2. Exemplo de partida em andamento.

Movimentação. O primeiro a jogar é definido por sorteio, e a partir daí os jogadores alternam-se em turnos, seguindo o sentido anti-horário. Em seu turno, o jogador deve retirar uma das cartas que tem na mão e colocá-la na mesa, vizinha a uma carta que ocupa uma das extremidades (isto é, carta que participa de no máximo uma conexão), e então conectar as duas cartas com uma seta. A seta representa a conexão entre um atributo de uma das cartas e uma carta adjacente a ela. Um atributo só pode se conectar a uma carta adjacente se essa carta for do mesmo tipo do atributo ou se for de algum subtipo dele (ou subtipo de um subtipo, e assim por diante). Note que a conexão pode ocorrer de um atributo da carta que estava na mesa para a carta jogada, ou de um atributo da carta jogada para a carta da mesa.

Por exemplo, considere uma partida, ilustrada na Fig. 2, em que a carta inicial é *riquelme* (1), do tipo *Músico*. O primeiro jogador tem a carta *camelô* em mãos. A carta *riquelme* não pode se conectar a *camelô* através do atributo *frequenta* (do tipo *LojaInstrumento*), pois *camelô* é do tipo *Loja*, e *Loja* não é um subtipo de *LojaInstrumento*, mas *camelô* pode se conectar a *riquelme* através do atributo *temDono* (do tipo *Pessoa*), pois *riquelme* é do tipo *Músico*, que é subtipo de *Pessoa*. Assim, deve-se rotacionar a carta *camelô* de forma que o atributo *temDono* seja adjacente à carta *riquelme*, e colocar uma seta partindo do atributo *temDono* até a carta *riquelme*. A interpretação fica: o *camelô* tem um dono, que é *riquelme*.

Seguindo a partida, o próximo jogador joga a carta *fulano* (3), que tem um filho chamado *riquelme*. Para isso ele conecta o atributo *temFilho* à carta *riquelme*. Neste momento, o último jogador deve jogar uma carta que se conecte a uma das extremidades, isto é, *camelô* (2) ou *fulano* (3) – a carta *riquelme* possui duas conexões, e por isso não é considerada extremidade. Ele opta por jogar a carta *garfo* (4), conectando o atributo *possui* de *fulano* a esta com uma seta. A mesa neste momento pode ser lida como "riquelme, o dono do camelô, é filho de fulano, aquele que possui um garfo".

Se o jogador, em seu turno, não possuir jogadas válidas, ele deve pegar a carta do topo do monte. Se a nova carta puder se conectar a uma das cartas da mesa, ele deve fazê-lo; caso contrário, ele deve passar a vez para o próximo jogador.

Condição de vitória. O jogador que ficar sem cartas vence a partida e ganha um ponto, além de ser o primeiro a jogar na próxima partida. Se o monte se esgotar e todos os jogadores passarem a vez, a partida se encerra e ninguém pontua.

Variações. Uma variação do jogo, voltada para jogadores mais experientes, inclui uma jogada chamada *upgrade*. Durante o seu turno, o jogador pode, a qualquer momento e quantas vezes quiser, sobrepor uma carta qualquer da mesa com uma carta em sua mão cujo tipo seja um subtipo do tipo da carta da mesa. Essa jogada só é permitida se as conexões continuarem sendo válidas após o *upgrade*. No momento em que o jogador fizer uma jogada de conexão, seu turno se encerra. Se o jogador não puder fazer uma jogada de conexão, ele deve pegar uma carta do topo do monte. Depois disso ele ainda pode realizar *upgrades* e, ao final, fazer uma conexão ou então passar a vez.

B. Analogia com a programação orientada a objetos

Dominoops apresenta conceitos de programação orientada a objetos, em alguns casos sem usar a terminologia técnica. Cartas representam objetos, e referências entre objetos são representadas como setas que conectam duas cartas. Os tipos do jogo representam classes (ou mesmo estruturas similares, como interfaces e enumerações), e não é permitida herança múltipla. As classes não são representadas explicitamente dentro do jogo, mas sua estrutura pode ser inferida a partir das cartas que representam os objetos.

O jogo não pretende, no entanto, representar todos os conceitos disponíveis em linguagens de programação orientadas a objetos, e nesse sentido possui limitações. Por exemplo, não há métodos, somente atributos. Além disso, não há o conceito de membros estáticos ou abstratos, nem restrições de visibilidade (privado, público etc.). Por fim, não há tipos primitivos (como ocorre na linguagem Java); em vez disso, tudo são objetos (como, por exemplo, na linguagem Ruby).

Os conceitos representados em Dominoops se alinham melhor a linguagens de programação com tipagem estática, como Java e C++, nas quais a compatibilidade de tipos depende dos tipos dos objetos. Em linguagens com tipagem dinâmica, como Python, a compatibilidade de tipos não depende do tipo declarado, e sim da presença de determinados atributos ou métodos.

IV. Avaliação

Para avaliar o jogo com relação à experiência de jogadores (com ou sem conhecimentos de programação), conduzimos sessões de *playtesting*. Este é o termo utilizado para o teste do jogo durante o processo de *game design* por potenciais jogadores. Em uma sessão de *playtesting*, usamos uma variedade de técnicas e ferramentas para analisar o comportamento dos jogadores, visando obter informações que ajudem a melhorar a experiência final [14].

A. Metodologia

Devido à pandemia do COVID-19, não foi possível realizar as sessões de *playtesting* presencialmente, com cartas impressas, e tampouco foi possível avaliar o jogo em sala de aula. Por essa razão, implementamos um protótipo do jogo no ambiente virtual Tabletop Simulator[1]. O Tabletop Simulator é um software pago, e cada jogador deve possuir uma cópia para jogar online com outros jogadores, o que limita o número de potenciais participantes para a avaliação. No entanto, ele foi escolhido pela sua popularidade e pela facilidade de se criar jogos de cartas personalizados.

Para avaliar Dominoops, conduzimos 3 sessões de *playtesting*, incluindo como participantes estudantes e profissionais da área de computação e de outras áreas. Antes do início de cada sessão, os participantes tiveram acesso ao manual do jogo e preencheram um termo de consentimento.

Durante as sessões, utilizamos o Discord[2] como plataforma de comunicação via voz. Cada sessão foi composta de 4 partidas, sendo as duas primeiras usando o conjunto de regras básico e as duas últimas usando a variação do jogo que inclui a jogada de *upgrade*. Todas as sessões contaram com 4 jogadores, incluindo ao menos um dos autores, que atuou como jogador e anfitrião.

Ao todo foram selecionados 8 participantes, sendo estes 5 pessoas que são da área de TI e 3 de outras áreas. Na primeira e na última sessão participaram pessoas que iniciaram ou concluíram uma disciplina de programação orientada a objetos na universidade. Na segunda sessão participaram leigos em programação, com experiência em game design e educação.

Ao final de cada sessão, cada participante manifestou suas impressões gerais sobre o jogo. Além disso o anfitrião, seguindo um roteiro semiestruturado, questionou os jogadores especificamente sobre a facilidade de entendimento das regras do jogo e do manual, a experiência de jogo, a sua aplicabilidade em sala de aula, o *design* das cartas, a regra de *upgrade* e sugestões de melhorias.

Com o consentimento dos participantes, cada sessão foi gravada em vídeo e áudio. Com isso, foi possível relembrar os comentários dos participantes, agrupando-os em conceitos, relembrar os erros comuns dos participantes durante o jogo, e medir a duração de cada partida.

B. Resultados

A Tabela I mostra o tempo decorrido durante as partidas, em minutos e segundos. O tempo total das partidas em cada sessão não excedeu uma hora, o que sugere que o uso do jogo em sala de aula possuiria um impacto pequeno sobre o planejamento de uma disciplina. A primeira e a terceira partidas se estenderam mais, pois os participantes ainda estavam se familiarizando com as regras do jogo e do movimento de *upgrade*.

Para a segunda sessão, foi preciso uma explicação e apoio maior durante as partidas devido ao perfil dos participantes, leigos em programação. A terceira sessão, apesar de ter

[1]Disponível em https://www.tabletopsimulator.com/
[2]Disponível em https://discord.com/

Duração das partidas					
	Partida 1	Partida 2	Partida 3	Partida 4	Total
Sessão 1	12m20s	07m30s	11m30s	05m40s	37m00s
Sessão 2	12m35s	06m40s	11m05s	10m00s	40m20s
Sessão 3	16m45s	07m20s	12m00s	07m45s	43m50s

Tabela I. Duração das partidas em minutos e segundos.

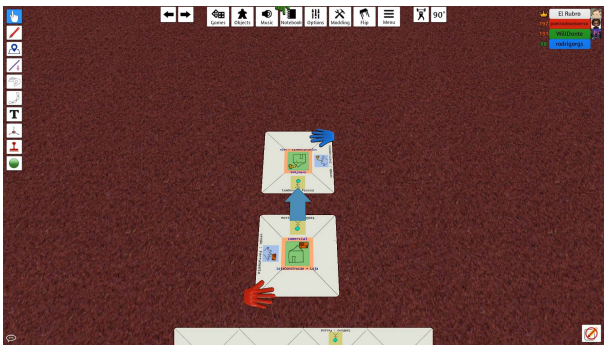


Fig. 3. Exemplo de erro ao conectar atributo com atributo, tirado de sessão de playtesting no Tabletop Simulator.

participantes da área, foi composta por pessoas que adquiriram recentemente o Tabletop Simulator, o que resultou em uma demora para explicação do ambiente.

A maioria concordou que o jogo é divertido ou ao menos interessante, e todos se mostraram favoráveis ao uso do Dominoops no contexto de ensino. Alguns participantes comentaram que a regra de *upgrade* é essencial, pois torna o jogo mais complexo e também mais dinâmico e interessante. O *design* das cartas foi considerado funcional e o fato de cada tipo (ex.: Pessoa, Loja) ser de uma cor diferente ajudou a visualizar as jogadas possíveis. Os participantes também identificaram estratégias de jogo, como rotacionar a carta jogada para bloquear um de seus atributos com uma carta adjacente.

Alguns participantes relataram dificuldade de entender as regras após ler o manual. Em particular, leigos em programação tiveram mais dificuldade com o linguajar usado no manual e indicaram que o conhecimento dos conceitos de programação orientada a objetos seria útil para entender as regras.

Os erros mais comuns praticados pelo jogadores envolviam posicionar as setas no sentido inverso (isto é, de uma carta para um atributo), conectar um atributo com outro atributo (e não com uma carta) e fazer o *upgrade* de uma carta por outra do mesmo tipo. A partida ilustrada na Fig. 3 mostra um exemplo de erro comum, onde o jogador da vez tentou conectar a carta do tipo *Loja de Construção* com a carta do tipo *Loja de Instrumento*, devido à semelhança do atributo *Pessoa* encontrado em ambas as cartas.

Foram feitas diversas sugestões durante as sessões, como aumentar o número de cartas na mão do jogador (com o intuito de eliminar o monte de compra) e haver uma variedade maior de tipos de cartas. Um participante chegou a sugerir que as cartas dos jogadores ficassem visíveis durante todo o jogo, tendo em mente a contagem de peças como opção de estratégia, assim como no Dominó. Foi sugerido também que um jogador possa fazer a jogada de *upgrade* mesmo fora de seu turno, tornando a partida mais dinâmica. Todas as sugestões são promissoras e revelam o interesse dos participantes na experiência, o que incentiva muitas melhorias a serem implementadas.

V. Considerações finais

Este artigo apresentou o Dominoops, um jogo de cartas que reforça conceitos de programação orientada a objetos. Através de sessões de *playtesting*, concluímos que o jogo tem potencial para ser usado em sala de aula e identificamos oportunidades de melhoria, sobretudo no manual do jogo.

Há diversas oportunidades para trabalhos futuros. Desejamos avaliar as variações propostas pelos participantes com novas sessões de *playtesting* e, ao mesmo tempo, avaliar outras plataformas para criação de protótipos de jogos de cartas, como o (roll20.net), que possui plano gratuito e exige menos recursos computacionais, viabilizando o uso do jogo em salas de aula virtuais. Por fim, planejamos conduzir simulações computacionais das partidas para entender o impacto de certas decisões de *design* – como número de cartas por jogador e quantidade de cartas por tipo – sobre variáveis como duração das partidas e número de vitórias por jogador.

Referências

[1] M. L. Scott, *Programming language pragmatics*. Morgan Kaufmann, 2000.

[2] A. Schmolitzky, "Teaching inheritance concepts with Java," in *Proceedings of the 4th international symposium on Principles and practice of programming in Java*, 2006, pp. 203–207.

[3] N. Liberman, C. Beeri, and Y. Ben-David Kolikant, "Difficulties in learning inheritance and polymorphism," *ACM Transactions on Computing Education (TOCE)*, vol. 11, no. 1, pp. 1–23, 2011.

[4] S. Xinogalos, M. Sartatzemi, and V. Dagdilelis, "Studying students' difficulties in an OOP course based on BlueJ," in *IASTED International Conference on Computers and Advanced technology in Education*, 2006.

[5] M. Prensky, "Digital game-based learning," *Computers in Entertainment (CIE)*, vol. 1, no. 1, pp. 21–21, 2003.

[6] M. Souza, L. Veado, R. Moreira, E. Figueiredo, and H. Costa, "A systematic mapping study on game-related methods for software engineering education," *Information and software technology*, vol. 95, pp. 201–218, 2018.

[7] M. Kölling, "The greenfoot programming environment," *ACM Transactions on Computing Education (TOCE)*, vol. 10, no. 4, pp. 1–21, 2010.

[8] S. Cooper, W. Dann, and R. Pausch, "Alice: a 3-d tool for introductory programming concepts," *Journal of computing sciences in colleges*, vol. 15, no. 5, pp. 107–116, 2000.

[9] A. M. Phelps, C. A. Egert, and K. J. Bierre, "Muppets: multi-user programming pedagogy for enhancing traditional study: an environment for both upper and lower division students," in *Proceedings Frontiers in Education 35th Annual Conference*. IEEE, 2005, pp. S2H–8.

[10] S. Xinogalos, M. Satratzemi, and C. Malliarakis, "Microworlds, games, animations, mobile apps, puzzle editors and more: What is important for an introductory programming environment?" *Education and Information Technologies*, vol. 22, no. 1, pp. 145–176, 2017.

[11] Y. S. Wong, M. Y. M. Hayati, and W. H. Tan, "A propriety game-based learning game as learning tool to learn object-oriented programming paradigm," in *Joint Int'l Conference on Serious Games*. Springer, 2016.

[12] J. Livovskỳ and J. Porubän, "Learning object-oriented paradigm by playing computer games: concepts first approach," *Open Computer Science*, vol. 4, no. 3, pp. 171–182, 2014.

[13] J. Hammer and J. Black, "Games and (preparation for future) learning," *Educational Technology*, pp. 29–34, 2009.

[14] P. Mirza-Babaei, N. Moosajee, and B. Drenikow, "Playtesting for indie studios," in *Proceedings of the 20th International Academic Mindtrek Conference*, ser. AcademicMindtrek '16. Association for Computing Machinery, 2016, p. 366–374.