Universidade Federal do Maranhão
Centro de Ciências Exatas e Tecnologias
Departamento de Informática
Curso de Ciência da Computação

Guilherme Afonso Melo Sousa Melo

Ballgorithm - Uma Ferramenta Introdutória Para Conceitos de Programação

São Luís
2018

GUILHERME AFONSO MELO SOUSA MELO

Ballgorithm - Uma Ferramenta Introdutória Para Conceitos de Programação

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Carlos de Salles Soares Neto

São Luís

2018

Ficha gerada por meio do SIGAA/Biblioteca com dados fornecidos pelo(a) autor(a).
Núcleo Integrado de Bibliotecas/UFMA

Melo Sousa Melo, Guilherme Afonso.
Ballgorithm : Uma Ferramenta Introdutória Para Conceitos de Programação / Guilherme Afonso Melo Sousa Melo. – 2018.
37 f.

Orientador(a): Carlos Salles Soares Neto.
Monografia (Graduação) – Curso de Ciência da Computação, Universidade Federal do Maranhão, São Luís, 2018.

1. Algoritmos. 2. Ensino. 3. Gamificação. 4. Jogos. I. Salles Soares Neto, Carlos. II. Título.

Guilherme Afonso Melo Sousa Melo

Ballgorithm - Uma Ferramenta Introdutória Para Conceitos de Programação

Trabalho de Conclusão apresentado como requisito parcial para obtenção de grau de Bacharel em Ciência da Computação

Trabalho aprovado. São Luís, 23 de janeiro de 2018:

Carlos de Salles Soares Neto
Orientador

Tiago Bonini Borchartt

Ana Carolina Melo de Oliveira Farias

São Luís
2018

Agradecimentos

Agradeço aos meus pais, Francinê e Amélia, por sempre me apoiarem de forma irrestrita e por acreditar no meu potencial, por todos os exemplos, cuidado, paciência, presteza e carinho. Não há dúvidas de que não seria um milésimo do que sou sem eles.

Agradeço aos membros d'A Sociedade, por serem os melhores amigos que alguém como eu poderia ter. Aos grandes amigos que tive o prazer de conhecer ao longo da minha jornada na graduação. Agradeço a minha namorada, Amanda, por ser sempre presente e companheira.

Agradeço também aos ótimos professores que foram fundamentais durante a minha formação acadêmica e profissional, em especial a professora Greiciane, de Cálculo, e o meu orientador, sempre disposto a me ajudar durante essa jornada.

"Do a barrel roll.
(Peppy Hare)

Resumo

O curso de Ciência da Computação possui uma das maiores taxas de evasão dentre os cursos de ensino superior do país. Além disso, é sabido que a disciplina de Algoritmos, além de ser a base do curso, também é o primeiro contato com programação de muitos dos ingressantes. Tendo tudo isso em mente, o *Ballgorithm* foi idealizado, sendo uma ferramenta que conta com uma linguagem simples e com uma abordagem lúdica, fazendo a utilização de elementos visuais de jogos digitais. Este trabalho tem como objetivo apresentar o desenvolvimento do *Ballgorithm*, bem como a linguagem criada para a utilização do mesmo, a *Ballcode*, e as decisões de *design* referentes à arquitetura do sistema. Primeiramente a estrutura da linguagem *Ballcode* é apresentada, mostrando seus comandos, limitações e abstrações utilizadas. Após isso a arquitetura do sistema como um todo é analisada, mostrando os procedimentos de interpretação da linguagem e renderização dos objetos do jogo com mais detalhes e em mais baixo nível. Por fim, analisamos casos de uso do sistema, explicitando cada passo dos processos que ocorrem da submissão do código em *Ballcode* até a exibição dos elementos criados pelo usuário na tela.

Palavras-chave: Jogo sério. Gamificação. Ensino.

Abstract

The Computer Science Course presents one of the highest dropouts rates among higher education coureses 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. Initially, the Ballcode language basic structure is presented, showing it's basic commands, limitations and the abstractions used to idealize it. Then, the system architecture is analyzed, showing the procedures of the language interpretation and the rendering of the objects on screen at a lower level. Finally, use cases of the system are analyzed, showing every every step of the process between the submission of the Ballcode written by the user and the rendering of the objects in the screen.

Keywords: Serious Games. Gamification. Teaching.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

Um problema que assola 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% (PALMEIRA; SANTOS, 2015). 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% (MEC/INEP, 2017). Segundo Santos e Costa, 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.

Sendo assim, é fundamental que o ensino de algoritmos seja tratado com o maior grau de importância possível, 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, que sejam mais dinâmicos e lúdicos (HUITT, 2011).

O*Ballgorithm* é um jogo sério idealizado para trazer uma abordagem mais lúdica ao aprendizado de algoritmos, apresentando atividades simples e dinâmicas, bem como a representação visual dos algoritmos como um jogo.

1.1 Objetivos

1.1.1 Objetivo Geral

O principal objetivo deste trabalho é o desenvolvimento do *Ballgorithm*, um jogo sério que auxilie seus usuários no aprendizado de algoritmos e linguagens de programação, bem como a criação de uma linguagem simples que deve ser utilizada dentro do jogo.

1.1.2 Objetivos Específicos

a) Criação de um jogo que utilize as abstrações mais recorrentes durante o estágio inicial do aprendizado de algoritmos;

b) Criação de uma linguagem simples que possa ser utilizada dentro do jogo, de forma que os usuários apliquem as abstrações utilizadas de forma intuitiva e simples;

c) Interpretar esta linguagem com o objetivo de renderizar em tela objetos do jogo de acordo com os códigos escritos pelos usuários.

1.2 Estrutura do Trabalho

Este trabalho está organizado de forma a exibir gradativamente o processo de desenvolvimento da ferramenta *Ballgorithm*, bem como todos os passos que ocorrem durante a sua execução. No Capítulo 2 podemos analisar trabalhos com objetivos semelhantes aos deste, os pontos positivos e abordagens para o ensino presentes em cada um. Em seguida, no Capítulo 3, é mostrado o funcionamento do jogo *Ballgorithm*, e suas regras são explicadas de forma intuitiva.

Já no Capítulo 4, o funcionamento do sistema é mostrado com mais detalhes e em mais baixo nível, detalhando os processos de interpretação da linguagem e a renderização da tela do jogo. No Capítulo 5 é feita uma análise de problemas propostos e suas respectivas soluções utilizando o *Ballgorithm*. Por fim, os resultados obtidos são analisados, bem como os trabalhos futuros e perspectivas relacionadas a esta ferramenta.

2 Trabalhos Relacionados

Este capítulo apresenta uma relação de trabalhos relacionados ao *Ballgorithm*. Sendo assim, realiza-se um levantamento de ferramentas que possuem semelhanças com o trabalho proposto. Por ser ao mesmo tempo um jogo e um ambiente que permite que o usuário escreva código, é importante notar que os trabalhos aqui presentes podem ser tanto jogos como ambientes de desenvolvimento.

2.1 Chien 2D

A Chien 2D é uma biblioteca voltada para o desenvolvimento de jogos baseada na linguagem C (RADTKE; BINDER, 2010). Esta biblioteca foi criada visando motivar os alunos do curso de Ciência da Computação através do desenvolvimento de jogos. Tendo em vista o conhecimento limitado dos alunos em estágios iniciais do curso, esta biblioteca busca simplificar ao máximo os comandos que nela são utilizados, inclusive usando palavras reservadas em português.

A Chien 2D conta com vários métodos referentes ao desenvolvimento de jogos, que abrangem todos os passos referentes à implementação de jogos, desde física, *sprites* e telas até a implementação de sons.

O projeto foi implementado pela primeira vez em 2006 na PUCPR, obtendo resultados satisfatórios dentro de sua proposta, obtendo um nível de satisfação de usuários bem elevado e aumentando significativamente o engajamento dos mesmos em suas respectivas comunidades de desenvolvedores de jogos (RADTKE; BINDER, 2010).

A Figura 1 mostra um jogo criado com o uso da biblioteca.

Figura 1 – Jogo implementado com a Chien 2D

Fonte: (RADTKE; BINDER, 2010)

2.2 Wanda

Com o objetivo auxiliar alunos de Algoritmos e reduzir alarmantes níveis de evasão dentro do curso de Ciência da Computação, o Wanda utiliza mecânicas de jogos de cartas para motivar os alunos (DRUMOND; SALLES, 2014).

O Wanda é um *framework* desenvolvido em Love2D, uma plataforma de desenvolvimento de jogos, que visa dar ao usuário ferramentas para criação de jogos de cartas. Fornecendo um conjunto de regras iniciais, o Wanda permite que seus usuários criem jogos de cartas dentro dessas regras de execução, e inclusive permite que usuários diferentes disputem entre si, de forma a comparar seus algoritmos dentro de jogos gerados através do *framework*.

Através do Wanda, diferentes jogos de cartas podem ser gerados e, dessa forma, diferentes habilidades e algoritmos podem ser testados, fazendo assim com que o Wanda seja extremamente versátil durante o ensino de algoritmos.

Na Figura 2, podemos ver o jogo Jo-Ken-Po sendo utilizado por alunos. Neste jogo, cada jogador deve criar uma lógica para que seu personagem jogue suas cartas de pedra papel ou tesoura (DRUMOND; SALLES, 2014).

Figura 2 – Jo-Ken-Po, um jogo criado através do Wanda sendo usado por alunos

Fonte: (DRUMOND; SALLES, 2014)

2.3 SORTIA

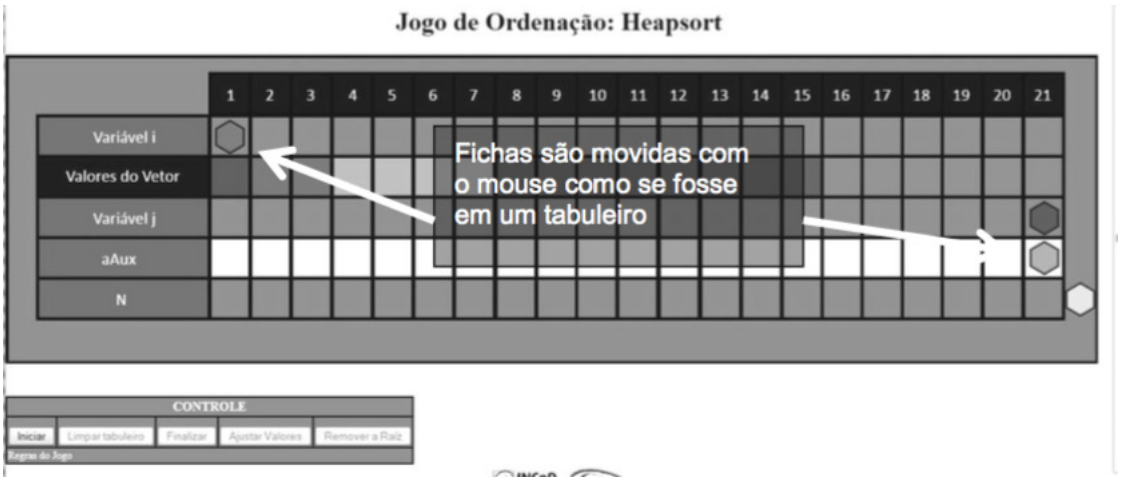
O SORTIA é um jogo digital que tem como objetivo ordenar valores numéricos seguindo os passos dos algoritmos de ordenação aprendidos na disciplina de Estrutura de Dados(BATTISTELLA; WANGENHEIM; VON, 2012).

Tendo em vista a dificuldade dos alunos ao absorver conteúdos abstratos que eram passados apenas com aulas expositivas (BATTISTELLA; WANGENHEIM; VON, 2012),

a criação de um jogo que aplica os conteúdos mostrados foi uma maneira de mostrar o conteúdo aos alunos de forma menos abstrata.

Na Figura 3, podemos observar a tela principal do jogo, que consiste um tabuleiro com campos para as diferentes variáveis utilizadas nos algoritmos de ordenação e em valores numéricos, onde o usuário deve, a cada passo, movimentar as variáveis de acordo com o que seria movimentado durante a execução dos algoritmos de ordenação.

Figura 3 – Tela do jogo SORTIA



Fonte: (BATTISTELLA; WANGENHEIM; VON, 2012)

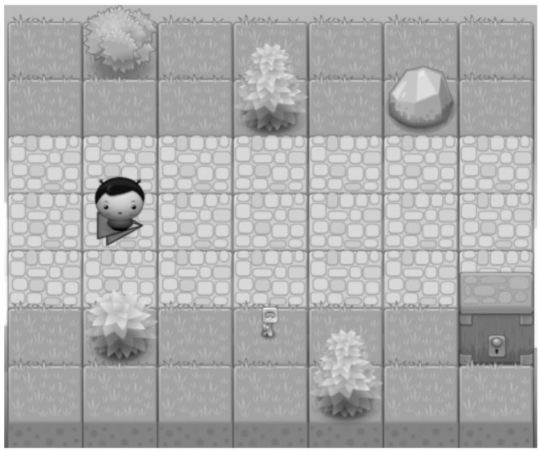
2.4 Takkou

O Takkou é uma ferramenta lúdica voltada ao ensino e prática de algoritmos e programação (BARBOSA; FERNANDES; CAMPOS, 2011). Nesta ferramenta, é preciso criar um bloco de instruções de forma que, quando executado, realize as tarefas requeridas com o mínimo de tempo possível. Na Figura 4 a tela principal do jogo, onde podemos ver um cenário dividido em *tiles*, com elementos passíveis de interação, como a porta ou a chave.

O Takkou pode ser definido como um jogo da categoria *puzzle*, por contar com desafios de lógica e solução de enigmas (ADAMS, 2014). Os desafios de lógica do Takkou consistem em mover o personagem pela tela e fazê-lo cumprir determinados objetivos, como pegar uma chave e abrir uma porta.

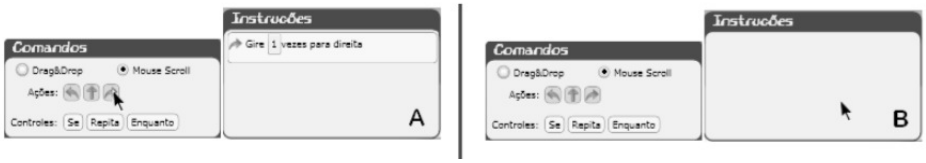
Os comandos do Takkou são selecionados a partir de uma interface simples e intuitiva, como podemos ver na Figura 5, tendo como principal objetivo fazer com que o usuário se preocupe apenas com a lógica da resolução dos problemas propostos (BARBOSA; FERNANDES; CAMPOS, 2011).

Figura 4 – Tela principal do Takkou



Fonte: (BARBOSA; FERNANDES; CAMPOS, 2011)

Figura 5 – Interface do Takkou

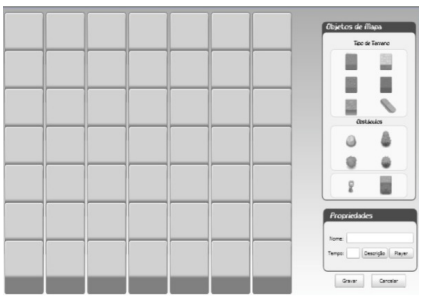


Fonte: (BARBOSA; FERNANDES; CAMPOS, 2011)

Além disso, o jogo possibilita a criação de novos níveis, fornecendo uma tela de criação de mapas, que podem ser modificados livremente, de forma que atendam diferentes necessidades pedagógicas dos usuários(BARBOSA; FERNANDES; CAMPOS, 2011).

A Figura 6 mostra a tela de criação de níveis, contando com vários elementos que podem ser utilizados na customização das fases do jogo.

Figura 6 – Tela de Construção de mapas do Takkou



Fonte: (BARBOSA; FERNANDES; CAMPOS, 2011)

2.5 Codecademy

O Codecademy é uma plataforma voltada para o ensino de diferentes linguagens de programação(CODECADEMY, 2018). O sistema consiste em inúmeras atividades de grau de dificuldade crescente, buscando abordar os conceitos básicos de cada linguagem. Os desafios contém sempre uma introdução e dicas que podem ajudar o usuário caso este sinta dificuldades. O sistema utiliza fortemente conceitos de gamificação, como *badges* e *achievements*.

2.6 Comparando trabalhos correlatos e o *Ballgorithm*

Todos os trabalhos citados anteriormente são propostas para solucionar diferentes problemas encontrados durante o aprendizado de algoritmos. O *Ballgorithm* busca solucionar a mesma problemática, buscando uma forma de mesclar diferentes soluções vistas em cada um destes trabalhos citados anteriormente. Como ilustra a Tabela **??**, o *Ballgorithm* busca trazer no projeto os pontos mais importantes encontrados nos trabalhos correlatos aqui apresentados.

Tabela 1 – Tabela comparativa entre os trabalhos correlatos e o *Balgorithm*

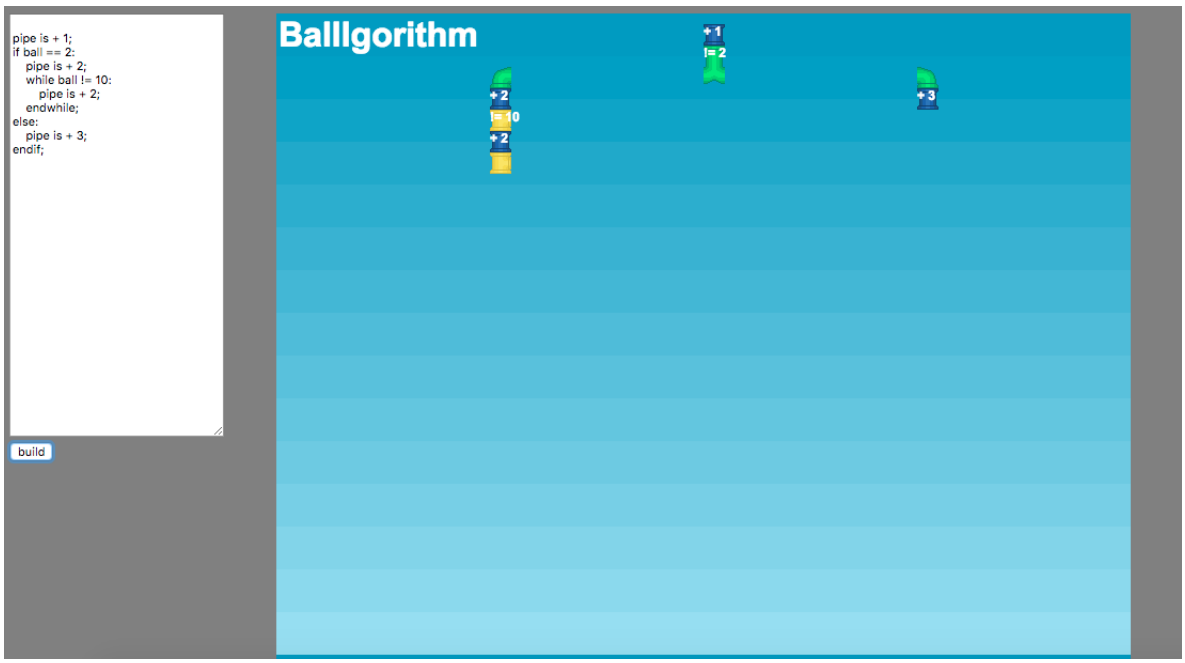
	Ballgorithm	Chien2d	Wanda	SORTIA	Takkou	Codecademy
Não necessita conhecimento prévio de programação	■			■	■	■
Possibilita criação de Código	■	■	■			■
Geração do código em interface gráfica	■	■	■			

3 *Ballgorithm*

O *Ballgorithm* é uma ferramenta que tem como objetivo a prática e o aprendizado de conceitos básicos de linguagens de programação. Sendo assim, a ferramenta (que pode ser qualificada como um jogo) conta com sua própria linguagem, chamada *Ballcode*. O jogo funciona da seguinte forma: desafios envolvendo os elementos do jogo (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, devem ter os seus valores equivalentes ao valor solicitado.

Na Figura 7, podemos ver a tela do jogo, que foi gerada após a submissão do código escrito pelo usuário.

Figura 7 – Tela do Ballgorithm



Fonte: Acervo do autor

3.1 Modelo Conceitual 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.

Sendo assim, 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).

3.1.1 Conceitos Utilizados

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 no jogo: 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.

3.2 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.

3.2.1 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 8, 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.

Figura 8 – Bloco de canos gerados pelo comando das Estruturas Condicionais

Fonte: Acervo do autor

3. **Atribuições de Valores/Operações Básicas:** As atribuições de valores permitem que o usuário altere o valor das bolas que foram declarados 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. sintaxe da atribuição de valores é a seguinte: "pipe is + 1;". Com este código, o usuário especifica que o cano é um cano que adiciona o número 1 ao valor atual da bola, como podemos ver na Figura 9.

Figura 9 – Cano gerado pelo comando de atribuição

Fonte: Acervo do autor

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

Figura 10 – Bloco de canos gerados pelo comando das Estruturas de repetição

Fonte: Acervo do autor

3.2.2 Especificação da Sintaxe *Ballcode*

Diferente do processo usual de compilação ou interpretação, os comandos na linguagem *Ballcode* são comparados com modelos pré-determinados de linhas que já possuem instruções, sendo que o usuário, ao digitar um comando, altera determinados pormenores, como os valores numéricos ao serem utilizados ou o tipo de comparação que deve ser feita.

Abaixo, a Tabela 2 demonstra os comandos de *Ballcode*, bem como a sintaxe correta para utilizá-los:

Tabela 2 – 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;**

3.3 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.

Nas Figuras 11 e 12 podemos observar, respectivamente, um código em *Ballcode* e o conjunto de canos gerados por este bloco de código.

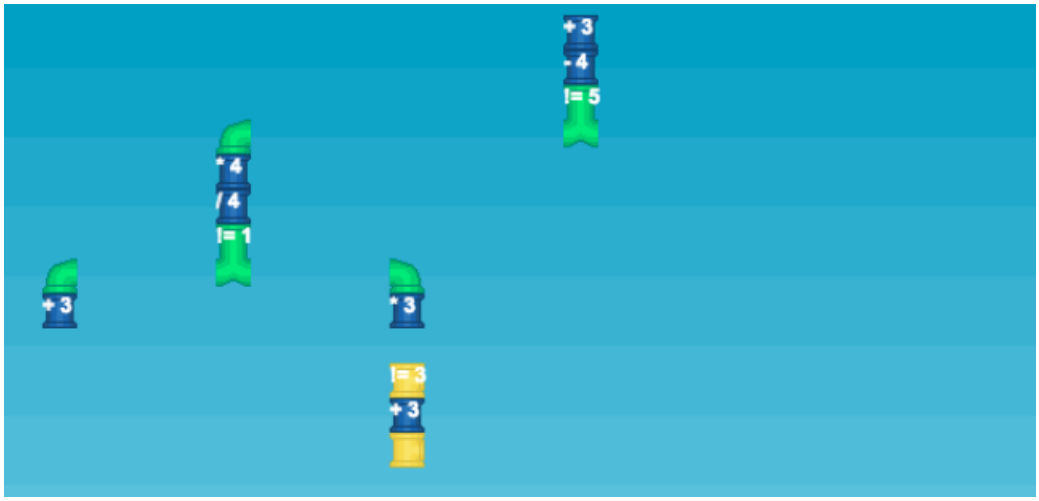
Figura 11 – Bloco de código em *Ballcode*

```
pipe is + 3;
pipe is - 4;

if ball == 5:
    pipe is * 4;
    pipe is / 4;
    if ball == 1:
        pipe is + 3;
    else:
        pipe is * 3;
    endif;
    while ball != 3:
        pipe is + 3;
    endwhile;
endif;
```

Fonte: Acervo do autor

Figura 12 – Conjunto de canos gerados pelo código descrito anteriormente



Fonte: Acervo do autor

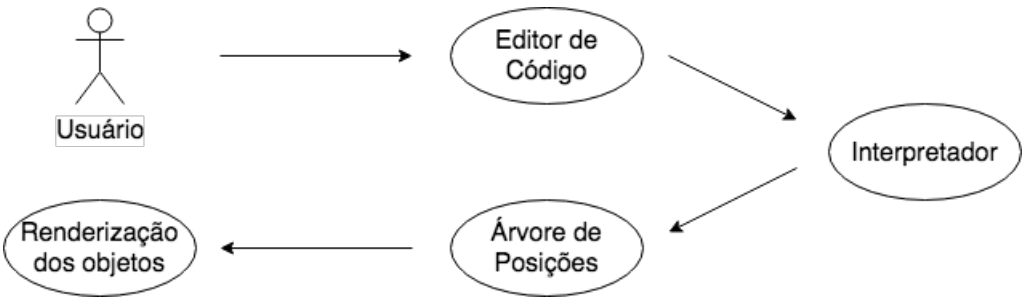
4 Arquitetura do *Ballgorithm*

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. Sendo assim, 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 13 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.

Figura 13 – Diagrama de Caso de Uso



Fonte: Acervo do autor

4.1 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 à certas funções presentes em um compilador.

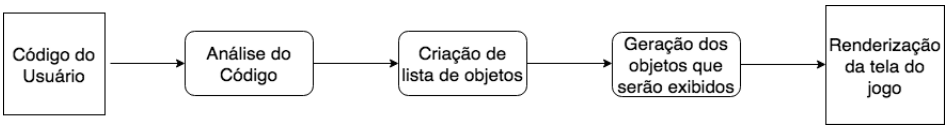
Considerando o modelo conceitual de um compilador, os passos por ele seguidos são análise do código, representação semântica e síntese, para então assim gerar o código executável (GRUNE et al., 2012).

De forma análoga ao compilador, o código do usuário, ao ser submetido, é analisado, gera uma lista com os objetos e sintetiza estes objetos nos objetos 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. Sendo assim, analisando as Figuras 14 e 15, é possível constatar a semelhança entre os modelos do compilador e do *Ballgorithm*, respectivamente.

Figura 14 – Modelo conceitual do compilador

Fonte: Adaptado de (GRUNE et al., 2012)

Figura 15 – Modelo conceitual da execução do *Ballgorithm*



Fonte: Acervo do autor

4.2 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.

4.2.1 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.

Sendo assim, 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.

4.2.2 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.

4.2.3 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 3, é possível ver as expressões regulares com as quais as linhas são comparadas.

Tabela 3 – 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*$/ /^\s*endwhile\s*;\s*$/

4.2.4 Criação dos objetos de tipo Linha

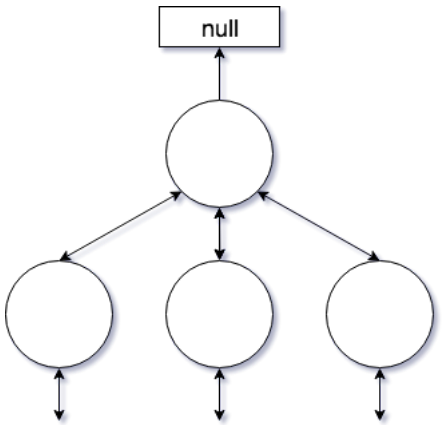
Cada linha, depois de classificada, é enviada a um vetor de linhas, contendo os seguintes dados: tipo de linha, bloco e conteúdo. O valor do conteúdo possui os elementos de texto presentes na *string* bruta, tendo em vista que estes elementos serão utilizados para definir as características dos objetos,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

4.2.5 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. Sendo assim, a árvore binária acabou sendo adaptada. A abstração que mais casava com a alocação dos canos era uma árvore ternária, já que os filhos de cada nó poderiam ser alocados à direita, esquerda ou abaixo. Além disso, cada nó aponta também para o seu nó pai, o que facilita os múltiplos acessos aos nós anteriores. Na Figura 16 podemos ver como os nós da árvore de posições são agrupados.

Figura 16 – Estrutura escolhida para a organização dos nós



Fonte: Acervo do autor

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.

4.3 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.

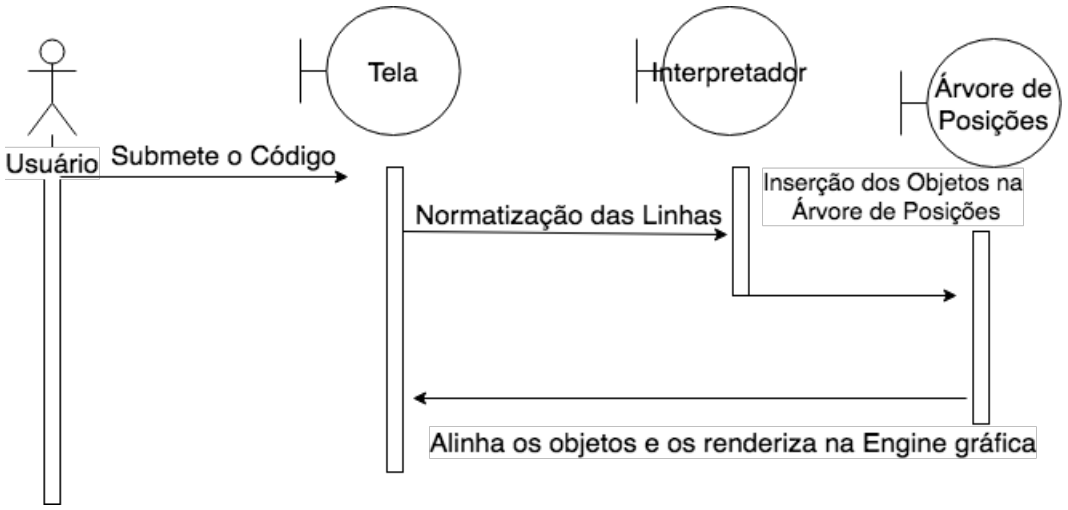
4.3.1 Instanciando os objetos

Os objetos de cada vetor são lidos, e deles são extraídos os seguintes dados: posições em (x,y) geradas durante a alocação dos objetos na árvore, valor numérico do nó, operação matemática (no caso de atribuições) e o valor booleano referente à comparação (no caso das estruturas condicionais e de controle). 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

4.3.2 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.

Figura 17 – Diagrama de sequência do *Ballgorithm*



Fonte: Acervo do autor

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 17, é mostrada a ordem de execução de cada um dos passos citados anteriormente.

5 Estudos de Caso

Tendo como objetivo exemplificar o funcionamento do *Ballgorithm* e detalhar os processos que ocorrem durante a sua execução, neste capítulo demonstraremos a criação de alguns percursos utilizando a linguagem *Ballcode*.

5.1 Caso 1: Alterando o valor das bolas

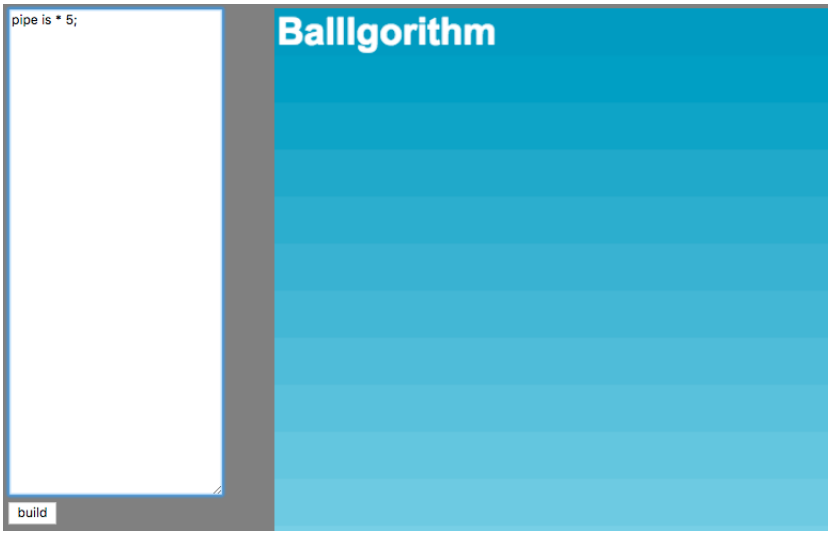
Foi proposto ao usuário a criação de uma máquina capaz de mudar o valor de todas as bolas de forma que, ao final do percurso, o valor das bolas seja igual ao valor inicial multiplicado por 5.

Sendo assim, tratando-se de uma alteração que deva ser feita em todas as bolas, não existem restrições para a execução destas alterações. Consequentemente, não há nenhuma necessidade de usarmos estruturas condicionais.

5.1.1 Escrevendo o código

Sabendo que o objetivo da máquina que deve ser criada é apenas multiplicar por 5 os valores das bolas que porventura forem criadas, o usuário pode cumprir esta tarefa facilmente com a criação de apenas um cano, utilizando a atribuição do valor da bola * 5, que pode ser feito através do comando **pipe is * 5;**. Podemos ver na figura 18 o usuário digitando seu código.

Figura 18 – Digitando o código que será utilizado

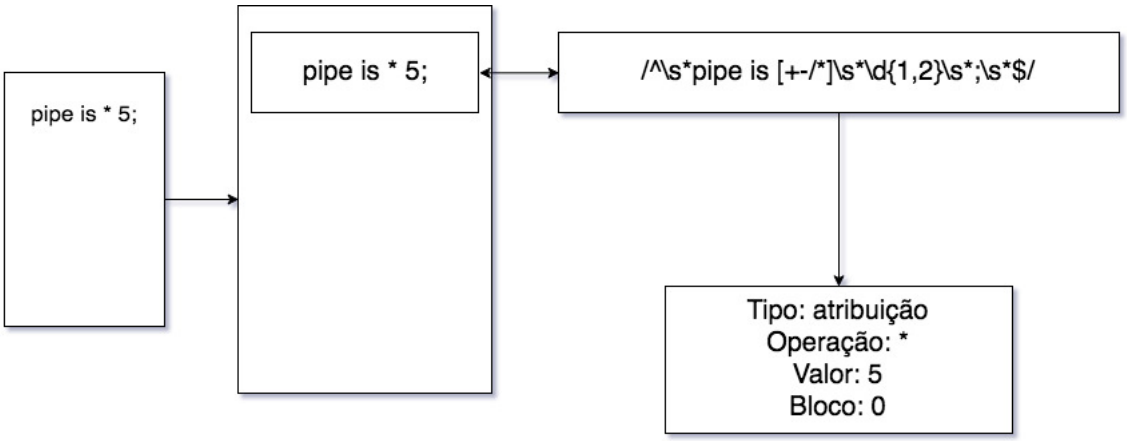


Fonte: Acervo do autor

5.1.2 Processamento do Código

A *string* "*rawLines*"é dividida de acordo com os símbolos de quebra de linha. Como neste caso só existe uma quebra de linhas, apenas uma linha é enviada ao vetor de linhas. Essa linha então é comparada com as expressões regulares referentes a cada linha válida de *Ballcode* e, caso a linha seja válida, é criado um objeto do tipo Linha. Além disso, o bloco de indentação da linha é armazenado neste objeto. Esses processos se encontram detalhados na Figura 19. Após isso, é criada a árvore de posições.

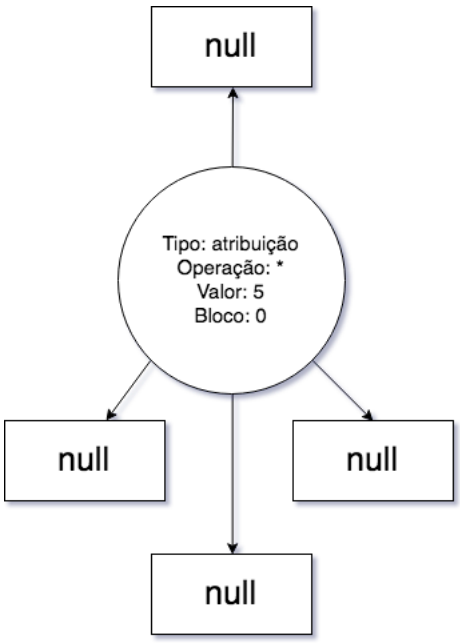
Figura 19 – Processo de interpretação do código



Fonte: Acervo do autor

Na Figura 20, podemos ver a árvore de posições. Porém, como só existe uma linha no código, a árvore só terá um elemento, a raiz, cuja chave é o objeto Linha, obtido anteriormente.

Figura 20 – Criação da Árvore de Posições



Fonte: Acervo do autor

Durante a criação da Árvore de Posições, os objetos tem suas posições em (x, y) geradas, sempre levando em consideração a sua posição com relação ao seu objeto pai. No caso do nó criado, como ele é a raiz, a sua posição é a posição inicial em y (0) e a posição inicial em x (o valor médio entre a posição mínima de x, que é zero, e a posição máxima de x, que é 800).

Tendo posse da posição em (x, y) de cada objeto, os objetos da árvore são alocados em vetores específicos para cada tipo de objeto. Após isso, cada elemento de cada vetor tem seus dados extraídos, e o objeto da *engine* gráfica é instanciado com base nesses dados. Na Figura 21, podemos ver o resultado do processo, com o objeto já renderizado.

Figura 21 – Objeto já renderizado na *engine* gráfica

Fonte: Acervo do autor

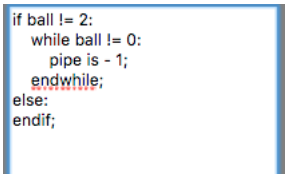
5.2 Caso 2: Alterando apenas algumas bolas

Propondo ao usuário que apenas bolas de valores 0 (zero) ou 2 (dois) sejam exibidas, é preciso criar uma máquina capaz de verificar os valores das bolas e, além disso, alterar os valores das outras bolas, para que elas tenham o valor desejado.

5.2.1 Escrevendo o código

Tendo o objetivo de apenas retornar bolas com os valores 0 (zero) ou 2 (dois), é preciso primeiramente checar quais bolas devem ter seus valores alterados ou não. Além disso, é preciso alterar os valores dos elementos que serão alterados repetidamente, de um em um. É importante lembrar que os valores numéricos presentes no *Ballgorithm* sempre são valores positivos. Na Figura 22 podemos ver o código que será utilizado para a resolução deste problema.

Figura 22 – Código digitado pelo usuário

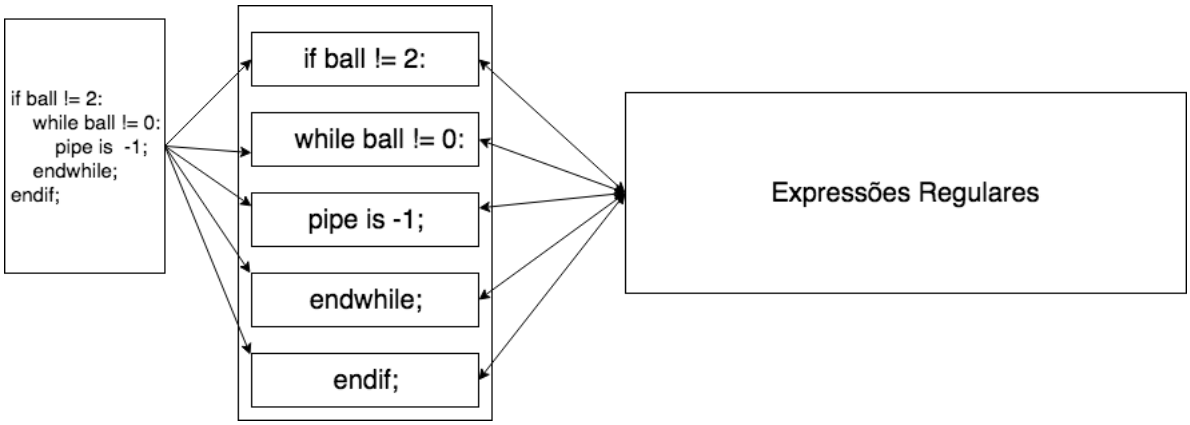


Fonte: Acervo do autor

5.2.2 Processamento do código

Na Figura 23 código do usuário é dividido em linhas e estas são alocadas em um vetor. Este vetor tem cada linha comparada com as expressões regulares referentes aos comandos do jogo. O espaço no início de cada linha também é armazenado, levando em conta os blocos de código de cada uma. Após validadas, as linhas são adicionadas à árvore de posições.

Figura 23 – Processando as linhas do código *engine* gráfica



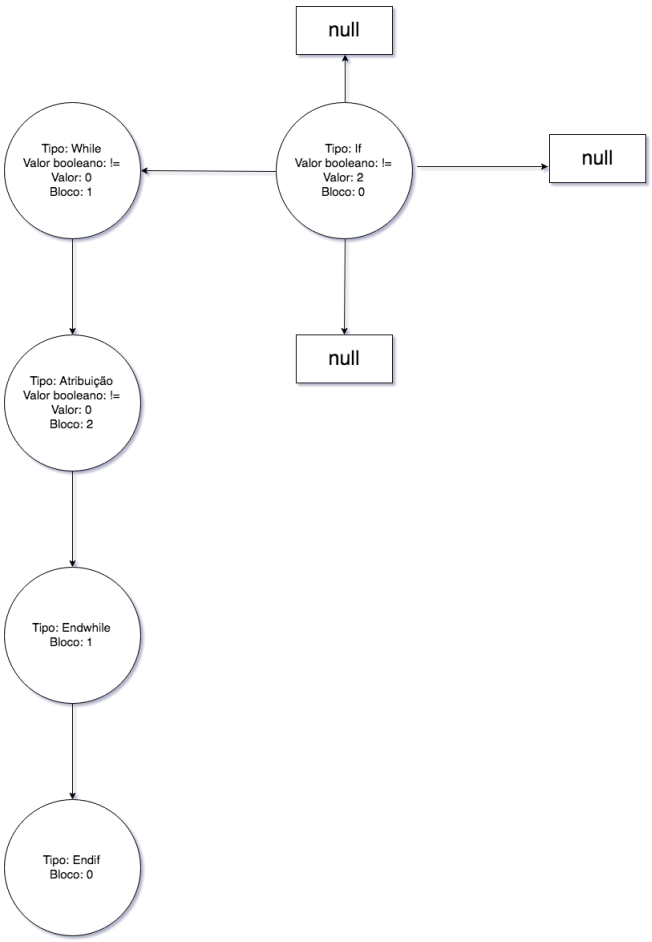
Fonte: Acervo do autor

Na árvore de posições os elementos terão suas posições em (x, y) definidas. Ao ler um objeto do tipo if, ele cria um nó para cada possibilidade do if, pros quais as bolas serão enviadas de acordo com as condições definidas. Cada nó adicionado à esquerda ou à direita de seu pai tem sua posição no eicho x alterada, sendo essa posição a média entre a posição do seu pai e a posição do seu avô.

Os nós adicionados abaixo de seus pais herdam a posição em x dos mesmos, e todos os objetos (com exceção dos objetos referentes à estruturas condicionais) têm suas posições em y obtidas através de um incremento da posição de seus pais. É possível observar com mais clareza na Figura 24 a organização dos elementos quando na árvore de posições.

Após serem alocados na árvore e obterem suas posições em (x, y), os objetos são

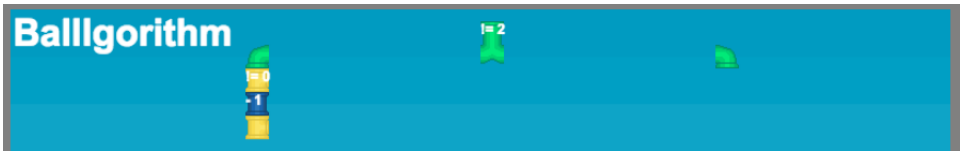
Figura 24 – Visualizando a árvore de posições deste código



Fonte: Acervo do autor

alocados em vetores, que serão percorridos afim de que cada objeto seja renderizado com a *engine*. Na Figura 25, podemos ver os objetos renderizados no jogo.

Figura 25 – Visualizando os objetos já renderizados no jogo



Fonte: Acervo do autor

6 Conclusão

O objetivo deste trabalho foi alcançado quanto à descrição e desenvolvimento do *Ballgorithm*. O processo de desenvolvimento culminou em uma ferramenta que atendesse os requisitos idealizados, onde o usuário pudesse escrever um código simples e de fácil assimilação graças ao *feedback* visual fornecido pela renderização dos objetos enquanto partes de um jogo. Além disso, apesar de existirem outras ferramentas e ambientes com o mesmo objetivo, nenhuma delas supre as necessidades que o projeto visa solucionar.

Um resultado direto do trabalho é a possibilidade do usuário do *Balgorithm* poder criar fases próprias para o jogo com nível de dificuldade escolhido para ele. Seria interessante criar uma métrica formal para avaliação do impacto em motivação e interesse entre classes de alunos ingressantes em cursos de graduação, o que pode ser um promissor trabalho futuro.

Acredita-se que o *Balgorithm* possa vir a ser uma plataforma de aprendizagem interessante e versátil ao ponto de ser personalizada para cenários de uso diversos, como para o ensino de crianças e de pessoas na terceira idade. Para tanto, os objetos usados no jogo podem ser personalizados visando essa finalidade, o que poderia ser um outro interessante trabalho futuro.

Além do desenvolvimento do projeto, ainda há um longo caminho a ser percorrido. Primeiramente, é preciso efetuar testes que visam validar decisões de *design* e usabilidade, baseando-se nos resultados obtidos. Dessa forma, novas funcionalidades podem ser implementadas no protótipo do *Ballgorithm*, como novas interfaces de usuário visando exclusivamente a criação de novos níveis, melhoria de performance do sistema e novas interfaces visando dispositivos *mobile*.

Referências

ADAMS, E. *Fundamentals of Game Design*. [S.l.]: New Riders, 2014. Citado na página 17.

BARBOSA, L.; FERNANDES, T.; CAMPOS, A. Takkou: Uma ferramenta proposta ao ensino de algoritmos. *CSBC*, 2011. Citado 2 vezes nas páginas 17 e 18.

BATTISTELLA, P. E.; WANGENHEIM, A. v.; VON, W. C. G. Sortia:um jogo para ensino de algoritmo de ordenação: Estudo de caso na disciplina de estrutura de dados. *SBIE*, 2012. Citado 2 vezes nas páginas 16 e 17.

CODECADEMY. 2018. (https://www.codecademy.com/about). Citado na página 19.

DRUMOND, R.; SALLES, C. Wanda: a framework to develop card based games to help motivate programming students. *SBGames*, 2014. Citado na página 16.

GRUNE, D. et al. *Modern Compiler Design*. [S.l.]: Springer, 2012. Citado na página 27.

HUITT, W. *Motivation to Learn: An Overview*. 2011. Disponível em: <http://www.edpsycinteractive.org/topics/motivation/motivate.html>. Citado na página 13.

MEC/INEP. *Taxa de conclusão média dos cursos de graduação presenciais*. 2017. Disponível em: <http://www.observatoriodopne.org.br/metas-pne/12-ensino-superior/estrategias/12-3-fluxo/indicadores>. Citado na página 13.

PALMEIRA, L. B.; SANTOS, M. P. Evasão no bacharelado em ciência da computação da universidade de brasília: análise e mineração de dados. 2015. Citado na página 13.

RADTKE, P. V. W.; BINDER, F. V. Chien 2d: A multiplatform library to teach the c language through games programming. *SBGames*, 2010. Citado na página 15.