Ballgorithm - Uma Ferramenta Introdutória Para Conceitos de Programação

Guilherme Afonso Melo Sousa Melo
Laboratório Telemídia - MA
Av. dos Portugueses, 1966
São Luís, MA 65080-805
guilhermemelo6@outlook.com

Carlos de Salles Soares Neto
Laboratório Telemídia - MA
Av. dos Portugueses, 1966
São Luís, MA 65080-805
csalles@deinf.ufma.br

ABSTRACT

The Computer Science Course presents one of the highest dropouts rates among higher education courses in Brazil. Besides that, it's known that the Algorithms discipline is the base of the knowledge learnt throughout the course and the first contact of students with computer programming. Having that said, the Ballgorithm was idealized to be a tool that includes a simple programming language and a ludical approach, making use of digital games visual elements. This paper aims to presente the development process of the Ballgorithm tool, the Ballcode language and the design decisions regarding to the system architecture.

KEYWORDS

Serious Games. Gamification. Teaching.

1 INTRODUÇÃO

Um problema recorrente entre os cursos de graduação em Ciência da Computação é a alta taxa de desistência. A taxa de estudantes que concluem o curso é de 14.3% [5]. Além de ser um número baixo por si só, torna-se ainda mais preocupante se comparado com a média geral de conclusões em universidades públicas, que é 45.9% [4]. Segundo Santos e Costa [1], o ensino de algoritmos acaba sendo a base do conhecimento durante o curso, e a dificuldade de aprendizado deste conteúdo acaba tornando-se um dos principais motivos para desistência do curso.

Dessa forma, é fundamental que o ensino de algoritmos seja tratado com relevante grau de importância, já que, além de ser a base dos conhecimentos que serão aprendidos ao longo da graduação, acaba sendo também o primeiro contato de muitos dos ingressantes com programação de computadores. Tendo isto em vista, é importante buscar métodos alternativos de ensino que aumentem a motivação do aluno, os quais sejam mais dinâmicos e lúdicos [3].

O objetivo deste trabalho é apresentar a ferramenta *Ballgorithm*, idealizada com o intuito de trazer uma abordagem mais lúdica ao aprendizado de algoritmos, apresentando atividades simples e dinâmicas, bem como a representação visual dos algoritmos como um jogo. Na Seção 2 apresentaremos a estrutura do *Ballgorithm*, falando de seus componentes e de como a ferramenta pode ser usada pelos usuários. Na Seção 3 entramos em detalhes sobre o processo de interpretação da linguagem e exibição dos objetos em tela. Por fim, na Seção 4 falamos brevemente sobre resultados preliminares obtidos através de testes com usuários.

In: XVII Workshop de Ferramentas e Aplicações (WFA 2018), Salvador, Brasil. Anais do XXIV Simpósio Brasileiro de Sistemas Multimídia e Web: Workshops e Pôsteres. Porto Alegre: Sociedade Brasileira de Computação, 2018.

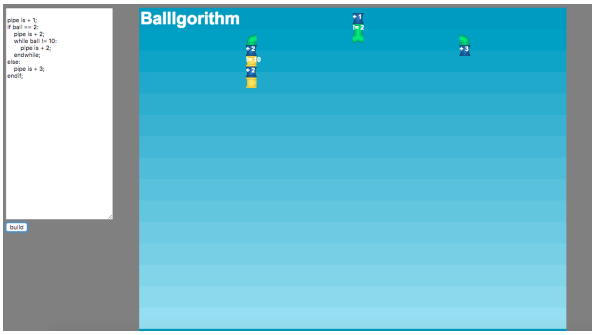
ISBN 978-85-7669-435-9.

2 BALLGORITHM

O *Ballgorithm* é uma ferramenta que tem como objetivo a prática e o aprendizado de conceitos básicos de linguagens de programação. Por conta disso, a ferramenta (que pode ser qualificada como um jogo sério) conta com sua própria linguagem, chamada *Ballcode*. O jogo funciona da seguinte forma: desafios envolvendo os elementos em tela (canos com operações matemáticas e bolas compostas por valores numéricos) devem ser resolvidos pelo jogador, criando um percurso com canos que irão alterar os valores numéricos presentes nas bolas. Ao final deste percurso, as bolas devem ter os seus valores equivalentes ao valor solicitado.

Na Figura 1, podemos ver a tela do jogo, que foi gerada após a submissão do código escrito pelo usuário.

Figure 1: Tela do Ballgorithm



O jogo consiste em se levar uma ou mais bolas do topo da tela a um ponto na base da tela, fazendo com que, ao final do percurso, os valores numéricos das bolas sejam iguais ao valor solicitado. Os valores especificados podem ser declarados tanto dentro dos níveis do jogo (pré-programados por outros usuários) quanto propostos por outra pessoa (professores, amigos, etc).

Para alterar estes valores, o jogador precisa utilizar os diferentes tipos de canos que se encontram disponíveis através da linguagem *Ballcode*, criada especificamente para o jogo. Cada cano tem uma função diferente, e parte do desafio presente no jogo é entender como alocá-los de forma eficiente, levando-se em consideração o número de canos utilizados. É importante ressaltar também o potencial de competitividade entre os usuários, tendo em vista que

a pontuação gerada ao término de um nível leva em consideração o número de canos utilizados na solução.

Assim sendo, pode-se afirmar que o *Ballgorithm* é, na verdade, uma ferramenta que simula uma máquina formada por canos, capaz de efetuar o processamento em sua entrada (bolas com valores numéricos quaisquer) e retornar de forma bem sucedida uma saída (bolas com os valores desejados).

Buscando uma forma mais simples de demonstrar na prática os conceitos básicos ensinados em programação, chegou-se no modelo atual do jogo. Note que os conceitos básicos de programação devem ser apresentados a usuários que tenham pouco (ou nenhum) conhecimento prévio. Dessa forma, os seguintes conceitos foram escolhidos:

- Entrada e Saída
- Atribuições
- Estruturas Condicionais
- Estruturas de Repetição

Estes conceitos podem ser percebidos nas abstrações presentes nos objetos que podem ser utilizados na ferramenta: a criação de novas bolas representa a entrada; os canos de atribuições; estruturas condicionais e de repetição fazem as vias de unidade de processamento; enquanto o valor final da bola nada mais é do que a saída da máquina criada pelo usuário utilizando tais elementos.

2.1 A Linguagem *Ballcode*

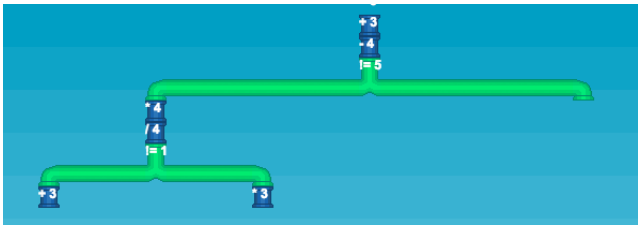
A linguagem *Ballcode* foi criada especificamente para o *Ballgorithm*, e o ponto mais importante da sua criação foi a sua simplicidade, já que é importante que toda (ou a maior parte) da dificuldade relacionada à resolução dos desafios fosse proveniente exclusivamente do raciocínio lógico envolvido nas questões. Dessa forma, optou-se por criar uma linguagem com sintaxe baseada em Python, por essa ser uma linguagem de fácil leitura e pelo foco que a mesma tem na indentação do código escrito.

2.1.1 *Comandos da Linguagem.*

(1) **Declarar Instância de uma nova bola:** O comando de atribuição atribui um valor a uma bola que será criada. Tal comando só atribui valores numéricos a bolas que estão sendo instanciadas. Ou seja, não é possível alterar o valor de bolas já existentes no jogo através do comando de atribuição.

(2) **If/ Estruturas Condicionais:** O comando de estruturas condicionais tem como objetivo definir uma bifurcação no percurso que será feito pelas bolas. As bolas entram no cano da estrutura condicional e, baseando-se no seu valor numérico atual, define-se qual caminho será percorrido. Diferentemente das estruturas semelhantes nas mais diversas linguagens de programação, que checam se os valores são iguais, diferentes, maior que e menor que, as estruturas de controle em *Ballcode* checam apenas se os valores comparados são iguais ou diferentes. Na Figura 2, vemos duas estruturas condicionais aninhadas, onde, no primeiro caso, se o valor da bola for diferente de 5 (cinco), esta segue pra direita, e caso contrário, ela segue para a esquerda. Dentro deste cano existem duas operações, onde o valor da bola é multiplicado por 4 (quatro) e depois dividido por 4 (quatro). Depois disso, há mais uma estrutura condicional, onde, caso o cano seja diferente de 1 (um), ele seguirá para a direita, e caso contrário, para a esquerda.

Figure 2: Bloco de canos gerados pelo comando das Estruturas Condicionais



(3) **Atribuições de Valores/Operações Básicas:** As atribuições de valores permitem que o usuário altere o valor das bolas que foram declaradas anteriormente, através de operações matemáticas básicas, como adição, subtração, multiplicação e divisão. A atribuição de valores é a principal ferramenta para a resolução dos problemas propostos. Na figura 3 podemos ver um exemplo de cano de atribuição.

Figure 3: Cano gerado pelo comando de atribuição

(4) **While/Estrutura de repetição:** As estruturas de repetição permitem ao usuário utilizar um comando (ou blocos de comandos) repetidas vezes, de forma a evitar redundâncias e reduzir os caminhos de canos que serão montados. A estrutura de repetição no projeto conta com dois canos. Um cano determina a posição inicial do que seria o percurso a ser repetido e outro determina a posição final. Caso a condição especificada para a parada da repetição não seja cumprida, a bola, ao chegar na posição final do percurso, será mandada de volta para a posição inicial, até que a condição de parada seja atendida e, assim, ela possa continuar o trajeto.

Na Figura 4 podemos ver a estrutura que será gerada com o uso de estruturas de repetição, contendo uma atribuição dentro do bloco de código.

2.1.2 *Especificação da Sintaxe Ballcode.*

Os comandos textuais utilizados no *Ballgorithm* são avaliados através de uma comparação com comandos pré-determinados existentes na linguagem. Estes comandos foram criados de forma a simular as operações mais básicas presentes na criação de algoritmos.

Abaixo, a Tabela 1 demonstra os comandos de *Ballcode*, bem como a sintaxe correta para utilizá-los:

Figure 4: Bloco de canos gerados pelo comando das Estruturas de repetição

Table 1: Comandos em *Ballcode*

Declaração de um novo objeto	**new ball = [valor inteiro de um dígito];**
Atribuição de um valor	**pipe is [operação básica (+, -, * ou /)];**
Estrutura condicional	**if ball ['==' ou '!='] [valor inteiro de até 2 dígitos]:** //bloco de código **else:** //bloco de código **endif;**
Estrutura de Repetição	**while ball ['==' ou '!='] [valor inteiro de até 2 dígitos]:** //bloco de código **endwhile;**

2.2 O Usuário enquanto agente ativo

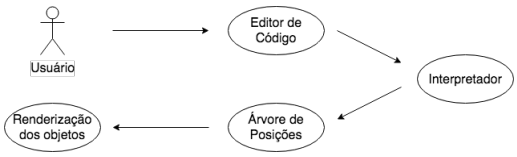
Embora o *Ballgorithm* tenha sido idealizado inicialmente como uma ferramenta onde o usuário final não exerce nenhuma atividade além de resolver as questões solicitadas, viu-se que existe a possibilidade de o usuário submeter seus próprios desafios, tendo em vista que as máquinas podem ser modificadas de forma a manter determinado comando "trancado", de forma a fazer com que os usuários encontrem uma resposta para os enigmas sendo obrigados a jogar em condições mais limitadas, em vez de fazer com que a máquina seja gerada apenas com os comandos de sua escolha.

Como já foi descrito anteriormente, o *Ballgorithm* possui dois grandes pilares: a linguagem *Ballcode*, responsável por adicionar na tela os elementos utilizados no jogo, e o jogo propriamente dito, que exibe os objetos criados através da linguagem, além de contar com um sistema de física e movimentação para os objetos. Pode-se dizer que o funcionamento do *Ballgorithm* baseia-se em dois grandes componentes: o interpretador de *Ballcode* e o gerenciamento dos objetos presentes na tela do jogo. Além desses dois grandes componentes, também existem funções e objetos que têm como objetivo fazer a interação entre eles.

No momento em que o usuário decide enfim gerar o percurso descrito em seu código, o interpretador do *Ballgorithm* passa pelas seguintes etapas: leitura das linhas, normatização e classificação, criação de objetos do tipo linha, definição das posições dos objetos na tela do jogo, criação de objetos presentes no jogo (*sprites*), instanciação dos *sprites* e, enfim, exibição do que foi gerado.

A Figura 5 ilustra, através de um diagrama de caso de uso, os processos que ocorrem entre a submissão do código do usuário e a renderização dos componentes do jogo em tela.

Figure 5: Diagrama de Caso de Uso



3 O *BALLGORITHM* E O COMPILADOR

Embora a linguagem *Ballcode* seja de fundamental importância para a experiência do usuário, é importante ressaltar que, de nenhuma forma, este projeto tem como objetivo funcionar como um compilador, embora muitas das funções existentes nele sejam análogas a certas funções presentes em um compilador. Considerando o modelo conceitual de um compilador, os passos por ele seguidos são análise do código, representação semântica e síntese, para então assim gerar o código executável [2].

De forma análoga ao compilador, o código do usuário, ao ser submetido, é analisado, gera uma lista com os objetos e sintetiza estes objetos naqueles que serão renderizados na tela do jogo, gerando assim o percurso descrito pelo usuário, o que seria, em uma comparação grosseira, o executável gerado pelo compilador.

3.1 O interpretador da linguagem *Ballcode*

O interpretador do *Ballgorithm* tem como objetivo extrair o conteúdo das linhas do código do usuário e, a partir destas, criar os objetos que podem ser organizados na árvore de posições. O interpretador de *Ballcode* foi todo escrito em Javascript, contando com cerca de 600 linhas de código.

3.1.1 Leitura das linhas. Quando o usuário termina de escrever seu código em *Ballcode*, o primeiro passo do interpretador é a leitura do conteúdo bruto, que será dividido em linhas, que posteriormente serão alocadas em um vetor. Optou-se por utilizar o símbolo de ponto e vírgula (';') para delimitar o fim de uma linha. No caso das linhas que referenciam o início de blocos de código (linhas de *if* e *while*), o símbolo que delimita o fim da linha é o símbolo de dois pontos (':'). Embora esses símbolos não sejam de fundamental importância para o interpretador, optou-se por deixá-los obrigatórios por fins pedagógicos. Os blocos de códigos são delimitados de forma semelhante aos blocos da linguagem Python: não são utilizadas chaves, e sim a indentação do código.

Durante a leitura do conteúdo bruto, cada símbolo de quebra de linha ('\n') adiciona o conteúdo lido até o momento no vetor, salvando assim cada linha do conteúdo bruto como uma *string* individual. Depois disso, tendo o conteúdo bruto dividido em linhas, é preciso normatizar este conteúdo.

3.2 Normatização das linhas

Tendo cada linha devidamente armazenada como *string*, é preciso classificá-las. Porém, antes desse processo, três coisas devem ser feitas: remover eventuais linhas em branco, descobrir se as linhas se encontram dentro de um bloco de código (pois isto será de fundamental importância quando gerarmos os elementos na tela) e remover eventuais redundâncias de espaçamento, tendo em vista que isso pode dificultar o processo de classificação.

Caso o conteúdo da *string* de linha seja vazio, ela simplesmente é excluída do processo, e nem mesmo é adicionada ao vetor de linhas em formato bruto. Também é importante salientar que os comandos, em hipótese alguma, podem ser quebrados em mais de uma linha.

Verificar se a linha está dentro de algum bloco de código é relativamente simples: como o conteúdo de cada linha foi obtido em sua totalidade, só precisamos verificar quantos símbolos de espaço ('\u0020') foram utilizados no começo da linha. Por motivos pedagógicos, a definição dos blocos é bastante restrita, e quaisquer erros de indentação podem (e muito provavelmente) gerarão erros de interpretação do código. Determinar se uma linha se encontra dentro de um bloco de código é um processo fundamental para o agrupamento dos objetos, tendo em vista que isso determinará a posição do objeto em relação aos objetos do contexto, já que eventuais erros de indentação fariam com que um objeto saísse da sequência desejada, ou mesmo passasse a compor outra estrutura (um comando que deveria ser do cano à direita do if, se indentado erroneamente, acabaria por não fazer parte do cano).

Após verificarmos se a linha pertence ou não a um bloco de código, removemos as redundâncias de espaçamento, substituindo qualquer quantidade de espaço por um espaço simples, de forma a facilitar o processo de classificação das linhas.

3.3 Classificação das linhas

Após a normatização das linhas, elas são classificadas. A classificação ocorre da seguinte maneira: cada linha é comparada com um conjunto de expressões regulares, sendo que cada expressão regular é referente a um tipo de linha diferente. Caso a comparação entre a linha e uma das expressões regulares retorne um valor verdadeiro, sabemos qual o tipo da linha, e esse dado será adicionado em um objeto do tipo linha. Caso a comparação com as expressões regulares não retorne verdadeiro em nenhum caso, podemos afirmar que a linha não constitui um comando válido, e isso retornará ao usuário um erro. Na Tabela 2, é possível ver as expressões regulares com as quais as linhas são comparadas.

3.4 Criação dos objetos de tipo Linha

Cada linha, depois de classificada, é enviada a um vetor de linhas, contendo os seguintes dados: tipo de linha, bloco e conteúdo. O valor do conteúdo possui os elementos de texto presentes na *string* bruta, tendo em vista que estes elementos serão utilizados para definir as características dos objetos, como o valor numérico das bolas, a operação dos canos, valores das estruturas de controle, etc.

3.5 Definição dos objetos em tela

Como o principal objetivo do jogo é demonstrar de forma visual o caminho pelo qual as variáveis do código passam, é de fundamental importância que o jogo seja assertivo neste ponto, definindo com exatidão as posições dos objetos que estarão presentes em tela. Sendo assim, a ideia para definir as posições dos objetos foi o uso de uma árvore binária, já que a abstração usada para uma raiz e suas folhas se assemelha muito à abstração usada nos canos de estruturas de controle presentes no *Ballgorithm*. Esta foi a estrutura de dados utilizada para a organização dos elementos. Porém, com o passar do tempo, a estrutura mostrou-se limitada para lidar com os diferentes casos de canos. O fato das folhas das sub-árvores não guardarem quem são os seus nós-pais também se mostrou um problema, já que, para determinar as posições de certos elementos, é preciso visitar constantemente seus nós pais, e até mesmo avós. Desta forma, a árvore binária acabou sendo adaptada.A abstração que mais casava com a alocação dos canos era uma árvore ternária. Com a árvore ternária, é possível definir a posição dos objetos em tela apenas se baseando pela relação do objeto com o seu nó pai. Os canos sempre se localizarão abaixo de seu pai, e caso estes seja um filho à esquerda ou à direita, essa posição terá uma variação no eixo X.

Além disso, o último nó acessado na estrutura é guardado, evitando assim percorrer a árvore múltiplas vezes em busca de seu nó pai. Após a inserção dos objetos na estrutura, ela é percorrida, e durante esse processo os objetos têm suas posições na tela definidas, levando em consideração a sua relação com o nó raiz, que possui a posição inicial mapeada. Com isso, os processos que envolvem a interpretação das linhas terminam, e os objetos do jogo começam a ser criados.

Table 2: Comandos em *Ballcode* e as Expressões Regulares referentes

Comando	Expressão Regular
Nova bola	/^\s*new ball\s*=\s*\d{1,2}\s*;\s*$/
Atribuição	/^\s*pipe is [+-/*]\s*\d{1,2}\s*;\s*$/
Estrutura Condicional	/^\s*if ball\s* (== | !=) \s*\d{1,2}\s*:\s*$/ /^\s*else\s*:\s*$/ /\s*endif\s*;\s*/
Laço de Repetição	/^\s*while ball\s*(==|!=)\s*\d{1,2}\s*:\s*$/ /^\s*endwhile\s*;\s*$/

3.6 Gerenciando os objetos em tela

Após a criação dos objetos na árvore, os elementos são alocados em vetores, agrupando os objetos de acordo com seus tipos. Tendo os grupos definidos, o interpretador manda um sinal para a *engine* gráfica, que começa a instanciar grupos de objetos, baseados nos vetores em que os objetos anteriormente foram alocados. Assim, cada objeto é instanciado com suas posições na tela e seus dados relevantes, como o símbolo da operação ou o tipo de comparação.

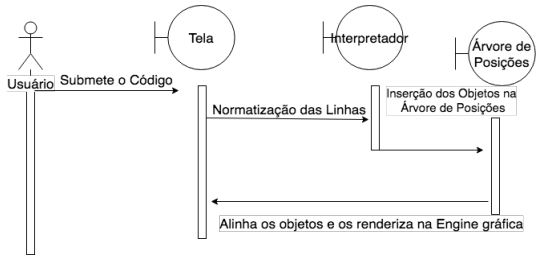
3.7 Instanciando os objetos

Os objetos de cada vetor são lidos, e deles são extraídos dados como suas respectivas posições em tela, informações sobre a operação referente ao objeto e valores. Os objetos recebem suas posições tendo a tela como base. A posição inicial no eixo X é a média entre 0 (zero) e o valor máximo, e cada elemento à esquerda deste objeto tem sua posição tirada da média entre o objeto inicial e 0 (zero), enquanto os elementos à direita tem suas posições definidas a partir da média entre o valor inicial e o valor máximo da tela, e assim os outros objetos tem suas posições determinadas recursivamente.

3.8 Definindo as características físicas dos objetos

Depois de alocados os objetos em tela, é preciso determinar o que ocorre durante as colisões, já que cada elemento, quando em contato com as bolas, altera seus percursos ou valores. Assim, através da *engine* gráfica, podemos atribuir aos objetos regras de colisão e gravidade (no caso da bola). Por uma limitação da *engine* utilizada, não podemos aplicar colisões apenas nas paredes dos canos, o que torna necessário usar o seguinte truque: ao colidir com determinado cano, a posição da bola em (x, y) é mudada para a posição do cano posterior do percurso.

Figure 6: Diagrama de sequência do *Ballgorithm*



Após todos esses processos, os objetos são enfim renderizados em tela, e as bolas seguem o percurso automaticamente, fazendo os desvios definidos pelo usuário em seu código. Na Figura 6, é mostrada a ordem de execução de cada um dos passos citados anteriormente.

4 RESULTADOS

Testes preliminares da ferramenta foram feitos com usuários, e suas impressões foram captadas através de um Questionário de Satisfação do Usuário, abordando aspectos como facilidade de uso, didática da apresentação dos contextos, simplicidade do tutorial, além de outros aspectos funcionais, como velocidade de resposta e interface da ferramenta. Os resultados foram em grande maioria positivos no tocante à facilidade do uso da ferramenta e às abstrações utilizadas para os conceitos de programação.

5 CONCLUSÃO

Com este trabalho, pode-se concluir que o *Ballgorithm* é uma ferramenta multimídia bastante promissora, que pode ser utilizada em sala de aula e até mesmo por usuários finais com interesse em aprender a programar. Embora falhas de interface tenham sido apontadas nos questionários, estas falhas não foram determinantes para prejudicar de forma decisiva o uso da ferramenta. É importante também salientar que a ferramenta ainda pode oferecer tradução da linguagem para diversos idiomas, desafios entre os usuários e outros modos de jogo, que podem ser adequados de acordo com o tópico a ser ensinado.

REFERÊNCIAS

[1] Rodrigo Pereira dos Santos and Heitor Augustus Xavier Costa. 2006. Análise de Metodologias e Ambientes de Ensino para Algoritmos, Estruturas de Dados e Programação aos iniciantes em Computação e Informática. *INFOCOMP* 5, 1 (2006), 41–50.

[2] Dick Grune, Kees Reeuwijk, Henri Bal, Ceriel Jacobs, and Koen Langendoen. 2012. *Modern Compiler Design.* Springer.

[3] W. Huitt. 2011. Motivation to Learn: An Overview. (2011). http://www.edpsycinteractive.org/topics/motivation/motivate.html

[4] MEC/Inep. 2017. Taxa de conclusão média dos cursos de graduação presenciais. (2017). http://www.observatoriodopne.org.br/metas-pne/12-ensino-superior/estrategias/12-3-fluxo/indicadores

[5] Luísa Behrens Palmeira and Matheus Parreiras Santos. 2015. Evasão no Bacharelado em Ciência da Computação da Universidade de Brasília: análise e mineração de dados. (2015).