SIMULADOR DE ASPIRADOR DE PÓ ROBÔ PARA PRÁTICAS DA DISCIPLINA DE ELETRÔNICA DIGITAL

Marco José de Sousa – marcojsousa@ufpa.br
Universidade Federal do Pará, Faculdade de Computação do Campus Tucuruí
Rua Itaipu 36 – Vila Permanente
CEP 68464-000 – Tucuruí – PA

Resumo: *Este artigo descreve o desenvolvimento e aplicação preliminar de um aplicativo web HTML5 adequado para o ensino da disciplina de Eletrônica Digital. O aplicativo oferece um ambiente experimental para teste de circuitos digitais sequenciais, especificamente para controlar um aspirador de pó autônomo simulado. O desafio é projetar um comportamento complexo para um modelo de autômato simples que possui poucas variáveis digitais de entrada e de saída. O simulador funciona sem necessidade de um servidor, o que possibilita o uso em locais não atendidos por redes de computadores ou Internet. Cada controlador de aspirador pode ser projetado por um estudante ou por um grupo de estudantes. Torneios podem ser realizados colocando cada autônomo participante para operar no mesmo ambiente simulado. A pontuação seria baseada na quantidade de resíduos coletados para um mesmo intervalo de tempo. Tais torneios podem substituir avaliações do curso. Outros jogos são propostos com base no simulador proposto.*

Palavras-chave: *Simulador, Eletrônica Digital, HTML5, Gamificação.*

1 INTRODUÇÃO

A problematização do ensino permite ao aluno entender as motivações da disciplina e como ela integra a realidade (SANTOS, 2012). Tradicionalmente os cursos de engenharia delegam a problematização para as práticas de laboratório, porém esse processo tem sido prejudicado para as disciplinas Eletrônica Digital (ED) (IDOETA, 2009). Não raro os cursos de Engenharia Elétrica e Engenharia da Computação estabelecem que ED será cursada já no primeiro semestre, quando os alunos ainda não dispõe de mínimos conhecimentos sobre eletricidade ou sobre os equipamentos mais comuns de laboratório. Para levar alunos calouros ao laboratório, o professor da disciplina precisa oferecer diversas aulas complementares, desde o acionamento de um simples Diodo Emissor de Luz até o funcionamento e uso das fontes de tensão, multímetros, osciloscópios, kits didáticos, etc. Algumas faculdades simplesmente eliminaram a parte prática dos cursos de ED, argumentando que diversas outras disciplinas do currículo mais adiante oferecerão oportunidade para práticas de laboratório.

Para compensar a falta dos laboratórios, os professores recorrem a exercícios e desafios que possam trazer a realidade para dentro da sala de aula. Softwares simuladores são utilizados, como o Logisim (LOGISIM, 2019), já que permitem a realização de experimentos complexos focados nos assuntos da disciplina. Embora esses softwares favoreçam a problematização, não fornecem um cenário ou um o ambiente apropriado para a realização de

jogos e competições, que têm o poder de aumentar o entusiasmo dos alunos, o que se reflete na eficiência do processo de aprendizado (MARKOPOULOS, 2015).

Considerando o caso quando a disciplina ED seja ministrada no primeiro semestre sem carga horária para prática, poucas opções restam ao professor para tornar a aula mais interessante aos alunos. Em função dessa problemática, este trabalho desenvolve um software para a simulação de um robô aspirador de pó. Embora menos emocionante do que simuladores de batalhas com vários participantes simultâneos, como feito no caso do Robocode (ROBOCODE, 2019), essa abordagem fornece uma dinâmica de competições para iniciantes, focando em simplicidade e eficiência. A ideia do simulador partiu de um exercício sugerido pelo autor, em que o aluno era convidado a elaborar um circuito digital sequencial para controlar um aspirador de pó robô, de modo que pudesse percorrer toda a área de um compartimento fechado retangular. O modelo era severamente simplificado para que pudesse ser resolvido como uma proposta de exercício desafio. Na ocasião as respostas dos alunos foram diversificadas, variando de absurdas até as mais engenhosas. A maior dificuldade foi na avaliação das respostas, pois demandava longas análises lógicas de circuitos complexos. Ficou evidente que a elaboração do problema diretamente em simulador dedicado seria a melhor opção para exercícios dessa natureza, simplificando a avaliação e permitindo experimentações por parte do aluno.

O simulador proposto coloca o robô em um compartimento retangular, repleto de sujeira e obstáculos contornáveis. O circuito digital sequencial a ser desenvolvido pelos alunos deve gerar sinais digitais para controlar o robô em função de apenas duas entradas: o sinal digital de um sensor de obstáculos e o sinal de sincronismo. Toda a dinâmica do jogo ocorre do ponto de vista superior como se a sala fosse um tabuleiro. Torneios podem ser realizadas em laboratório ou em sala de aula, apresentadas aos alunos por meio de projetor digital conectado ao computador do professor. Cada robô será colocado à prova em dinâmicas que substituam avaliações, com a pontuação proporcional ao número de "detritos" coletados, dentro de um intervalo de tempo fixo, e também em função da simplicidade do circuito de controle.

O simulador foi desenvolvido em HTML5 e *Javascript.* Os desenhos e animações são feitas em usando SVG. O programa executa em navegadores Web atuais e não exige um servidor na *Internet* para ser utilizado (HTML5, 2019),(JAVASCRIPT, 2019) e (SVG, 2019).

2 O ASPIRADOR

O personagem, a principal e única peça do jogo é o robô aspirador de pó. O aspirador é controlado por um *script*, cuja forma foi inspirada no Assembly (IBM, 2019). Nesta seção será definida a aparência do aspirador e como ele funciona do ponto de vista do jogador que deverá programar o *script*.

A aparência do autômato no tabuleiro é mostrada na Figura 1 (à esquerda), sendo um objeto circular com a frente diferenciada pela linha radial. O único sensor do aspirador é posicionado na interseção entre a linha e o círculo, direcionado ao longo da projeção da linha radial. Como mostrado na Figura 1 (à direita), o robô possui 50 cm de diâmetro e o sensor de obstáculo possui alcance de 10 cm, colocando uma variável lógica "SE" em nível "1" quando qualquer obstáculo entre no raio de alcance diretamente à frente do robô. Por simplicidade este sensor opera com um ângulo de abertura nulo, portanto sendo capaz de detectar apenas os obstáculos que interceptarem a projeção da linha radial, representada pela seta vermelha na Figura 1.

Figura 1 – Representação do robô aspirador no tabuleiro (à esquerda).
Dimensões do robô e posicionamento do sensor (à direita).

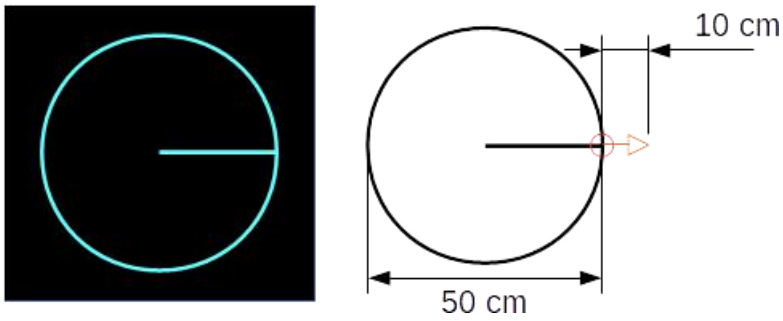


Tabela 1 - Relação das variáveis lógicas disponíveis para a operação do circuito de controle do robô.

Nome	Tipo	Comportamento/Funcionalidade
SE	Entrada	Nível 1 apenas quando obstáculos diretamente à frente a menos de 10 cm.
CLK	Entrada	Sinal de sincronismo (*clock*) tipicamente 10 Hz.
MF	Saída	Quando em nível alto, o robô se deslocará para frente.
MT	Saída	Quando em nível alto, o robô se deslocará para trás.
GH	Saída	Quando em nível alto, o robô deverá girar no sentido horário.
GA	Saída	Quando em nível alto, o robô deverá girar no sentido anti-horário.

Do ponto de vista do aluno de projeta o *script*, o sensor digital é representado pela variável lógica "SE", sendo "1" apenas quando houver obstáculo dentro do alcance de 10 cm. Outras variáveis disponíveis para controle: o sinal de sincronismo "CLK" e os sinais de controle de movimento "MF", "MT", "GH", "GA". Todas elas são relacionadas na Tabela 1. As variáveis lógicas de saída são mutuamente exclusivas de modo que, para realizar um determinado movimento, apenas uma delas deverá estar em nível "1", caso contrário o robô permanecerá parado. A velocidade padrão de deslocamento do robô foi fixada em 50 cm/s e a velocidade de rotação padrão foi fixada em 90 graus/s.

2.1 Definição do circuito de controle

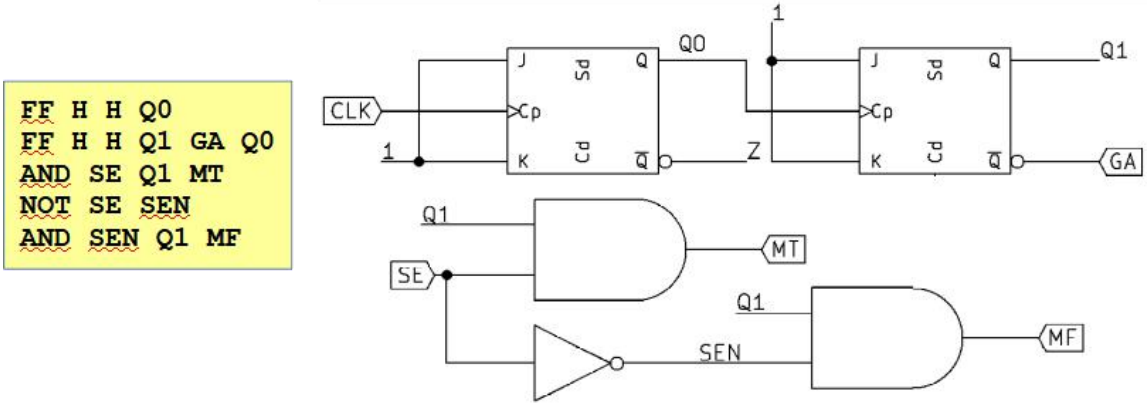
O circuito do robô deverá ser especificado na forma textual pelo aluno. Cada linha do texto representará um componente seguindo o formato geral: "nome_componente var1 var2 var3 … varN", onde "nome_componente" pode ser "not", "and", "or", "xor", "nand", "nor", "xnor" e "ff".

A interpretação do texto de definição do circuito não faz distinção entre caracteres maiúsculos e minúsculos. Todos os componentes representam portas lógicas intuitivamente identificadas exceto "FF", que representa um flip-flop do tipo JK com *Clock* sensível a borda de subida, entradas assíncronas de *Set* e *Reset* ativas em nível lógico "1". A ordem das variáveis "VAR1" até "VARN" definirá quais sinais ou variáveis serão conectados às diversas portas do componente. A Tabela 2 exibe a sintaxe simplificada para os comandos. Para o comando "FF", caso não sejam especificados, *Clock* será CLK, *Set* e *Reset* serão "0".

O código do circuito também pode usar valores constantes para definição de entradas e saídas nos comandos. As constantes suportadas são "0", "1", "H", "L" e "Z", que podem ser usadas em minúsculo ou maiúsculo. Como exemplo de *script* de definição de circuito, considere a Figura 2. O *script* de 5 linhas à esquerda representa o circuito de 5 componentes à

direita. Observe que "CLK" é omitido na primeira linha, pois esse é o sinal de sincronismo padrão quando uma variável não é indicada explicitamente. Finalmente a constante "Z" deve ser usada para "desconectar" ou omitir o respectivo pino.

Figura 2 - Exemplo de circuito factível de acordo com as definições adotadas.



Seria propício questionar a necessidade da adoção de uma linguagem de *script* única que, embora seja simples, representou trabalho adicional no desenvolvimento do simulador. Um interpretador foi criado para este propósito, mesmo sendo francamente possível a utilização, por exemplo, do formato JSON (JSON, 2019) para a definição do circuito. A justificativa está no público para o qual o simulador é destinado. Como são alunos calouros, não se pode presumir que já saibam os princípios de programação, sendo nesse caso muito mais simples a adoção de um *script* que possa ser aprendido através da leitura de um manual que se resume na Tabela 2.

Tabela 2 - Relação dos comandos para a definição textual do circuito de controle do robô.

Componente	Sintaxe
Inversor	NOT *Entrada Saída*
Porta E "AND"	AND *Entrada1 Entrada2* [… *EntradaN*] *Saída*
Porta OU "OR"	OR *Entrada1 Entrada2* [… *EntradaN*] *Saída*
Porta OU exclusivo "XOR"	XOR *Entrada1 Entrada2* [… *EntradaN*] *Saída*
Porta E invertida "NAND"	NAND *Entrada1 Entrada2* [… *EntradaN*] *Saída*
Porta OU invertida "NOR"	NOR *Entrada1 Entrada2* [… *EntradaN*] *Saída*
Porta OU exclusivo invertido "XNOR"	XNOR *Entrada1 Entrada2* [… *EntradaN*] *Saída*
Flip-flop JK	FF *J K Q* [*~Q* [*Clock* [*Set* [*Reset*]]]

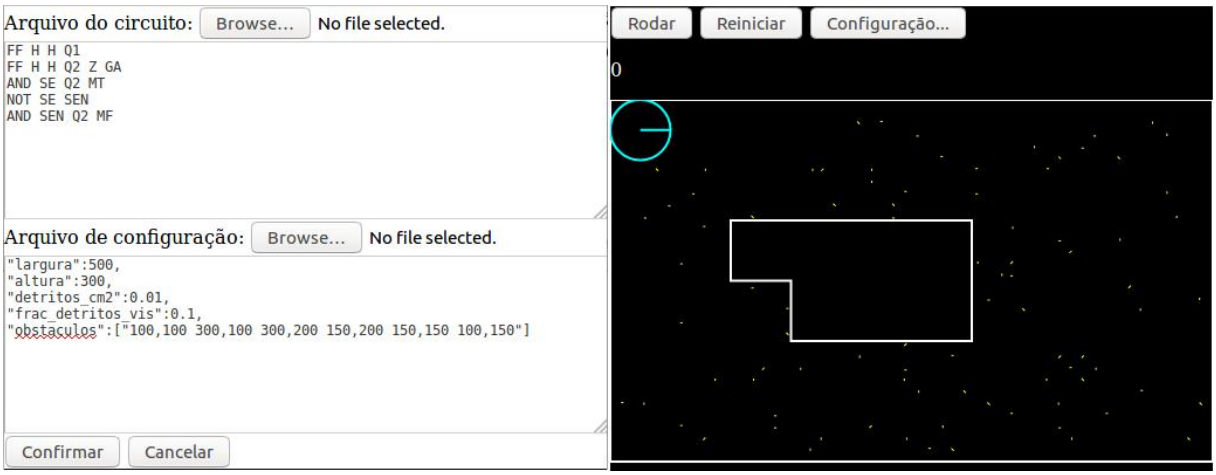
3 DEFINIÇÃO CENÁRIO

O cenário é composto pelas paredes do compartimento, obstáculos e os detritos a serem coletados pelo robô. A Figura 3 exibe, à esquerda, a tela para definição do *script* do circuito e também do JSON de configuração. A Figura3 à direita exibe o cenário resultante desenhado pronto para o início da simulação. No interior do retângulo externo que define o compartimento, os detritos são representados pelos pontos amarelos. Existe ainda um obstáculo formado por um polígono côncavo no meio do compartimento. A figura mostra também o robô aspirador no canto superior esquerdo pronto para o início da simulação. Todas as medidas são em centímetros e o desenho é feito caber na janela através de transformações geométricas definidas dinamicamente diretamente no código SVG.

As dimensões do compartimento, quantidade e formatos dos obstáculos, concentração de detritos, etc são especificados por um arquivo JSON que pode ser carregado na tela de

configuração, que é exibida através do botão "Configuração...". Se arquivos forem selecionados, os mesmos serão carregados diretamente para a página usando FileReader do HTML5, sem a necessidade de um servidor (FILEREADER,2019).

Figura 3 - Tela inicial com cenário desenhado, incluindo paredes, robô, detritos e um obstáculo.



4 A SIMULAÇÃO

São duas as partes principais da simulação: a simulação lógica e a física. A primeira é realizada através do processamento do circuito para atualização dos valores das variáveis lógicas, enquanto que a segunda realiza a animação da simulação respeitando as possibilidades físicas (mecânica, cinemática, etc).

4.1 Simulação lógica

O estado do circuito é definido pelo estado conjunto de todas as variáveis lógicas do sistema. A simulação lógica precisa duplicar a memória de modo a permitir que cada componente seja simulado sem interferir com o estado das variáveis de entrada de outros componentes. A simulação também precisa ser iterativa devido ao tempo de propagação de sinais ao longo de um circuito. O passo de simulação só é concluído depois que os estados lógicos das variáveis de saída se estabilizem. Caso não haja estabilização, é preciso interromper a simulação e reportar a instabilidade para que o aluno corrija o circuito.

Para entender o funcionamento da simulação lógica, considere que todos os componentes digitais do circuito possam ser abstraídos por funções lógicas $fp(M_2,M_3)$ para portas lógicas e $ff(M_1,M_2,M_3)$ para flip-flops, onde M_1, M_2 e M_3 representam três estados distintos, cada um deles compreendendo as mesmas variáveis lógicas do vetor M original do circuito. O motivo da diferenciação entre $fp(\ )$ e $ff(\ )$ em termos de números de argumentos se deve pelas entradas síncronas sensíveis à borda que $ff(\ )$ precisa considerar, que demandam a comparação entre os estados lógicos de tempos diferentes representados pelos vetores M_1 e M_2. Portanto, $fp(M_2,M_3)$ considera todas as variáveis lógicas de M_2 como entrada, aplica as expressões lógicas do componente em particular e gera um novo estado de saída M_3. A função $ff(M_1,M_2,M_3)$ faz o mesmo usando como entrada M_2 e saída M_3, porém considerando diferenças entre M_1 e M_2 (para detectar bordas de subida). O algoritmo da simulação lógica é mostrado na Figura 4. É importante notar na linha 2 a simulação física é realizada através da função SimulFis, que retorna uma cópia do vetor M com as variáveis lógicas "SE" e "CLK" atualizadas.

Figura 4 - Algoritmo para simulação lógica.

```
1.  M1 = M;
2.  M2 = SimulFis(M);
3.  Enquanto M1 != M2 :
4.      Para cada porta lógica:
5.          fp(M2,M3);
6.      Para cada flip-flop:
7.          ff(M1,M2,M3);
8.      M1=M2;
9.      M2=M3;
10. M = M3
```

4.2 Simulação física

A simulação física simula o movimento do robô aspirador dentro do compartimento, respeitando os obstáculos e as paredes. Os movimentos são determinados pelas variáveis lógicas de saída "MF", "MT", "GH" e "GA", porém a simulação física também deve atualizar recorrentemente as variáveis "CLK" e "SE" em função do tempo de simulação e da ocorrência de obstáculos próximos da visada do sensor de obstáculos.

Para criar a animação da movimentação do robô e o sincronismo necessário para gerar "CLK", foi utilizado um temporizador o intervalo t_F em milissegundos lido do JSON de configuração a partir da variável "t_F". O valor padrão para t_F é de 10ms. O cada chamada do temporizador, é realizado o algoritmo da Figura 4, isto é, a simulação lógica, a qual executa a simulação física por meio da função SimulFis( ). É esta função que de fato atualiza a animação e que atualiza as variáveis "SE" e "CLK". A frequência com que SimulFis( ) é chamada determina a taxa de *frames* por segundo (fps) da animação, que é de cerca de 100fps para t_F = 10ms.

Um outro parâmetro importante para a função SimulFis( ) é a variável "n_CLK", que também é lida a partir do JSON de configuração. Essa variável controla o período do sinal de "CLK". A cada chamada da função SimulFis( ) uma variável inteira k_T é incrementada. Quando o resto da divisão inteira de k_T pela variável "n_CLK" resulta zero, o estado de "CLK" é alterado de "0" para "1" ou de "1" para "0". Portanto, para o valor mínimo de "n_CLK" de 1, a frequência de "CLK" seria 50 Hz (máxima). O valor padrão de "n_CLK" é 5, o que implica uma frequência de 10 Hz.

O ajuste da taxa de frames e do período de "CLK" afeta a velocidade da animação, pois a velocidade do robô é determinada pelo deslocamento por *frame* em centímetros, enquanto a velocidade de rotação é medida em graus por *frame*. Estas variáveis são lidas do JSON de configuração a partir das ariáveis "robo_vel" e "robo_giro", cujos valores padrão são respectivamente de 0,5 (centímetros) e 0,9 (graus). Portanto as velocidades padrão do robô serão de 50 cm/s e 90 graus/s.

Uma das primeiras tarefas da função SimulFis( ) é atualizar o vetor do sensor **V**, que é definido em função do ângulo atual de rotação a do robô: **V** = (cos(a) , sen(a)). Considera-se que o valor a usado na definição de V esteja atualizado em função das variáveis lógicas "GH" e "GA". Em seguida é feita a atualização da posição **R** do robô:

$$\mathbf{R} = \begin{cases} \mathbf{R} + \mathbf{V}dr, \mathrm{MF} = 1 \\ \mathbf{R} - \mathbf{V}dr, \mathrm{MT} = 1 \end{cases} \quad (1)$$

Onde *dr* representa o parâmetro "robo_vel".

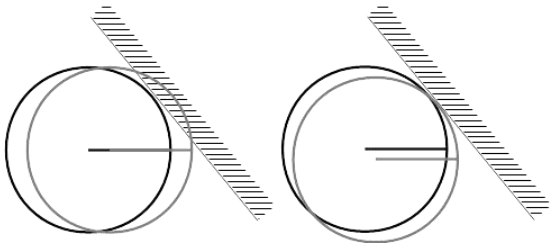
Após o deslocamento do robô, o teste de colisão é realizado com todos os segmentos de para cada polígono do cenário. Caso a distância entre o segmento e o ponto **R** seja inferior ao raio *r* do robô, então uma colisão ocorreu com este segmento e **R** precisará ser corrigido.

Considere que o segmento atingido seja definido pelos pontos **A** e **B**. Calcula-se o vetor unitário **U** ortogonal ao segmento **AB**. A distância d_{AB} entre o centro do robô e o segmento **AB** pode ser obtida através do produto escalar entre (**R**–**A**) e **U**. A correção de **R** pode ser realizada em função de *r*, *dAB* e de **U**:

$$\mathbf{R} = \mathbf{R} + \left(r - |d_{AB}|\right)\left(|d_{AB}|/d_{AB}\right)\mathbf{U} \qquad (2)$$

Observa-se que os obstáculos agem desviando (resvalando) o robô quando uma colisão ocorre. Esse comportamento tem a vantagem de evitar trancar os movimentos do robô, situação em que apenas circuitos de controle mais complexos ou um modelo de robô aspirador com muito mais sensores à disposição teriam capacidade de remediar. A Figura 5 ilustra, através da sombra em cinza, o movimento e a correção subsequente do robô.

Figura 5 - Movimento sem correção (à esquerda) e movimento corrigido (à direita).



A atualização de "SE" consiste no cálculo da interseção do vetor do sentido do sensor de obstáculo com todos os segmentos de retas do cenário, incluindo obstáculos e paredes. Caso a distância euclidiana do ponto de interseção seja inferior a 10 cm, "SE" é definido como "1", caso contrário "SE" será "0".

Finalmente, após consolidado o movimento do robô e atualizadas as variáveis lógicas, deve-se contar e apagar os detritos que estejam a uma distância inferior a *r* do centro do robô **R**, além de atualizar o desenho do robô na tela. A Figura 6 apresenta o algoritmo para a "SimulFis" de acordo com as etapas discutidas nesta seção. Considere que "SE", "CLK", "MF", "MT", "GA" e "GH" sejam variáveis do vetor M_2 retornado por SimulFis( ) no algoritmo da Figura 4.

Figura 6 - Algoritmo para a função SimulFis.

```
1.  Atualiza CLK;
2.  Se GA=1 ou (exclusivo) GH=1 e MT=MF=0 :
3.  |   Atualiza o ângulo a do robô em função de robo_giro;
4.  Se MT=1 ou (exclusivo) MF=1 e GA=GH=0 :
5.  |   Atualiza a posição R do robô usando a Eq. (1);
6.  |   Corrige R usando Eq. (2);
7.  Calcula a menor distância d entre o sensor SE e os obstáculos;
8.  Se não houverem obstáculos ou d > 10cm:
9.  |   SE = 0;
10. Senão :
11. |   SE = 1;
12. Para cada detrito em P = (x,y):
13. |   Se ||P – R|| <= r :
14. |   |   Remove detrito;
15. |   |   Incrementa pontuação;
16. Atualiza tempo de simulação e atualiza o desenho do robô;
```

5 APLICAÇÃO

O programa está sendo aplicado em uma turma de Eletrônica Digital II do curso de Engenharia Elétrica do campus de atuação do autor. Estão previstas 6 avaliações baseadas em exercícios-desafio, cada desafio valendo 5 pontos.

O software na sua versão preliminar foi apresentado aos alunos como proposta de exercício-desafio na quarta avaliação, para que desenvolvessem um circuito de controle do aspirador de pó robô capaz de remover pelo menos 50% da sujeira do cenário da Figura 3 em 30 minutos (tempo em perspectiva do robô). A recepção foi excelente, visto que a participação dos alunos cresceu a ponto de haver interesse por parte deles na programação do software, havendo inclusive várias sugestões para melhorias do simulador e até mesmo a correção um pequeno erro de programação por parte de um dos alunos.

Infelizmente ainda existem alguns erros de programação que impediram o uso correto do simulador por parte de alguns alunos mais avançados, que escreveram *scripts* mais longos. Foi observado que instabilidades podem ocorrer em circuitos com mais de 50 linhas. Para esses casos a avaliação das respostas dos alunos precisou ser feita de forma manual com auxílio de simuladores como o Logisim e o CricuitVerse (CIRCUITVERSE, 2019).

Apesar dos problemas, o impacto nas notas foi interessante. Até o desafio 2 as questões, apesar do esforço na problematização, foram relativamente simples e não muito diferentes dos exemplos resolvidos em sala de aula. Para o desafio 3 houve a primeira problematização, com uma questão que perdia aos alunos que formulassem um circuito sequencial para acender sequencialmente um conjunto de LEDs arranjados em linha, para que o efeito luminoso fosse similar àquele dos ciclopes robóticos de uma conhecida série de televisão de ficção científica. A receptividade foi boa mas as notas ainda foram insuficientes.

Porém, a partir da aplicação do simulador, o engajamento melhorou e isso se refletiu na média das notas da turma, como mostrado na Tabela 3.

Tabela 3 - Pontuação média dos exercícios desafios utilizados para avaliação.

Desafio	Nota média (Máximo = 5 pontos)
1	2,2
2	0,8
3	1,4
4	4,2

Para o desafio 5 será solicitado ao aluno que escreva ou aprimore um circuito para recolher pelo menos 50% dos detritos em 30 minutos, porém para um cenário mais complexo, formado por obstáculos que simulem um compartimento equipado com um conjunto de mesa com quatro cadeiras.

Para o último desafio será realizado um torneio de equipes com no máximo 4 alunos por equipe. Os compartimentos serão de 5 m por 10 m (50 m^2), com e sem obstáculos. Para cada linha de *script* serão subtraídos o equivalente a 1 m^2 de detritos da contagem total coletada pela equipe, de modo que circuitos mais simples serão beneficiados (pontuações negativas serão possíveis).

A nota da avaliação será feita da seguinte forma: as equipes serão ranqueadas em função da pontuação. Todas as equipes que participarem e coletarem pelo menos o equivalente a 1 m2 de detritos em cada cenário ganham 2,5 pontos. Os outros 2,5 pontos serão calculados em função do ranque. A equipe primeira colocada (ranque 1) receberá 2,5 pontos. A equipe em

último lugar no ranque (ranque n) recebe zero pontos. Portanto a quantidade de pontos p_k para a equipe de ranque r_k será calculada como:

$$p_k = 2{,}5\left(\frac{n - r_k}{n - 1}\right) \tag{3}$$

6 CONSIDERAÇÕES FINAIS

Os resultados preliminares indicam que o simuladores e dinâmicas competitivas tem o potencial de melhorar o interesse e a aplicação dos alunos. Entretanto, o inesperado interesse por parte dos alunos na programação do próprio simulador abre espaço para atividades multidisciplinares. Por exemplo, o desenvolvimento e aplicação do simulador poderia ser realizado como prática avaliativa de disciplinas como "Programação" dos currículos dos cursos de Engenharia da Computação e Engenharia Elétrica. Certamente novas versões deste simulador serão desenvolvidas com o esforço conjunto de alunos, sejam eles em iniciação científica ou em atividades curriculares avaliativas.

O simulador atual foi desenvolvido especificamente para a disciplina ED, porém há grande potencial para atender, no futuro, outras disciplinas do currículo de Engenharia Elétrica ou Engenharia da Computação. Por exemplo, o modelo do robô aspirador poderia ser mais realista, incluindo mais sensores e modelos de motores elétricos de tração, etc. O próprio *script* de definição do circuito poderia considerar outros tipos de componentes, como multiplexadores, memórias, contadores, etc. Suporte a outras linguagens permitiria o uso do simulador em disciplinas como "Arquitetura e Organização de Computadores", "Microntroladores e Microprocessadores" ou "Sistemas Digitais".

Uma versão melhorada deste software poderia mesmo suportar a definição do circuito por meio de código VHDL (*Very High speed integrated circuit hardware Description Language*). Nesse caso o software poderia ser alterado para o modelo cliente-servidor, onde um simulador VHDL, como o GHDL (GHDL, 2019), poderia ser executado no servidor para controlar o robô. Essa versão poderia ser aproveita em disciplinas como "Lógica Programável", "Sistemas Digitais", "Microeletrônica", etc.

REFERÊNCIAS

Livros:

IODETA, Ivan V.; CAPUANO, Francisco G. **Elementos de Eletrônica Digital.** São Paulo: Editora Érica. 2009.

Artigos de periódicos:

MARKOPOULOS, Angelos. P. *et al.* Gamification in engineering education and professional training. **International Journal of Mechanical Engineering Education**, 43(2), 118–131. 2015.

Trabalhos em eventos

SANTOS, Amélia M. *et al.* O Ensino da Engenharia por Meio da Metodologia da Problematização. In: XL Congresso Brasileiro de Educação em Engenharia 2012, Belém. **Anais.** Belém, PA, 2012.

Internet:

FILEREADER. **FileReader – Referência da API Web | MDN.** Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/API/FileReader. Acesso em: 10 mar. 2019.

HTML5. **HTML5 – HTML | MDN.** Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/HTML/HTML5 . Acesso em: 09 mar. 2019.

IBM. **Assembler Language.** Disponível em: https://www.ibm.com/support/knowledgecenter/SSLTBW_2.1.0/com.ibm.zos.v2r1.asma400/asmr102112.htm. Acesso em: 10 abr. 2019.

JAVASCRIPT. **JavaScript Tutorial.** Disponível em: https://www.w3schools.com/js/. Acesso em: 09 mar. 2019.

JSON. **Introducing JSON.** Disponível em: https://www.json.org/. Acesso em: 09 mar. 2019.

SVG. **SVG Tutorial**. Disponível em: https://www.w3schools.com/graphics/svg_intro.asp. Acesso em: 09 mar. 2019.

Dados e softwares abertos:

AVES do Amapá: banco de dados. Disponível em: http://www.bdt.org/bdt/avifauna/aves. Acesso em: 25 nov. 1998.

CIRCUITVERSE. CircuitVerse - Online Digital Logic Circuit Simulator. Disponível em: https://circuitverse.org/ . Acesso em: 10 mar. 2019.

LOGISIM. **Logisim [Português]**. Disponível em: http://www.cburch.com/logisim/pt/index.html. Acesso em: 09 mar. 2019.

ROBOCODE. **Robocode Home.** Disponível em: https://robocode.sourceforge.io/. Acesso em: 09 de mar. 2019.

ROBOT VACUUM CLEANER SIMULATOR FOR DIGITAL ELECTRONICS DISCIPLINE PRACTICES

Abstract: *This article describes the development and operation of an HTML5 web application suitable for teaching the discipline of Digital Electronics. The application provides an experimental environment for testing sequential digital circuits, specifically to control a simulated autonomous vacuum cleaner. The challenge is to design a complex behavior for a simple automaton model that has few input and output digital variables. The simulator works without the need of a server, which allows the use in places not served by computer networks or the Internet. Each vacuum cleaner controller can be designed by a student or a group of students. Tournaments can be performed by placing each participant automaton to operate in the same simulated environment. The score would be based on the amount of waste collected for the same time interval. Such tournaments can replace course evaluations. Other games are proposed based on the presented simulator.*

Key-words: *Simulator, Digital Electronics, HTML5, Gamification.*