Jogo de batalha 2d orientado à execução de scripts de uma linguagem de domínio específico.

Samuel José Eugênio

Ciência da Computação - Universidade do Sul de Santa Catarina (UNISUL) - Tubarão - SC - Brasil

samuelgenio28@gmail.com

Abstract. Just like on the internet, games are already part of the daily life of young people around the world, according to IBGE (2017), more than 69% of the Brazilian population, over 10 years old have access to the internet. Game-based learning is becoming more and more continuous as it brings knowledge in a fun and interactive way. Following this learning methodology, it is interesting to develop a solution that adds fun and encourages problem solving through challenges. Through its own computational language, the player is motivated to implement his own character for competitions.

Resumo. Assim como a internet, os jogos já fazem parte do cotidiano de jovens do mundo todo, segundo IBGE (ano de 2017), mais de 69% da população brasileira, acima de 10 anos possui acesso à internet. A aprendizagem baseada em jogos torna-se cada vez mais sólida, por trazer o conhecimento de forma divertida e interativa. Seguindo essa metodologia de aprendizagem, torna-se interessante o desenvolvimento de uma solução que agregue diversão e estímulo à solução de problemas através de desafios. Através de uma linguagem computacional própria, o jogador é motivado a implementar seu próprio personagem para competições.

Palavras-chave: aprendizagem, jogos, programação

1. Introdução

Seguindo o fluxo crescente da tecnologia, segundo [Prensky 2001], os alunos de hoje que já cresceram na ambientação tecnológica, com o advento da era digital, passam milhares de horas conectados a estes dispositivos e aprendem de maneira diferente das gerações pré-digitais (considerados imigrantes digitais). Considerado nativos digitais, esses alunos passam dificuldade no aprendizado devido a educadores imigrantes digitais, que usufruem de uma linguagem pré-digital, onde ensinam indivíduos que possuem um dialeto diferente.

Atualmente os jogos digitais seguem a mesma linha de crescimento como superproduções hollywoodianas, chegando a movimentar financeiramente valores maiores que filmes. O ser humano sempre aprende com o ambiente que ele convive e isso não é diferente para os jogos. Com uma linguagem própria, que alinha narrativas audiovisuais, sistemas de regras lúdicas e possibilidades diversas de interatividade, o

videogame permite que jogadores explorem e interajam com ambientes digitais [Bello 2017].

Tendo em vista o ambiente em que vivemos, onde a inovação transborda em nossas mãos, torna-se interessante aliar jogos eletrônicos com educação voltada para a área tecnológica. O mundo digital oferece novas formas de criatividade, aprendizagem, empreendimento e inovação, mas essas oportunidades exigem habilidades [Palfrey & Gasser, 2008].

Segundo pesquisa realizada pelo [Posso 2016], em um grupo de mais dez mil crianças australianas entre 15 e 17 anos, nas áreas de matemática, leitura e ciências, constatou-se que os indivíduos que jogam jogos online diariamente pontuaram até 15 pontos a mais em comparação com os alunos que não jogavam. Ao contrário das redes sociais, os jogos permitem que os alunos apliquem e pratiquem o conhecimento adquirido por meio de diversos desafios.

A capacidade de resolução de problemas é impactada diretamente pela utilização adequada dessas tecnologias, pois se o indivíduo possui conhecimento de análise para uma determinada situação, consequentemente poderá escolher a melhor forma de solução. Essa habilidade de resolução de problemas são diferenciais para diversas áreas que vivenciamos no dia a dia, podendo variar desde uma questão de matemática básica até problemas profissionais.

Um jogo que alia a interatividade por meio da tomada de decisão, induzindo o aprendizado de como uma linguagem tecnológica funciona, que explora a capacidade de resolução de problemas para alcançar objetivos, torna-se uma excelente opção para indivíduos que buscam desafios, explorando a lógica de um jeito divertido podendo aflorar novas capacidades e direcionar um jovem à área tecnológica.

2. Objetivos

Desenvolvimento de um jogo de batalha 2D, PVP (player versus player) e PVE (player versus environment) onde o jogador necessita aprender a pré programar seu personagem para batalha. Deste modo, a solução é dividida nos seguintes ramos de desenvolvimento:

- Desenvolver linguagem para a programação do script de ação do personagem.
- Interface para a programação do script.
- Compilador e máquina virtual para execução da batalha.
- Jogo, que unifica a máquina virtual e a interface da aplicação.

3. Trabalhos Relacionados

Atualmente existem diversas soluções com a mesma intenção do jogo em questão, para diversos tipos de faixa etária, alguns deles são:

3.1. Croac

Jogo brasileiro, em que o jogador necessita auxiliar o sapo, aprendiz de rei, a voltar para casa através de lógica de blocos. Indicado para crianças a partir de 4 anos. Trazendo o conceito de musicalização, os jogadores deverão desenvolver sequências de instruções de movimentações guiando o sapo através da melodia de uma flauta.

3.2. Cargo-Bot

Jogo lançado em 2012, inicialmente somente para sistemas IOS, posteriormente disponibilizado para Windows, traz a mecânica de instrução de blocos. O jogador necessita controlar uma garra e empilhar caixas de acordo com o desafio proposto, a cada nível o desafio fica mais trabalhoso e para conseguir maior pontuação o jogador necessita desenvolver menor quantidade de instruções possíveis.

3.3. Lightbot

Jogo disponibilizado para diversas plataformas, traz também mecânicas de instruções em blocos, inclusive traz duas versões, a primeira para crianças de 4 a 8 anos e uma segunda, para maiores de 8 anos. O jogador necessita controlar um pequeno robô e alcançar determinados pontos no mapa, o jogo força o jogador a desenvolver as rotinas de movimentações cada vez menores, para obter maior pontuação.

3.4. Robocode

Jogo lançado em meados dos anos 2000, é um jogo de programação, onde o objetivo é desenvolver as mecânicas de um tanque de guerra para batalhar contra outros tanques utilizando Java ou NET. As batalhas de robôs são executadas de forma fluída sem interferência do jogador. O jogo requer conhecimento intermediário de programação, pois para controlar o robô é necessário implementar diversos eventos, requerido conhecimento em orientação a objetos. Robocode traz uma proposta de aprendizagem robusta, possibilitando aplicação de campeonatos com grandes premiações e diversas equipes [Robocode Brasil].

4. Diferenciais

Em comparação com as soluções já existentes, que podem ser divididas em jogos mais voltados ao público infantil (lógica em blocos) ou para desenvolvedores intermediários e avançados, a solução proposta tem o intuito de alcançar público com breve conhecimento em lógica e programação. Levando em consideração o grau de dificuldade para desenvolvimento do script e o quão desafiador pode ser elaborar soluções inovadoras.

5. Solução implementada

O jogo implementa batalhas que são executadas através da compilação de scripts desenvolvidos por cada jogador. O Campo de batalha 2D é dividido em pequenas áreas

chamadas de nó. Esses nós indicam os pontos de movimentação que o personagem pode se locomover, não podendo estar no mesmo nó do oponente.

Antes mesmo de entrar em uma batalha o jogador precisará configurar seu personagem para o desafio através de uma linguagem própria de desenvolvimento, sendo possível pré-programar a movimentação e ações de ataque e de defesa do personagem.

5.1. Linguagem

Baseada principalmente na linguagem LMS, que é uma linguagem com fins didáticos criada a partir de elementos da linguagem Pascal. Utilizando símbolos e estruturas da linguagem Java, foi desenvolvida a DLS (Linguagem Específica de Domínio) para o jogo. Segundo [MERNIK, 2005] DLSs providenciam notações e construções sob medida para um domínio de aplicativo específico. A Linguagem é fracamente tipada, onde sua principal semelhança com LMS pode ser notada na declaração de variável, que deve ser feita no início do script, porém a estrutura de execução baseada em métodos e chamadas são inspiradas na linguagem Java.

A Linguagem possui dois tipos de dados, numeral (inteiro) e literal (string), onde não é necessário identificar o tipo da variável, ela é dinamicamente identificada ao receber seu primeiro valor. Porém caso uma variável do tipo inteiro, após inicializada, receba um valor literal, o compilador irá acusar erro. A Linguagem é case-sensitive e permitirá a criação de variáveis personalizadas, assim como métodos, desde que não possuam o nome de nenhum método ou variável reservada.

O Jogo possui um compilador próprio. Um Compilador é definido como um processo que ao receber como entrada um programa fonte o traduz para um programa equivalente em outra linguagem [LOUDEN, 2004]. O processo de compilação possui o fluxo como mostrado na figura 1:

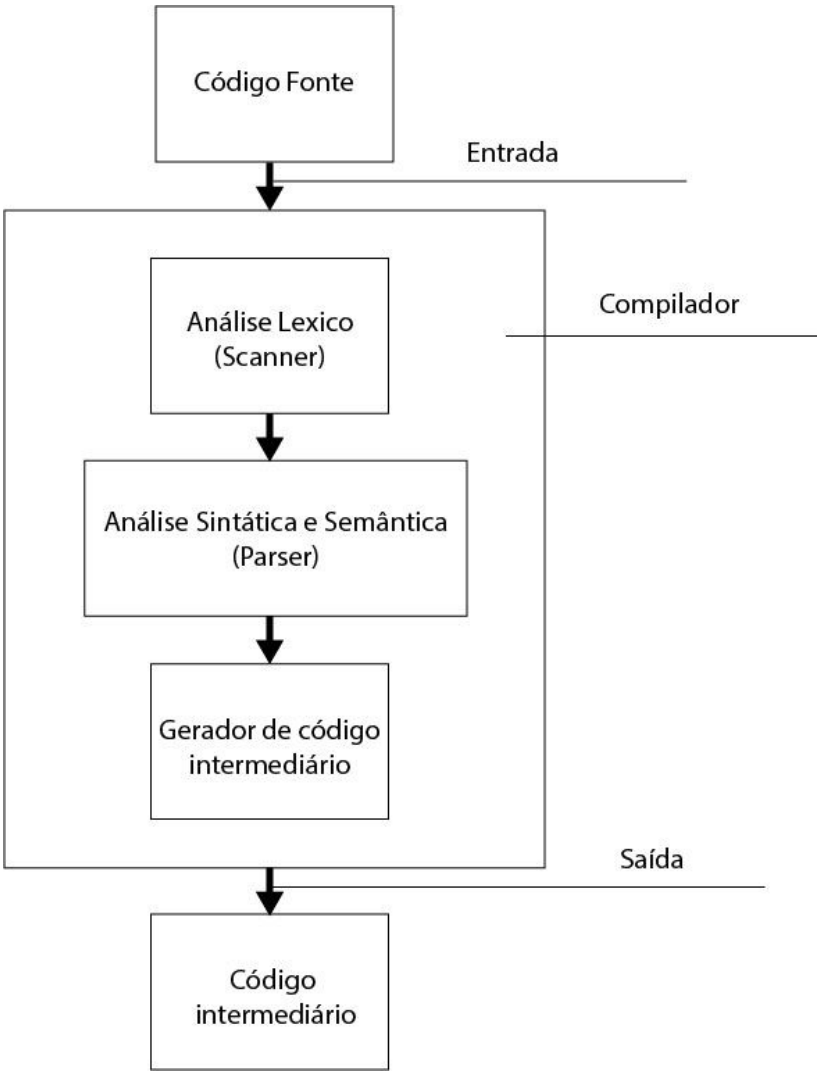

Figura 1. Fluxo de compilação.
Fonte: Autor, 2019

Para a geração das matrizes de parse e léxica, foi utilizado o software GALS, que é uma ferramenta de Software Livre para geração de analisadores léxicos e sintáticos [Gesser, 2003]. O software permite definir os tokens da linguagem juntamente com as regras gramaticais.

5.2. Definições Léxicas

A análise léxica examina o programa fonte e reconhece os tokens, as palavras reservada e símbolos são exemplos de tokens [HOPCROFT, 2003]. Também conhecido como scanner, ao efetuar a análise dos tokens, caso encontre algum elemento não existente na linguagem, o processo é encerrado.

Tokens definidos da linguagem:

- **Operadores e comparadores:** +; -; *; /; ==; >; >=; <; <=; <>;
- **Símbolos:** =; **;**; **:**; **,**; **.**; **(**; **)**; {; }; **'**;

- **Palavras reservadas: CONTROLLER**; **ID**; **_**; **IF**; **ELSE**; **WHILE**; **OR**; **AND**; **NOT**; **FOR**; **AHEAD**; **JUMP**; **ATTACK**; **TURN**; **getEnemyPos**; **getEnemyName**; **getEnemyHealth**; **getEnemyDirection**; **getName**; **getHealth**; **getPos**; **getDirection**; **STOP**;

Identificadores (**IDENT**) e literais são definidos através da seguinte expressão regular: **[A-Za-z] ([A-Za-z] | [0-9]| _)***. Inteiros (**INTEIRO**) são definidos da seguinte maneira: **[0-9]+**. Os comentários são definidos no padrão **'/* comentário */'**, estes são ignorados pelo analisador.

5.3. Definições Sintáticas

Durante a análise sintática, os tokens já reconhecidos são validados de acordo com as regras de produção. Esses tokens são separados em dois grupos, símbolos terminais e não terminais. Os símbolos terminais são todos os tokens reconhecidos na análise léxica, já os não terminais podem ser definidos como o conjunto de elementos que definem a linguagem, são eles:

<PROGRAMA>; **<BLOCO>**; **<LID>**; **<REPIDENT>**; **<DCLVAR>**; **<LDVAR>**; **<LTVAR>**; **<DCLEVENT>**; **<EVEPAR>**; **<BODY>**; **<REPCOMANDO>**; **<COMANDO>**; **<PARAMETROS>**; **<REPPAR>**; **<ELSEPARTE>**; **<VARIAVEL>**; **<EXPRESSAO>**; **<REPEXPSIMP>**; **<EXPSIMP>**; **<REPEXP>**; **<TERMO>**; **<REPTERMO>**; **<FATOR>**; **<ACTNAME>**; **<ACTNAMEMET>**;

Tais elementos são utilizados para formar as regras de produção que por sua vez, são formadas por um grupo de regras sintáticas que representam toda a linguagem. As regras de produção da linguagem podem ser vista na figura 2:

```
PROGRAMA ::= "CONTROLLER" IDENT "{" BLOCO "}"
BLOCO ::= DCLVAR DCLEVENT
LID ::= IDENT REPIDENT
REPIDENT ::= "" | "," IDENT REPIDENT
DCLVAR ::= "" | "ID" IDENT "=" LTVAR ";" LDVAR'
LDVAR ::= "" | IDENT "=" LTVAR ";" LDVAR
LTVAR ::= INTEIRO
LTVAR ::= "'" LITERAL "'"
DCLEVENT ::= "" | "_" IDENT EVEPAR BODY DCLEVENT
EVEPAR ::= "(" LID")"
BODY ::= "{" COMANDO REPCOMANDO "}"
REPCOMANDO ::= "" | COMANDO REPCOMANDO
COMANDO ::= "" | IDENT "=" EXPRESSAO ";" | BODY | "_"
IDENT PARAMETROS | "IF" EXPRESSAO BODY ELSEPARTE | "WHILE"
"(" EXPRESSAO ")" BODY | "FOR" "(" IDENT "=" EXPRESSAO ";"
EXPRESSAO ")" BODY | ACTNAME "(" EXPRESSAO ")" ";" |
ACTNAMEMET "(" ")" ";"
PARAMETROS ::= "(" EXPRESSAO REPPAR ")"
REPPAR ::= "" | "," EXPRESSAO REPPAR
ELSEPARTE ::= "" | "ELSE" BODY
VARIAVEL ::= IDENT
EXPRESSAO ::= EXPSIMP REPEXPSIMP
REPEXPSIMP ::= "" | "==" EXPSIMP | "<" EXPSIMP | ">"
EXPSIMP | ">=" EXPSIMP | "<=" EXPSIMP | "<>" EXPSIMP
EXPSIMP ::= "+" TERMO REPEXP | "-" TERMO REPEXP | TERMO
REPEXP
REPEXP ::= "" | "+" TERMO REPEXP | "-" TERMO REPEXP | "OR"
TERMO REPEXP
TERMO ::= FATOR REPTERMO
REPTERMO ::= "" | "*" FATOR REPTERMO | "/" FATOR REPTERMO
| "AND" FATOR REPTERMO
FATOR ::= INTEIRO | "'" LITERAL "'" | "(" EXPRESSAO ")" |
| "NOT" FATOR | VARIAVEL | ACTNAMEMET "(" ")"
ACTNAME ::= "AHEAD" | "JUMP" | "RUN" | "ATTACK" | "TURN"
ACTNAMEMET ::= "getEnemyPos" | "getEnemyName" |
"getEnemyHealth" | "getEnemyDirection" | "getName" |
"getHealth" | "getPos" | "getDirection" | "STOP"
```

Figura 2. Sintaxe da linguagem no formato BNF (Backus - Naur form).
Fonte: Autor, 2019

5.4. Definições Semânticas

Responsável por verificar o significado das instruções. Segundo [AHO, 2008] os tokens de entrada são analisados sintaticamente, avaliando as regras semânticas de cada nó, os resultados dessa análise podem gerar códigos intermediários finais da aplicação ou emitir mensagens de erro. Essa etapa é executada juntamente com a análise sintática, onde são efetuadas validações que não podem ser feitas na etapa anterior, desta maneira sua principal função é garantir que o script analisado possa se converter em código intermediário (fluxo mostrado na figura 1).

Denominada ações semânticas, essas validações identificam se o código fonte está coerente com a linguagem. Um exemplo de validação semântica, é definido como o controle de declaração de identificadores, como mostrado na figura 3, onde o identificador declarado é **movimentacao**, porém o identificador utilizado é **velocidade**. O Analisador resultará em erro pois o identificador **velocidade** não foi declarado.

```
Controller Controle {
    id movimentacao = 0;

    _onAlive() {
        velocidade = 10;
    }
}
```

```
COMANDO ::= IDENT #ACAO114 "=" <EXPRESSAO> #ACAO115 ";"
```

Figura 3. Exemplo de validação semântica.
Fonte: Autor, 2019

A ação semântica responsável por validar a declaração de identificadores é a ação número **#114**, feita logo após a identificação de um token do tipo **IDENT**. Na mesma regra de produção é possível identificar a ação **#115**, essa por sua vez é responsável por gerar o código intermediário de atribuição ao identificador.

Algumas das ações semânticas são:

- Ao utilizar um identificador, é verificado se o mesmo já foi declarado;
- Verificar a existência de identificadores e métodos duplicados;
- Após a tipagem de um identificador, se o mesmo está sendo utilizado conforme seu tipo;
- Na chamada de métodos auxiliares, assim como para os identificadores, sua é existência verificada;
- Verificar se o método principal onAlive(), foi declarado no script;

- Verificar se a quantidade de parâmetros de um método chamado é coerente com a quantidade esperado pelo método.

Algumas características da linguagem:

- Fracamente tipada, porém após receber um valor não pode ter seu tipo alterado;
- Linguagem case-sensitive;
- Unicidade de identificadores e métodos;

5.5. Estrutura e funcionamento

Toda a movimentação do personagem é baseada na chamada dos métodos de ações ou de obtenção de informações.

Ações:

- **ahead()**: Movimenta o personagem 1 nó para frente, caso o inimigo esteja na posição em questão a movimentação não é executada;
- **jump()**: Executa a movimentação de salto, caso esteja executado enquanto o personagem está se movimentando o personagem salta durante a movimentação.
- **attack()**: Executa movimentação e ataque;
- **turn()**: Altera a direção do personagem;
- **stop():** Para a movimentação do personagem;

Informações:

- **getEnemyPos()**: Obtém o nó de posição atual do inimigo;
- **getEnemyName():** Obtém nome do inimigo;
- **getEnemyHealth():** Obtém pontos de vida restantes do inimigo;
- **getEnemyDirection():** Obtém direção que o inimigo está caminhando (maior que 0 para direita, menor que 0 para esquerda);
- **getPos()**: Obtém o nó de posição atual do personagem;
- **getName():** Obtém nome do personagem;
- **getHealth()**: Obtém pontos de vida restantes do personagem;
- **getDirection()**: Obtém direção do personagem (maior que 0 para direita, menor que 0 para esquerda);

Figura exemplo da estrutura de um script:

```
1  Controller SimpleMov {
2
3      id enemyPos = 0; distance = 0;
4
5      _onAlive() {
6
7          enemyPos = getEnemyPos();
8
9          distance = getPos();
10
11         _turnDirection();
12
13         distance = enemyPos - distance;
14
15         if (distance == 1) {
16             attack(1);
17         } else {
18             ahead(1);
19         }
20     }
21
22     _turnDirection() {
23         if (enemyPos > distance && getDirection() < 0) {
24             turn();
25         } else {
26             if (enemyPos < distance && getDirection() > 0) {
27                 turn();
28             }
29         }
30     }
31 }
```

Figura 4. Script de execução.
Fonte: Autor, 2019

É possível identificar a semelhança com classes da Linguagem Java, onde existe a organização por métodos. Logo na linha 1 é definido o nome da classe **SimpleMov** que é utilizado para localizar o mesmo na tela de seleção de personagem.

Na linha 3 segue a definição de variáveis utilizadas em todo o escopo. Logo em seguida (linhas 5 à 20) existe o método **_onAlive** que por sua vez chamado ao iniciar a execução do script. Ao finalizar a execução de todo conteúdo do método, a máquina virtual volta a executar o mesmo novamente, o processo se repete enquanto o personagem possuir pontos de vida. A partir da linha 22 à 30 contém o método auxiliar **_turnDirection**, podendo conter qualquer nomenclatura de acordo com a lógica do desenvolvedor.

Não há limites para criação de métodos auxiliares, mesmo métodos não utilizados são validados pelo compilador.

6. O Jogo

Buscou-se desenvolver a interface de maneira a facilitar a experiência do usuário. O Fluxo para execução do jogo é estabelecido como mostrado na figura 5:

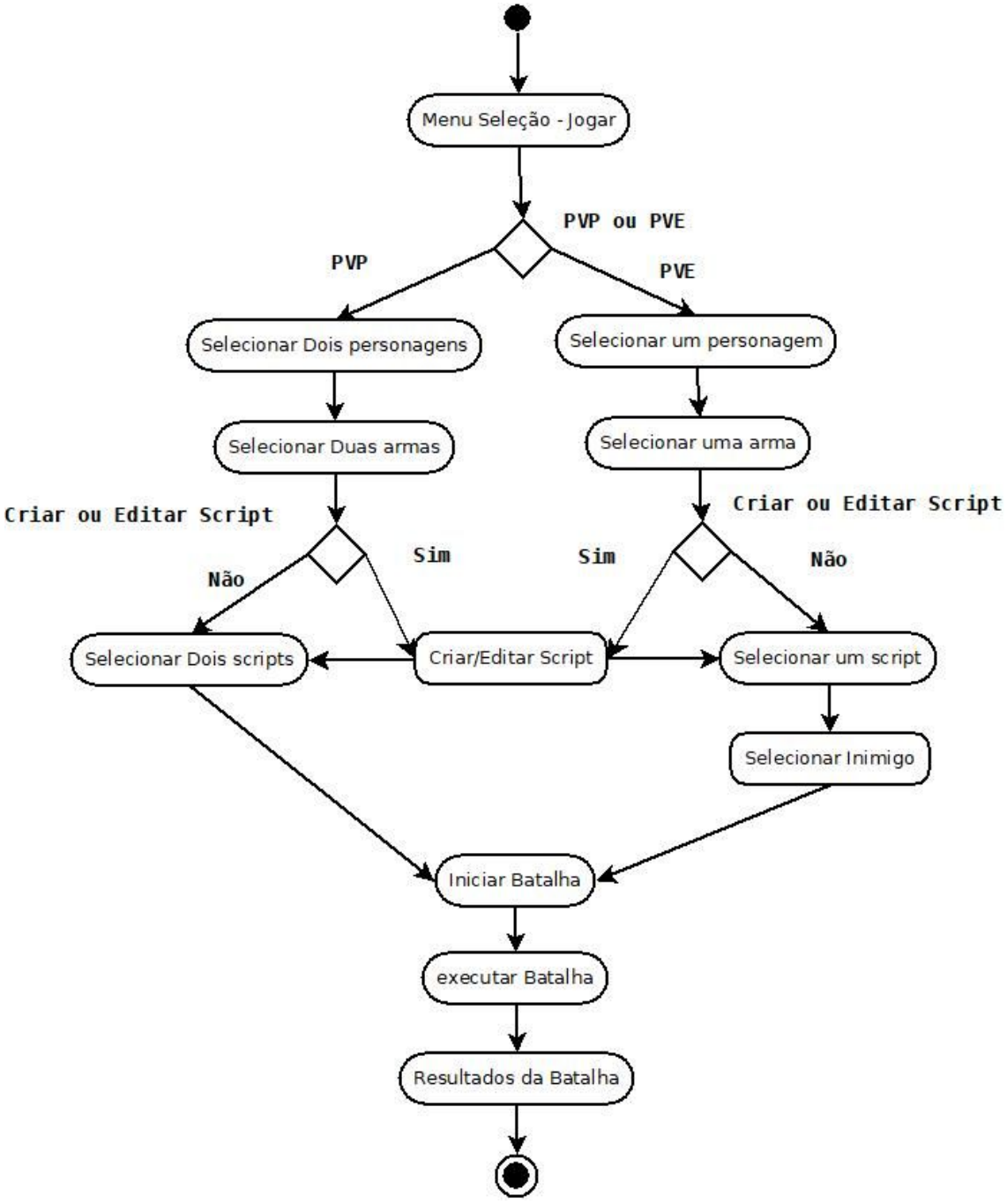

Figura 5. Fluxo de processo do jogo.
Fonte: Autor, 2019

6.1. Pré batalha

Ao primeiro contato com o jogo, o jogador visualiza o menu principal, que contém duas opções, como mostra a figura 6, os botões **Jogar** e **Sobre**:

Figura 6. Menu Principal.
Fonte: Autor, 2019

O Menu **Sobre** direciona o jogador à tela com informações sobre a criação do jogo e sua finalidade.

Ao selecionar **Jogar**, o jogador é direcionado à tela de preparação de batalha onde deve selecionar seu personagem, sua arma e seu script de movimentação.

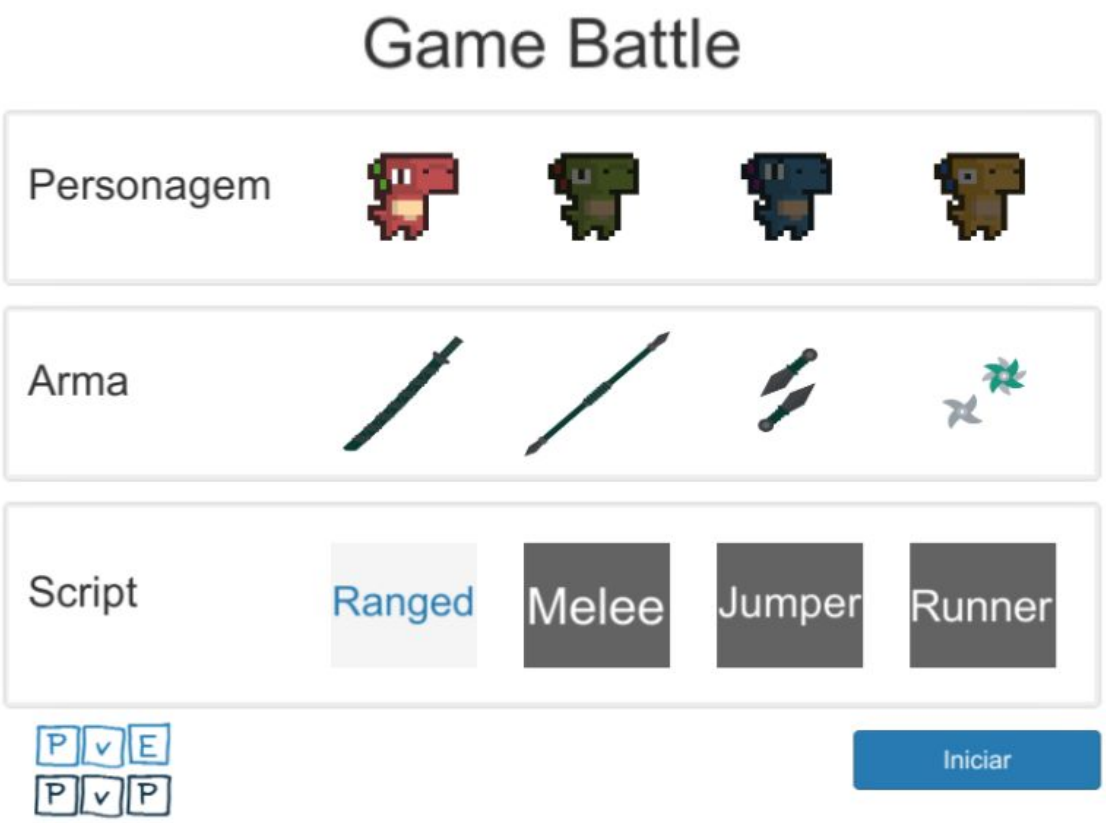

Figura 7. Tela de preparação para a batalha.
Fonte: Autor, 2019

Para iniciar um jogo PVE, é necessário selecionar apenas um item de cada opção, como mostrado na figura 7, onde o personagem foi definido como vermelho, a arma como shuriken e por fim seu script como Ranged. Para habilitar o modo PVP, é necessário selecionar dois itens de cada opção.

Os personagens não possuem diferença em seus atributos, todos possuem uma barra de vida com 100 pontos, se distinguem somente em suas cores.

As armas possuem atributos distintos, que são:

- Katana: Força: 13; Tempo de recarga (intervalo entre ataques, caso exista chamada de ataque durante o tempo de recarga, a ação é desconsiderada): 0.4 segundos.
- Lança: Força: 15; Tempo de recarga: 0.5 segundos.
- Kunai: Força: 10; Tempo de recarga: 0.5 segundos.
- Shuriken: Força: 5; Tempo de recarga: 1 segundo

O jogo permite ao jogador a edição de seu script de batalha, através do duplo clique sobre em um script selecionado, onde é aberto o editor do próprio do jogo.

6.2. Editor

O Editor (figura 8) permite a visualização e validação do script.

Figura 8. Editor.
Fonte: Autor, 2019

Durante o desenvolvimento do script é possível executar a validação (botão **Validar**), onde o Editor irá executar a compilação e informará ao usuário a situação. Caso ocorra algum erro na compilação a descrição do erro será exibida na barra inferior da tela. Ao finalizar, é necessário clicar no botão **Salvar**.

6.3. A batalha

Durante o carregamento da batalha o script de execução de cada jogador é carregado, executando assim a compilação completa.

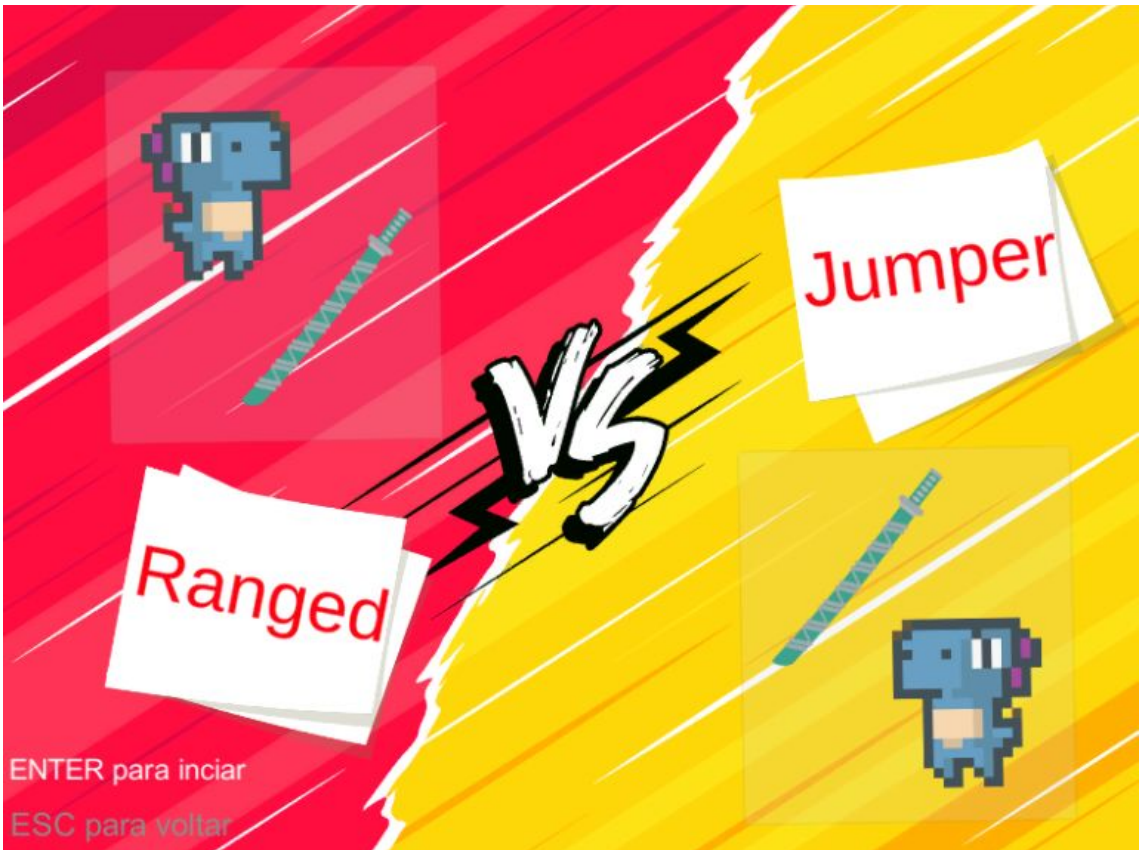

Figura 9. Tela de Carregamento da batalha.
Fonte: Autor, 2019

A arena possui 26 nós para movimentação dos personagens, sendo que cada personagem tem seu início em uma extremidade da tela. No início da batalha o contador é iniciado com o valor de 60 segundos, batalhas não devem durar mais que 1 minuto. Quando o contador chegar a zero a batalha é finalizada sendo o vencedor o personagem que possuir maior quantidade de vida.

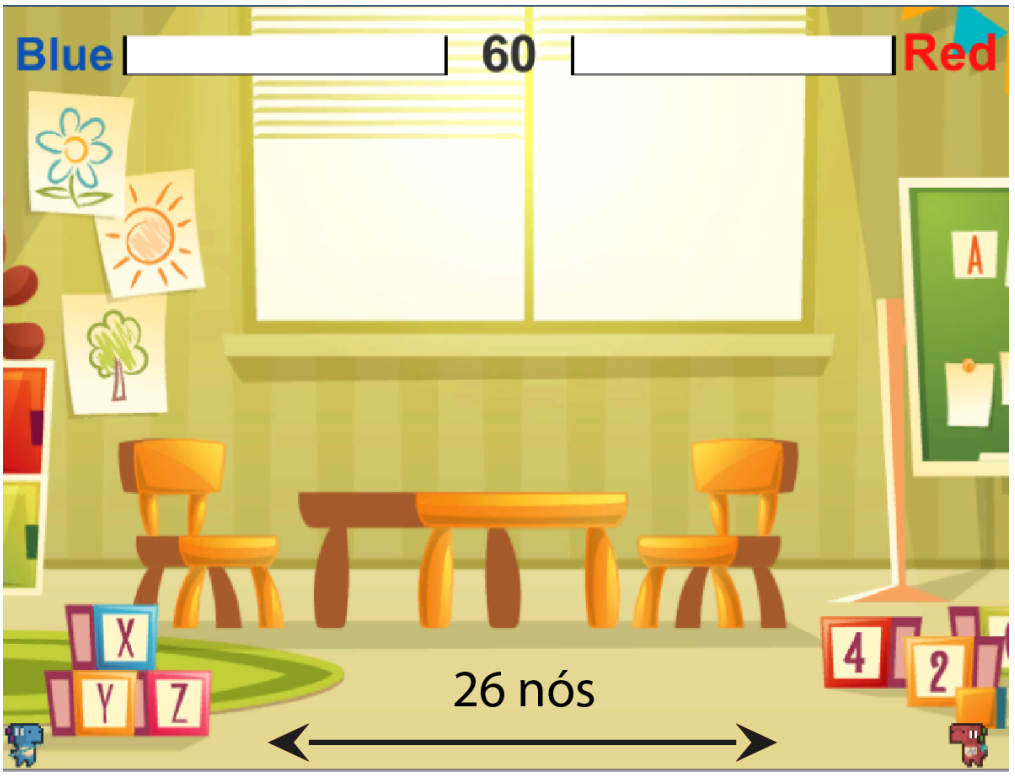

Figura 10. Representação da arena de batalha.
Fonte: Autor, 2019

Para garantir que a execução dos scripts de ambos os jogadores sejam simultâneos, a execução dos script foi dividida em fatias executáveis. A cada 0,4 segundos é executado uma requisição à máquina virtual para obter a próxima fatia a ser executada. Exemplo de execução de script:

```
Controller Classe1 {

    id enemyPos = 0; distance = 0;

    _onAlive() {

        enemyPos = getEnemyPos();

        distance = getPos();

        if (enemyPos > distance) {
            ahead(1);
            distance = enemyPos - getPos();
        } else {
            distance = getPos() - enemyPos;
        };

        if (distance == 1) {
            attack(1);
        }
    }
}
```

```
Controller Classe2 {

    _onAlive() {
        ahead(1);
    }

}
```

Figura 11. Tela de Carregamento da batalha.

Fonte: Autor, 2019

De acordo com os dois scripts visualizados na figura 11 (Classe 1 e Classe2), as fatias são executadas da seguinte maneira:

1º Fatia executável:

- **Classe1:** Movimenta para frente (**ahead**);
- **Classe2:** Movimenta para frente (**ahead**);

2º Fatia executável:

- **Classe1:** Movimenta para frente (**ahead**);
- **Classe2:** Movimenta para frente (**ahead**);

12º Fatia executável:

- **Classe1:** Executa ataque (**attack**);
- **Classe2:** Movimenta para frente (**ahead**);

Dessa maneira, independente da quantidade de verificações do script, cada fatia de executável garante a movimentação simultânea dos personagens.

6.4. Pós batalha

Após o término da batalha, o jogador poderá visualizar os detalhes obtidos da mesma. Com a experiência adquirida observando o desempenho de seu script, o jogador poderá aprimorar seu próprio código fonte, de modo que em uma próxima partida contra outros personagens se obtenha vantagem.

7. Resultados

Para execução das simulações em um ambiente real, foi elaborado um torneio supervisionado, onde os participantes deveriam criar seus próprios scripts de batalha. Como motivação, foi definido que os quatro primeiros colocados em que os personagens possuam armas distintas teriam seus scripts adicionados ao jogo como desafio.

Ao todo foram selecionados 8 competidores com idades entre 15 e 32 anos. Inicialmente foi delimitado um período de 30 minutos para que fosse possível conhecer o jogo, desde os menus até a execução de batalhas PVE. Durante todo o período houve apenas intervenção externa para auxílio na linguagem, todo o fluxo do jogo foi descoberto de maneira individual.

Após a conclusão do período de conhecimento da solução, deu-se início a competição. De modo que a mesma ficasse balanceada, foram sorteadas duas armas de cada tipo aos competidores. Para o desenvolvimento da lógica os competidores tiveram

40 minutos de desenvolvimento, e após uma vitória o jogador teria acesso ao fonte do oponente, tendo assim até 10 minutos para aprimorar seu script.

Após o desenvolvimento dos scripts, foi realizado um sorteio do chaveamento através da ferramenta **challonge.com**, que simplifica o sorteio e geração de chaves para torneios. As chaves ficaram da seguinte maneira:

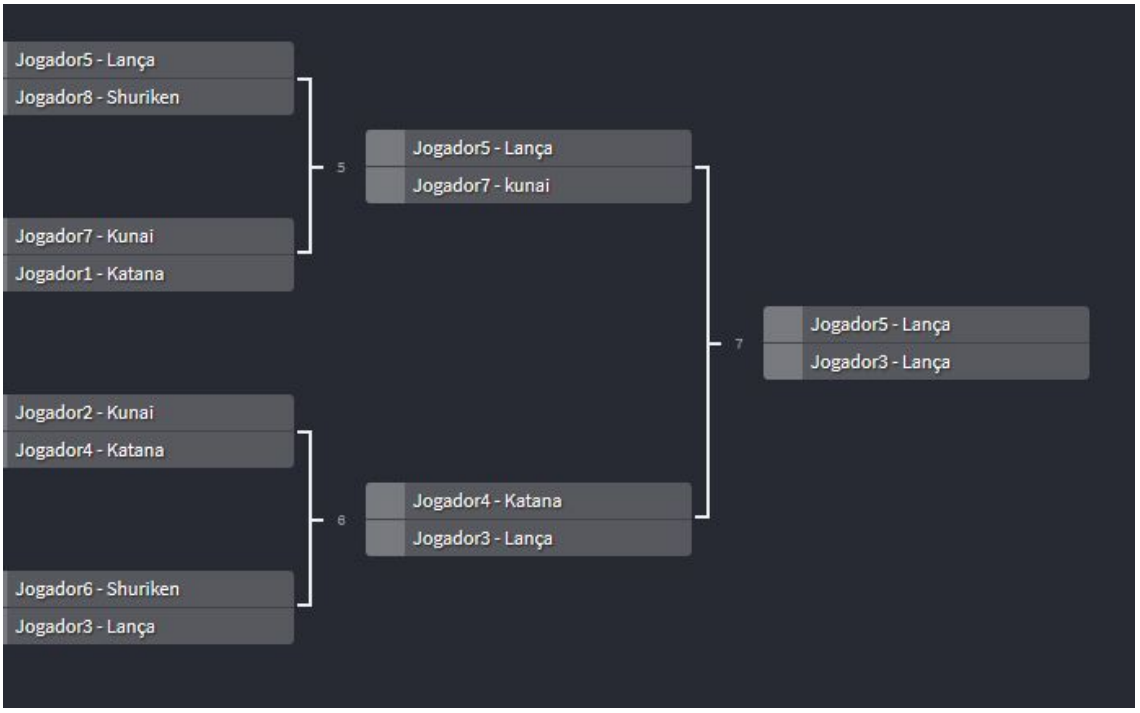

Figura 12.Chaveamento da competição.
Fonte: CHALLONGE, 2019

Ao fim do torneio foi possível constatar que, as armas que possuem maior dano (lança e katana), possuem maior facilidade perante as demais, principalmente pois as armas que atacam a distância, para obter vantagem, necessitam que o personagem não esteja muito próximo do inimigo, o que requer uma complementação mais detalhada da lógica empregada ao script.

Durante a execução do período inicial de adaptação ao jogo, foi possível notar a facilidade no acesso das batalhas, vale ressaltar que a quantidade de desafios PVE vencidos pelos competidores foram distintos. Foi possível constatar também que os jogadores passaram mais de 50% do tempo de execução do jogo no editor de script, consultando a lógica dos personagens inimigos do desafio e editando seus próprios scripts.

Durante o desenvolvimento dos scripts, ao primeiro contato, houve uma pequena dificuldade ao se adaptar aos comandos da linguagem, porém como a linguagem conta com uma quantidade limitada de elementos, portando somente os principais termos das

linguagens de programação, foi possível visualizar a adaptação na manipulação da linguagem. Após a análise dos scripts iniciais do jogo e execução de algumas batalhas PVE, onde os jogadores puderam visualizar a lógica do inimigo derrotado através da interface de desenvolvimento de script, foi possível identificar o aprendizado ao combinar lógicas de diversas fontes incrementando o script de ação do seu personagem.

8. Conclusão

O Jogo traz a possibilidade de descoberta e um novo aprendizado, focado em lógica de programação, contando com um domínio específico de linguagem limitado extraídos da união de elementos de outras duas linguagens. Desta maneira a linguagem contém somente os principais termos de estruturas de decisão, laços de repetição e operadores, com o intuito de facilitar o aprendizado e memorização dos comandos.

Conclui-se que o desenvolvimento de scripts, e execução de diversas batalhas, o jogador é motivado a melhorar a lógica empregada a fim de obter uma solução melhor para os próximos desafios. Com a execução da competição supervisionada, foi possível identificar a curva de aprendizado após a visualização da lógica de outros oponentes e unificação de diversos códigos. Desta maneira contribuindo para o conhecimento de lógica de programação de maneira divertida permitindo o aprimoramento pessoal do jogador. Possibilitando assim a capacitação na identificação e resolução de problemas.

Faz-se necessário ressaltar, que para fins evolutivos, o jogo em questão tendo a finalidade de atrair diversos públicos, e entendendo a demanda por interfaces cada vez mais atrativas e dinâmicas diante dos concorrentes desse nicho, necessita de uma elaboração gráfica mais consistente e interativa. Com relação a funcionalidades para trabalhos futuros tem-se como objetivo a implementação de novas armas, adição de caracteristicas distintas entre as classes (atributos como: força, velocidade, resistência) e disponibilidade do jogo como aplicativo para plataformas mobiles.

9. Referências

AHO, A. V.; LAM, M. S.;SETHI, R.; ULLMAN, J. D. Compiladores: Princípios, técnicas e ferramentas. Addison Wesley, 2008.

Posso, A. Internet Usage and Educational Outcomes Among 15-Year-Old Australian Students. Disponível em: https://ijoc.org/index.php/ijoc/article/view/5586/1742. Publicado em: 15/03/2006. Acesso: 2019.

Bello, Robson Scarassati. História e Videogames: como os jogos eletrônicos podem ser pensados por historiadores (Artigo). In: Café História – história feita com cliques. Disponível em: https://www.cafehistoria.com.br/historia-e-videogames. Publicado em: 13 nov. 2017. Acesso: 2019.

CARGO-BOT, disponível em: https://twolivesleft.com/CargoBot/. Acesso: 2019.

CHALLONGE, disponível em: https://challonge.com/pt_BR/. Acesso: 2019.

Gesser, Carlos Eduardo. Gals Gerador de Analisadores léxicos e sintáricos, disponível em: https://projetos.inf.ufsc.br/arquivos_projetos/projeto_353/Gals.pdf. Publicado em:02/2003. Acesso: 2019.

HOPCROFT, John E.; ULLMAN, Jeffrey D.; MOTWANI, Rajeev. Introdução à teoria de autômatos, linguagens e computação. Rio de Janeiro: Campus, 2003. 560 p. ISBN 8535210725.

JOHN PALFREY, Urs Gasser, Born Digital: Understanding the First Generation of Digital Natives, Basic Books, Inc., New York, NY, 2008.

LIGHT-BOT, disponível em: http://lightbot.com/. Acesso: 2019.

LOUDEN, Kenneth C. Compiladores: princípios e práticas. São Paulo: Pioneira Thomson, 2004. 569 p. ISBN 8522104220.

Prensky, M. (2001) "Digital Natives, Digital Immigrants Part 1", On the Horizon, Vol. 9 Issue: 5, pp.1-6.

MERNIK, Marjan & Heering, Jan & Sloane, Anthony. (2005). When and How to Develop Domain-Specific Languages. ACM Comput. Surv.. 37. 316-. 10.1145/1118890.1118892.

ROBOCODE BRASIL- Robocode 2019, disponível em: https://liag.ft.unicamp.br/robocode/robocode-2018/. Acesso: 2019.

Pesquisa Nacional por Amostra de Domicílios Contínua - PNAD Contínua, disponível em: https://www.ibge.gov.br/estatisticas/sociais/trabalho/17270-pnad-continua.html?=&t=o-que-e. Acesso: 2019.

PLAYTABLE - Croc uma melodia verde, disponível em: http://playtable.com.br/jogos/croac. Acesso: 2019.