Princípios de Programação em Sala de Aula

ALUNO: ____________________________________________________

CURSO: __________________________________________________

TURMA: __________________________________________________
<table>
<thead>
<tr>
<th>Aula 1</th>
<th>Princípios de Programação</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aula 2 e 3</td>
<td>Introdução ao Scratch e sua estrutura</td>
<td>16</td>
</tr>
<tr>
<td>Aula 4</td>
<td>Scratch como simulador – parte 1</td>
<td>27</td>
</tr>
<tr>
<td>Aula 5 e 6</td>
<td>Scratch como simulador – parte 2</td>
<td>32</td>
</tr>
</tbody>
</table>
## SUMÁRIO

### AULA 1 – LÓGICA DE PROGRAMAÇÃO

1. Introdução .................................................................................................................. 1
2. Conceito de Lógica de Programação: Algoritmo ...................................................... 4
3. Representando um algoritmo ..................................................................................... 5
4. Aplicando a taxonomia de Bloom ............................................................................ 7
5. Hora de exercitar ......................................................................................................... 9
6. Referências .................................................................................................................. 12

### AULA 2 E 3 – DIFICULDADE DE PROGRAMAÇÃO, SCRATCH É UMA SOLUÇÃO: INTRODUÇÃO E ESTRUTURA

7. A dificuldade do aluno em lógica de programação .................................................. 13
8. Conhecendo o Scratch .............................................................................................. 17
9. Estrutura do Scratch .................................................................................................. 18
10. Agora é com você ...................................................................................................... 22
11. Referências ............................................................................................................... 23

### AULA 4 – SCRATCH COMO SIMULADOR – PARTE 1

12. Entendendo um pouco sobre simulações ................................................................. 24
13. Agregando conhecimento: outra plataforma de simulação ....................................... 25
14. Forçando a criatividade ............................................................................................. 27
15. Referências ............................................................................................................... 28

### AULA 5 e 6 – SCRATCH COMO SIMULADOR – PARTE 2

16. Ambiente de programação do Scratch .................................................................... 29
16.1 O Stage ..................................................................................................................... 29
16.2 Lista de Sprites ......................................................................................................... 30
16.3 A aba Blocks ............................................................................................................ 31
16.4 A área de Scripts ..................................................................................................... 33
16.5 A barra de ferramentas .......................................................................................... 33
17. Blocos no Scratch ..................................................................................................... 34
18. Iniciando a simulação ................................................................................................. 36
19. Referências ............................................................................................................... 37
AULA 1 – LÓGICA DE PROGRAMAÇÃO

1. INTRODUÇÃO

Muito se usa a palavra lógica, geralmente relacionada à coerência e à racionalidade. Uma das preocupações da lógica é determinar quais operações são válidas e quais não são, fazendo análise das formas e leis do pensamento, com isso pode-se relacionar a lógica com a ‘correção do pensamento’. Podemos ainda dizer que a lógica é a ‘arte de bem pensar’, que é a ciência das formas de pensamento. Visto que a forma mais complexa do pensamento é o raciocínio, a lógica estuda a ‘correção do raciocínio’. Pode-se ainda dizer que a lógica te em vista a ‘ordem da razão’. Isso dá a entender que a nossa razão pode funcionar desordenadamente; por isso, a lógica estuda e ensina a colocar ‘ordem no pensamento’.

O raciocínio é algo abstrato, intangível. Os seres humanos têm a capacidade de expressá-lo através da palavra falada ou escrita, que por sua vez se baseia em um determinado idioma, que segue uma série de padrões (gramática). Um mesmo raciocínio pode ser expresso em qualquer um dos inúmeros idiomas existentes, mas continuará representando o mesmo raciocínio, usando apenas outra convenção. Algo similar ocorre com a Lógica de Programação, que pode ser concebida pela mente treinada e pode ser representada em qualquer uma das inúmeras linguagens de programação existentes. Essas, por sua vez, são muito atreladas a uma grande diversidade de detalhes computacionais, que pouco têm a ver com o raciocínio original. Para escapar dessa torre de Babel e, ao mesmo tempo, representar mais fielmente o raciocínio da Lógica da Programação, utilizamos os Algoritmos (veremos na seção a seguir).
Sempre que pensamos, a lógica ou a ilógica necessariamente nos acompanham. Quando falamos ou escrevemos, estamos expressando nosso pensamento, logo, precisamos usar a lógica nessas atividades. Podemos perceber a importância da lógica em nossa vida, não só na teoria, como na prática, já que, quando queremos pensar, falar, escrever ou agir corretamente, precisamos colocar ‘ordem no pensamento’, isto é, utilizar lógica.

Aprender programação é uma atividade importante, principalmente por desenvolver um conjunto de habilidades úteis em diversas áreas do conhecimento. Todavia, seu propósito geral não deve estar apenas focado na produção final de código. O cerne desta atividade envolve, sobretudo, desenvolver nos alunos competências relacionadas à abstração, raciocínio lógico, decomposição de problemas e criação de estratégias para a produção de soluções. Esses conceitos estão ligados diretamente aos aspectos cognitivos e psicológicos que precisam ser despertados, fortalecidos e praticados.
O ensino de programação no nível médio vem se mostrando uma iniciativa promissora, por se entender que esse conjunto de habilidades desenvolvidas pode ser empregado no cotidiano, não estando restrito apenas aos profissionais de computação. Uma das principais dificuldades para apoiar essa iniciativa encontra-se na escolha de uma linguagem adequada a esse público, que desperte a atenção e facilite a aprendizagem. Os trabalhos de [Aureliano e Tedesco, 2012], [Pereira et al, 2012] e [Scaico et al, 2012] apontam que o Scratch é uma linguagem adequada para a realidade das escolas brasileiras por utilizar um conceito inovador de desenvolvimento orientado o design, que privilegia a computação criativa, permite a manipulação de mídias, para a criação de animações e jogos, e está disponível gratuitamente no idioma português.
2. CONCEITO DE LÓGICA DE PROGRAMAÇÃO: ALGORITMO

O objetivo principal do estudo da Lógica de Programação é a construção de algoritmos coerentes e válidos. Mas o que é um algoritmo?

Segundo .... “... um algoritmo é a descrição de um conjunto de comandos que, obedecidos, resultam em uma sucessão finita de ações.”

A lógica de programação é importante para aquelas pessoas que desejam desenvolver programas, aplicativos e sistemas. Ela permite que o programador defina uma sequência lógica a seguir para o desenvolvimento de uma aplicação em uma determinada linguagem; permite que uma pessoa organize os pensamentos que ela tem, para transformá-lo em um objetivo, alcançado através da programação. A linguagem de programação é como uma língua normal, um grupo de palavras com significados. No caso da programação, a maioria das linguagens é escrita em Inglês. Estas linguagens fazem o computador assimilar cada comando e função de um algoritmo, depois executar cada função. O grande problema para muitos é o que “dizer” para o computador fazer o que é desejado.

Um algoritmo é uma sequência de instruções finita e ordenada de forma lógica para a resolução de uma determinada tarefa ou problema. São exemplos de algoritmos instruções de montagem, receitas, manuais de uso, etc. Um algoritmo não é a solução do problema, pois, se assim fosse, cada problema teria um único algoritmo; um algoritmo é um caminho para a solução de um problema. Em geral, existem muitos (senão infinitos) caminhos que levam a uma solução satisfatória.

Um algoritmo não computacional é um algoritmo cuja sequência de passos, a princípio, não pode ser executada por um computador. Abaixo é apresentado um algoritmo não computacional cujo objetivo é usar um telefone público. Provavelmente você “executou” o algoritmo deste exemplo diversas vezes. O termo algoritmo está muito ligado à Ciência da Computação, mas, na realidade, ele pode ser aplicado a qualquer problema cuja solução possa ser decomposta em um grupo de instruções.
Para que o algoritmo possa ser executado por uma máquina é importante que as instruções sejam corretas e sem ambiguidades. Aprender as palavras e regras que fazem parte dessa sintaxe é fundamental; no entanto, não é o maior objetivo deste curso. O que realmente exigirá um grande esforço por parte do estudante é aprender a resolver problemas utilizando a linguagem. Para isso, há somente um caminho: resolver muitos problemas. O processo é semelhante ao de tornar-se competente em um jogo qualquer: aprender as regras do jogo (a sintaxe) é só o primeiro passo, tornar-se um bom jogador (programador) exige tempo, muito exercício e dedicação.

3. REPRESENTANDO UM ALGORITMO

Convém enfatizar mais uma vez que algoritmo é uma linha de raciocínio, que pode ser descrito de diversas maneiras, de forma gráfica ou textual. Os algoritmos representados até o momento (Figura 3) estavam em forma textual, usando apenas o português coloquial.

As formas gráficas são mais puras por serem mais fiéis ao raciocínio original, substituindo um grande número de palavras por convenções de desenhos. Para fins de ilustração mostraremos como ficaria o algoritmo da figura 3 representado graficamente em um fluxograma tradicional.

Essa técnica possui suas vantagens e desvantagens; no entanto, podemos perceber que ela permite um nível grande de clareza quanto ao fluxo de execução. Contudo, deve ser menos fácil entender
essa representação do que em sua forma textual, como mostra a figura 3. Isso ocorre porque é necessário conhecer as convenções gráficas da técnica, que apesar de simples não são naturais, pois estamos mais condicionados a nos expressar por palavras.

Outra desvantagem é que sempre se mostra mais trabalhoso fazer um desenho do que escrever um texto, mesmo considerando o auxílio de réguas e moldes. A problemática é ainda maior quando é necessário fazer alguma alteração ou correção no desenho. Esses fatores podem desencorajar o uso de representações gráficas e, algumas vezes, erroneamente, a própria construção de algoritmos.

**Figura 3.1 – Estrutura de um algoritmo em forma de Fluxograma**
4. APLICANDO A TAXONOMIA DE BLOOM

A taxonomia de Bloom consiste em um instrumento cujo objetivo é apoiar a identificação de uma estrutura hierárquica com objetivos educacionais bem definidos, do mais simples para o mais complexo, abrangendo seis principais categorias de domínio cognitivo: conhecimento, compreensão, aplicação, análise, síntese e avaliação. Aliada a definição de cada um desses níveis do domínio cognitivo há um conjunto de verbos associados sugeridos, os quais auxiliam na elaboração de questões de avaliação para serem aplicados em cada uma das categorias da taxonomia. Assim, para que o aluno adquira as habilidades do próximo nível, é necessário que ele tenha adquirido habilidades do nível anterior.

Quadro 1 – Níveis da Taxonomia de Bloom

<table>
<thead>
<tr>
<th>Nível</th>
<th>Definição</th>
<th>Amostra de Verbos</th>
</tr>
</thead>
<tbody>
<tr>
<td>3. Aplicação</td>
<td>O aluno seleciona, transfere e usa dados e princípios para completar um problema ou tarefa com um mínimo de supervisão.</td>
<td>Use, Compute, Resolva, Demonstre, Aplique, Construa.</td>
</tr>
<tr>
<td>4. Análise</td>
<td>O aluno distingue, classifica e relaciona pressupostos, hipóteses, evidências ou estruturas de uma declaração ou questão.</td>
<td>Analise, Categorize, Compare, Contrastar e Separe.</td>
</tr>
<tr>
<td>5. Síntese</td>
<td>O aluno cria, integra e combina ideias em um produto, plano ou proposta, novos para ele.</td>
<td>Crie, Planeje, Elabore hipótese(s), invente e Desenvolva.</td>
</tr>
</tbody>
</table>

Os conteúdos de introdução a algoritmos, linguagem de programação, apresentação do ambiente Scratch e plano cartesiano (conceito que orienta a movimentação dos elementos no Scratch) foram ministrados norteados pelos níveis conhecimento e compreensão da taxonomia.

- **Nível conhecimento**: estabelece que lembrar consiste em reconhecer e recordar informações já adquiridas e guardadas na memória de longa duração.
• **Nível compreensão:** afirma que entender consiste na capacidade de fazer sua própria interpretação do material educacional. Para alcançar as metas propostas pela taxonomia, provoca-se o aluno a recordar de exemplos do seu cotidiano, como ir à escola, trocar uma lámpada, fazer uma ligação de celular.

• **Nível aplicação:** refere-se a usar procedimentos e conteúdos aprendidos em uma situação familiar ou nova. Essa atividade analisa a capacidade dos alunos de realizarem abstrações e uso de laço de repetição.

• **Nível análise:** consiste em dividir o conhecimento em partes e pensar como essas partes se relacionam com a estrutura geral.

• **Nível síntese:** refere-se à capacidade de planejamento e criação a partir do conteúdo ensinado. Os alunos devem ser capazes de reunir os elementos de programação ensinados e desenvolver o programa planejado.

• **Nível avaliação:** consiste na análise crítica do conhecimento adquirido pelo aluno; faz-se uma avaliação do tipo somativa a fim de conscientizar-se sobre o que o aluno absorveu.
5. HORA DE EXERCITAR...

1) Defina o que é algoritmo com suas palavras. Exemplifique.

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2) Explique o que é linguagem de programação com suas palavras.

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3) Para você, qual a ligação entre a Taxonomia de Bloom e o algoritmo?

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4) Defina o que é uma função com suas palavras. Exemplifique.

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4) Três senhoras – dona Branca, dona Rosa e dona Violeta – passeavam pelo parque quando a dona Rosa Disse: Não é curioso que estejamos usando vestidos de cores branca, rosa e violeta, embora nenhuma de nós esteja usando um vestido de cor igual ao seu próprio nome? Uma simples coincidência, respondeu a senhora com o vestido violeta. Qual a cor do vestido de cada senhora?

5) Analise o seguinte algoritmo e descreva o que ele faz.

```
1. Algoritmo "PrimeiroAlgoritmo"
2. var
3. NOTA1, NOTA2, NOTA3, NOTA4, MEDIA: real
4. NOME: caracter
5. inicio
6.   leia (NOME)
7.   leia (NOTA1)
8.   leia (NOTA2)
9.   leia (NOTA3)
10.  leia (NOTA4)
11.  MEDIA <- (NOTA1 + NOTA2 + NOTA3 + NOTA4) / 4;
12.  escreva (NOME, " obteve ", MEDIA)
13. finalgoritmo
```
6) Elabore um algoritmo que mova três discos de uma Torre e Hanói, que consiste em três hastes (a – b – c), uma das quais serve de suporte para três discos de tamanhos diferentes (1 – 2 – 3), os menores sobre os maiores. Pode-se mover um disco de cada vez para qualquer haste, contanto que nunca seja colocado um disco maior sobre um menor. O objetivo é transferir os três discos para outra haste.
6. REFERÊNCIAS

Apostila de lógica de programação.


7. A DIFICULDADE DO ALUNO EM LÓGICA DE PROGRAMAÇÃO

De acordo com pesquisas realizadas com alunos e ex-alunos dos cursos da área de informática ficou evidente que disciplinas ligadas à programação possuem alto índice de dificuldade e de reprovação. A lógica em geral trata-se da correção do pensamento e a arte de pensar corretamente. Visto que a forma mais complexa do pensamento é o raciocínio, a lógica estuda ou tem em vista a “correção do raciocínio”, como falado em aula anteriores. A falta de organização do pensamento para resolução de um determinado algoritmo leva muitos alunos a terem dificuldade e reprovarem em disciplinas que contenham lógica de programação e em consequência desistirem do curso. A lógica da programação é a técnica do uso correto do raciocínio para atingir um determinado objetivo, que resolve de forma ordenada a solução do problema, extremamente necessária para o desenvolvimento de programas, permitindo a definição de uma sequência lógica, ou seja, de passos a serem executados até atingir um determinado objetivo ou solução de um problema. A lógica de programação é parte do raciocínio lógico empregado no desenvolvimento de programas de computador, fazendo uso ordenado de elemento básico suportado por um dado estilo de programação.
Uma boa lógica de programação é desenvolvida a partir de um conjunto de elementos entre eles: Organização, Criatividade, Perseverança, Padronização e Otimização, desta forma, acredita-se que o uso do **Scratch** e o desenvolvimento de jogos que utilizam os mesmos conceitos e estruturas utilizadas para o desenvolvimento de algoritmos comerciais são de grande valor em séries iniciais dos cursos de tecnologia da informação, podendo desmistificar os conceitos e trazer maior engajamento dos alunos.

A literatura apresenta diversas justificações para a dificuldade inerente ao aprender a programar. Dijkstra (1989) argumenta que este tipo de aprendizagem é um processo lento e gradual. Almeida et al. (2002) referem que se observa, neste tipo de matérias, falta de interesse por parte dos alunos. Referem adicionalmente que esta desmotivação está associada a uma forte carga de conceitos abstratos que intervêm em todo o conhecimento envolvido na atividade de programação, onde as características próprias das linguagens e ambiente de programação, cada vez mais sofisticados e da máquina em si, tendem a dificultar a programação. Também Motil e Epstein (2000) afirmam que a maioria das linguagens de programação utilizadas nas disciplinas introdutórias apresenta uma sintaxe grande e complexa, mais adequada para ambientes de desenvolvimento industrial.

De acordo com Dijkstra (1989) e Perkins et al. (1988) a aprendizagem de programação requer um treino intensivo em resolução de problemas, envolvendo competências de diversas áreas para obter um pequeno retorno. Os autores afirmam ainda que, em simultâneo, se exige
uma precisão e atenção a detalhes muito mais elevada do que a requerida pela maioria das outras disciplinas. Porém, autores como Byrne e Lyons (2001) referem que não há nada inerentemente difícil no assunto, mas simplesmente, existem alunos que não têm as aptidões necessárias para programar, nomeadamente de resolução de problemas e de matemática. Igual opinião é manifestada em Júnior e Rapkiewicz (2004, 2005).

Jenkins (2002) refere várias causas do insucesso generalizado em disciplinas de programação, como sejam o baixo nível de abstração, a falta de competências de resolução de problemas, a inadequação dos métodos pedagógicos aos estilos de aprendizagem dos alunos, referindo ainda que as linguagens de programação possuem sintaxes adequadas para profissionais, mas não para aprendizes inexperientes. Sloane e Linn (1988) referem que algumas das capacidades exigidas são óbvias, considerando como essenciais a capacidade de resolução de problemas e alguma ideia da matemática subjacente ao processo. Consideram também que um programador deverá ser capaz de usar o computador eficientemente, tem de saber criar um programa num ficheiro, compilá-lo e encontrar os resultados gerados. O programa produzido deverá então ser testado, os bugs encontrados e corrigidos. Porém, segundo estes autores, estas são as competências fáceis de identificar. Consideram também que existem competências menos óbvias, classificadas como “competências de vida”. Os mesmos autores referem ainda que programar não consiste apenas numa única capacidade, não se trata
apenas de um conjunto de aptidões, mas antes de uma hierarquia de aptidões e um programador necessitará de muitas delas em algum momento da sua vida. Bereiter e Ng (1991) referem que um aluno que se depare com a aprendizagem de uma hierarquia de competências, geralmente começa pelas de mais baixo nível progredindo gradualmente até às mais exigentes. Os autores concretizam que, no caso da codificação (uma pequena parte de uma competência de programação) implica que os alunos aprendam a sintaxe básica e gradualmente aprendam a semântica, estrutura e finalmente o estilo.

O que sugere que aprender a programar é um processo que exige tempo e maturidade. Jenkins (2002) argumenta que a programação é normalmente ensinada como assuntos fundamental no início de um curso superior, sendo esse um momento de transição e muitas dificuldades/novidades, pois muitas vezes a transição para a universidade implica pela primeira vez uma vida autónoma, independente e longe dos familiares. Este autor considera que a programação é por si uma matéria difícil quando os alunos estão estáveis, sendo a situação agravada num período de transição. O mesmo autor refere-se também à falta de motivação devido à imagem negativa apresentada pelos programadores. Existe a imagem pública de um programador como um “desadequado social”, o que faz com que seja pouco provável que os alunos aspirem a uma imagem deste tipo. Outros autores referem que a generalidade dos alunos não tem motivação intrínseca para estudar estes assuntos e sem este tipo de motivação dificilmente serão bem-sucedidos (Bereiter e Ng, 1991). Por seu lado, Dunican (2002) afirma que os problemas verificados com os alunos de programação Irlandeses são o produto dos sistemas educativos (primários e secundários) sem módulos de resolução de problemas ou lógica, em qualquer uma das suas disciplinas. Em segundo lugar, estes autores afirmam que outro grande problema se prende com a natureza abstrata da tarefa de programar. Noções como variáveis, tipos de dados, memória dinâmica, entre outros, não têm correspondência na vida do dia-a-dia, e compreender estes conceitos fundamentais de programação não é simples. Em terceiro lugar, as exigências rígidas em termos de sintaxe quando comparada com a natureza inexata e livre da
língua inglesa faz com que muitos alunos não sejam capazes de escrever programas compiláveis bem-sucedidos.

8. CONHECENDO O SCRATCH

Mitchel Resnick, professor e pesquisador, no Media Lab do Massachusetts Institute of Technology (MIT), inspirado na filosofia educacional construcionista consolidada por Papert ao conceber o Logo no final da década de 1960, um tipo de linguagem de programação visual que possibilita estabelecer com o computador a comunicação por meio de instruções, a fim de gerar efeitos de acordo com a intencionalidade desejada, cuja ideia é incentivar as crianças a se tornarem produtoras do conhecimento com o auxílio do computador, idealizou o Scratch em 2003 e lançou em 2007, sob o slogan, “imaginar, programar, compartilhar”.

Considerado uma linguagem de programação visual de autoria, o Scratch permite criar produtos a partir da mistura de vários recursos de mídias e cenários gráficos, de modo criativo e lúdico, juntamente com a inserção da linguagem de programação visual, que consiste na montagem de blocos de comandos coloridos, comparando-se a um quebra-cabeça ou ao conceito do brinquedo LEGO. Esses blocos de comandos ao se encaixarem formam algoritmos sintaticamente corretos, resultando em instruções e/ou ações programadas para o objeto selecionado, permitindo a visualização dos efeitos antes de sua finalização. Esses blocos permitem promover os movimentos e demais ações idealizadas ao projeto, possibilitando ao usuário o desenvolvimento do domínio dessa linguagem, até mesmo para quem ainda não teve contato prévio.

Disponível no site do Scratch para baixar, podendo também ser manuseado no modo on-line e off-line, encontra-se disponibilizado em vários idiomas, inclusive em português. O Scratch é voltado para o público que se encontra na faixa etária entre 8 a 16 anos, porém pode ser usado por adultos de todas as categorias e idades (pais, professores, educadores, profissionais de outras áreas etc.).

Trata-se de um tipo de software livre, criado para utilização no âmbito educacional, com a finalidade de auxiliar a aprendizagem de conceitos matemáticos, desenvolver o pensamento.
computacional, o pensamento criativo, o raciocínio lógico e o trabalho colaborativo por meio da criação de produtos, tais como, histórias em quadrinhos, jogos matemáticos, animações, simulações, cartões etc. E ainda permite ao usuário utilizá-lo para diferentes propósitos, podendo ser adaptado de acordo com suas necessidades.

Figura 4 – Ideologia do Scratch

9. ESTRUTURA DO SCRATCH

O ambiente de programação do Scratch é composto pelos seguintes elementos para a elaboração de projetos de autoria: diversos tipos de mídias, como imagens, inserção de textos e áudios, os quais podem importados de outros arquivos ou produzidos a partir das ferramentas disponibilizadas no software. Ao abrirmos o ambiente de programação podemos visualizar cada elemento com suas respectivas funções, conforme mostramos na figura:
O Palco “Stage” (1) é o plano de fundo estático onde atores (sprites), executam as ações programadas pelos blocos de comandos. Na área para a inserção de atores “Sprites” (2), o usuário tem a opção de importar atores disponibilizados pelo próprio software, em arquivos da web ou pode até mesmo criá-los a partir da ferramenta “Paint”. Ao inseri-los, a ferramenta permite a eles ter mais de uma fantasia “Costumer”. Cada ator poderá receber comandos de instruções de maneira diferenciada, conforme comportamento desejado por meio dos encaixes dos blocos de comandos.

As paletas de blocos “Blocks” (3) são divididas em dez categorias (4): movimento, aparência, som, caneta, variáveis, eventos, controle, sensores e operadores e mais blocos, e precisam ser arrastados e soltos na área de comandos para que possam ser encaixados para gerar a ação de programação.

Na área para a inserção do roteiro “Script” (5), os blocos de comandos são arrastados, obedecendo a uma sequência lógica de sintaxe, que encaixados formam a programação atribuída a cada objeto. Podemos ver na figura 6 um exemplo de demonstração dos blocos ao serem arrastados e encaixados:
Na aba opções de salvamento e edições (6) é possível criar novos projetos, carregar, salvar um projeto no computador, editar, desfazer alterações ou feitas no projeto que está sendo trabalhado. Em arquivo podemos ainda compartilhar os projetos criados diretamente no site do Scratch.

Já na aba Botões para redimensionar os atores (7), permite que o palco e atores sejam ajustados em proporções de tamanhos maiores e menores. E por fim, a janela de dicas (8), funciona como suporte para esclarecer eventuais dúvidas ao usuário, bem como, indica sugestões de correções para ocorrências de situações de erros sintáticos na programação.

O site do Scratch dispõe de uma comunidade on-line para o compartilhamento de projetos e já ultrapassa mais de 20 milhões, onde é possível fazer o download e remixá-los, sendo que para isso, bastar criar uma conta. Outros recursos também podem ser encontrados no site, tais como: informações gerais, pesquisa de busca de projetos a partir da inserção de palavra-chave, fóruns de discussão, tutoriais para aprender a usá-lo, ajuda, acesso ao histórico dos projetos compartilhados, etc. A seguir, na figura 7, podemos visualizar a página inicial do site.
Contudo, nossa intenção com esta síntese de demonstração do ambiente de programação visual do Scratch, não esgota o conhecimento de outros elementos contidos nas demais funções. Para conhecê-lo com mais detalhes recomendamos explorá-lo a partir de exercícios básicos e projetos disponibilizados no próprio site do Scratch.
10. AGORA É COM VOCÊ...

1) Assista ao vídeo “Mitch Resnick no Transformar 2014: Formação de professores para utilizar a programação na escola”, disponibilizado no YouTube e, em seguida, de forma sucinta faça um pequeno resumo sobre.

2) Acesse o site do Scratch, crie um usuário e, partindo dos ensinamentos dados em sala de aula, faça um algoritmo em forma e fluxograma e em seguida reproduza-o no Scratch. Compartilhe com colegas e anote as experiências.
11. REFERÊNCIAS


AULA 4 – SCRATCH COMO SIMULADOR – PARTE 1

12. ENTENDENDO UM POUCO SOBRE SIMULAÇÕES

Ao longo de sua história os seres humanos têm construído modelos da realidade como a maneira de possibilitar a sua interação com essa realidade. Todas as ciências constroem modelos como forma de entendimento ou interação no campo a que se destina. Desse modo a humanidade vem construindo um cabedal de conhecimentos científicos que tem sido transmitido através dos tempos. Existem certos conceitos científicos difíceis de serem percebidos, seja por envolver um elevado grau de abstração ou por outros motivos ainda não completamente elucidados. Por exemplo, parte dos seres humanos intui a existência de uma relação direta entre a velocidade de deslocamento de um corpo e a resultante das forças que nele atua. Esse foi um tipo de relação estabelecida por Aristóteles e que figurou como entendimento predominante até Newton, quando esse último estabeleceu o paradigma vigente para o assunto até os dias de hoje. Segundo a mecânica newtoniana existe uma relação direta entre a variação da velocidade de um corpo e a resultante das forças que nele atua. Quando um aluno tem uma intuição aristotélica do movimento, ele enfrentará grandes dificuldades para um aprendizado da mecânica newtoniana, a menos que ele seja ajuda de maneira adequada a superar essa dicotomia.

A animação interativa utiliza um modelo aceito cientificamente para simular um evento específico. Podemos simultaneamente fazer animações de ideias antagônicas, e analisar quais as implicações de cada uma para o resultado final da simulação de um dado evento. As simulações computacionais possibilitam o entendimento de sistemas complexos para estudantes de idades, habilidades e níveis de aprendizagem variados. O computador, ao invés do estudante, assumiria a responsabilidade de solucionar as equações matemáticas pertinente ao sistema considerado no sentido a permitir que o estudante explore o sistema complexo focalizando inicialmente o entendimento conceitual (apud Tavares, 2008). A grande vantagem desta situação é a possibilidade de o aprendiz poder estabelecer o seu ritmo de
aprendizagem. Ele tem o controle da flecha do tempo (podendo ir e vir indefinidamente) e tem a liberdade de escolher as condições iniciais para o evento simulado, e desse modo visualizar as diversas possibilidades de evolução (apud Tavares, 2008). Desse modo cada aluno escolherá um ritmo conveniente para utilizar os recursos de uma animação, e ao agir dessa maneira ele evita uma sobrecarga em sua memória de curto prazo. Quando se apresentam informações num ritmo acima da capacidade de absorção do aprendiz, ele simplesmente irá ignorar aquilo que se configurar como sobrecarga cognitiva.

Figura 8 – algumas simulações/animações no Scratch

13. AGREGANDO CONHECIMENTO: OUTRA PLATAFORMA DE SIMULAÇÃO

O simulador PhET (https://PhET.colorado.edu/pt_BR/) é um facilitador de aprendizado desenvolvido pela Universidade do Colorado, em 2002. Foi criado pelo projeto Physics Educacional Technology (PhET), que visa pesquisar e desenvolver simulações para o ensino de ciências, matemática, física e química. O simulador PhET possibilita aos professores mesclarem aulas teóricas e práticas, visto que o uso da tecnologia torna as aulas mais dinâmicas e possibilita o aprendizado de forma interativa. Além disso, os professores podem usar o simulador como ferramenta de ensino no laboratório de informática da escola, sem a necessidade do uso da internet no momento da aula. Isso porque é possível instalar o simulador no computador ou dispositivo eletrônico,
desde que este possua um aplicativo Java, Flash ou HTML5, que permite armazenar aplicações da internet para uso offline. A quantidade de simulações prontas que favorecem o aprendizado de forma descontraída e ao mesmo tempo interessante, na forma de jogos, fazem do simulador PhET um local de potenciais aprendizagens.

**Figura 9 – Interface do site PHET**
14. FORÇANDO A CRIATIVIDADE...

1) Nessa aula, vimos muito sobre simulações no âmbito das ciências exatas; no entanto, engana-se quem pensa que as demais áreas não possam ter as suas próprias simulações. Partindo dessa ideia escolha uma temática que saia da sua zona de conforto e projete, em forma de fluxograma (ou mapa mental), uma simulação. Pense de forma criativa! Atividade pode ser realizada em, no máximo, três pessoas.

2) Quais foram as dificuldades encontradas por você e por seus colegas? Deixe-as explícitas para que possam ser corrigidas.

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
15. REFERÊNCIAS


AULA 5 e 6 – SCRATCH COMO SIMULADOR – PARTE 2

16. AMBIENTE DE PROGRAMAÇÃO DO SCRATCH

Para iniciar o Scratch, acesse o seu site (http://scratch.mit.edu/). Isso deve conduzi-lo à interface do editor de projetos do Scratch, mostrado na figura 10:

Figura 10 – Interface de edição

Você deverá ver uma única janela contendo pelo menos os três painéis a seguir: o Stage (Palco, na parte superior à esquerda), a Sprite List (Lista de Sprites, na parte inferior à esquerda) e a aba Scripts (Roteiros, à direita), que contém a aba Blocks (Blocos) e a Scripts Area (Área de Scripts). Se estiver logado em uma conta no site do Scratch, você verá também o Backpack (Mochila, na parte inferior à direita), que contém botões que permitem compartilhar o seu projeto e usar sprites (atores) e scripts (roteiros) de projetos existentes. Vamos dar uma olhada rápida nos três painéis principais.

16.1 O Stage

O Stage (Palco) é o local em que os seus sprites se movem, são desenhados e interagem. O Stage tem 480 passos de largura e 360 passos de altura, conforme mostrado na figura 11. O centro do Stage tem coordenada x igual a 0 e coordenada y igual a 0.
As coordenadas \((x, y)\) de qualquer ponto do Stage podem ser encontradas ao mover o cursor do mouse até esse ponto e observar os números na Área de Exibição da Posição \((x, y)\) do mouse, localizada logo abaixo do Stage. A pequena barra localizada acima do Stage contém diversos controles. O ícone do modo de Apresentação \((1)\) oculta todos os scripts e as ferramentas de programação e faz com que a área de Stage ocupe quase todo o seu monitor. A caixa de edição \((2)\) mostra o nome do projeto corrente. Os ícones de bandeira verde \((3)\) e de parar \((4)\) permitem iniciar e terminar o seu programa.

16.2 Lista de Sprites

A Lista de Sprites (Sprite List) exibe os nomes e as miniaturas (thumbnails) de todos os sprites de seu projeto. Novos projetos começam com o Stage em branco e um único sprite representado por um gato, conforme mostrado na figura 12.

Os botões acima da Lista de Sprites permitem adicionar novos sprites ao seu projeto a partir de um de quatro lugares: da biblioteca de sprites do Scratch \((1)\), do Paint Editor incluído \((2)\) (onde você pode desenhar sua própria fantasia), de uma câmera conectada ao seu computador \((3)\) ou de seu computador \((4)\).
16.3 A aba Blocks

Os blocos no Scratch estão divididos em dez categorias (paletas): Motion (Movimento), Looks (Aparência), Sound (Som), Pen (Caneta), Data (Variáveis), Events (Eventos), Control (Controle), Sensing (Sensores), Operators (Operadores) e More Blocks (Mais Blocos). Os blocos são diferenciados por cor para ajudar você a identificar facilmente aqueles que estão relacionados. O Scratch 2 tem mais de cem blocos, embora alguns apareçam somente em determinadas condições. Por exemplo, os blocos da paleta Data aparecem somente depois que uma variável ou uma lista for criada. Vamos dar uma olhada nos diversos componentes da aba Blocks na figura 13:

**Figura 12 - Sprite**

**Figura 13 – Componentes da aba Blocks**
Experimente clicar em um bloco para ver o que ele faz. Se você clicar em move 10 steps (mova 10 passos) da paleta Motion (Movimento), por exemplo, o sprite deslocará dez passos no Stage. Clique nesse bloco novamente e o sprite deslocará mais dez passos. Clique no bloco “say Hello! for 2 secs” (diga Hello! por 2 segundos) na paleta Looks (Aparência) para fazer o sprite mostrar “Hello!” em um balão durante dois segundos. Você também pode acessar a Introdução ao Scratch na tela de ajuda de um bloco ao selecionar Block help (Ajuda do bloco, que corresponde ao ícone de ponto de interrogação) na barra de ferramentas e clicar no bloco a respeito do qual você tem dúvidas. Alguns blocos exigem uma ou mais entradas (também chamadas de argumentos) que dizem ao bloco o que ele deve fazer. O número 10 no bloco move 10 steps (mova 10 passos) é um exemplo de um argumento. Observe a figura 14 para ver as diferentes maneiras pelas quais os blocos permitem que suas entradas sejam alteradas.

Figura 14 – diversos tipos de entradas nos blocos

Você pode alterar o número de passos em move 10 steps (mova 10 passos) ao clicar na área branca em que vemos o número 10 e inserir um novo número (1), por exemplo, 30, como pode ser visto na figura 14. Alguns blocos, como point in direction 90 (aponte para a direção 90 graus), também possuem menus suspensos para suas entradas (2). Você pode clicar na seta para baixo para ver uma lista das opções disponíveis e selecionar uma delas. Esse comando em particular apresenta uma área em branco editável, portanto você também pode simplesmente digitar um valor na caixa branca. Outros blocos, como point towards (aponte para) (3), forçarão você a escolher um valor a partir do menu suspenso.
16.4 A área de Scripts

Para fazer com que um sprite faça algo interessante, é necessário programá-lo arrastando blocos da aba Blocks (Blocos) para a Scripts Area (Área de Scripts), unindo-os. Ao arrastar um bloco para a Área de Scripts, uma área destacada em branco indica em que local esse bloco pode ser solto para formar uma conexão válida com outro bloco (Figura 15). Os blocos do Scratch somente se encaixam de determinadas maneiras, eliminando os erros de digitação que tendem a ocorrer quando as pessoas usam linguagens de programação baseadas em texto.

![Figura 15 – Área de Scripts](image)

Não é preciso completar os scripts para executá-los, o que significa que você pode testá-los à medida que os criar. Clicar em qualquer ponto de um script, esteja ele totalmente ou parcialmente criado, faz o script todo ser executado de cima para baixo.

16.5 A barra de ferramentas

Vamos dar uma olhada rápida na barra de ferramentas do Scratch na figura 16, começando por alguns dos botões. Utilize os botões Duplicate (Duplicar) e Delete (Apagar) para copiar e remover sprites, fantasias, sons, blocos ou scripts. O botão Grow (Aumentar) aumenta o sprite, enquanto o botão Shrink (Reduzir) faz com que ele diminua. Basta clicar no botão que você quiser usar e, em seguida, clicar em um sprite (ou em um script) para aplicar essa ação. Para retornar ao cursor em forma de seta, clique em qualquer área em branco da tela. Você pode utilizar o menu Language (Idiomas) para alterar o idioma da interface do usuário.
A partir do menu File (Arquivo), você pode criar novos projetos, carregar (abrir) um projeto existente em seu computador, descarregar (salvar) o projeto corrente para o seu computador ou desfazer (undo) todas as alterações feitas no projeto corrente. Os projetos do Scratch 2 têm uma extensão de arquivo .sb2 para distinguí-los dos projetos criados com a versão anterior do Scratch (.sb). No menu Edit (Editar), a opção Undelete (Recuperar) trará de volta o último bloco, script, sprite, fantasia ou som que você apagou. A opção Small stage layout (Disposição com palco pequeno) diminui o Stage e proporciona mais espaço para a Área de Scripts. Selecionar Turbo mode (Modo turbo) aumenta a velocidade de alguns blocos. Por exemplo, executar um bloco move (mova) 1.000 vezes pode levar cerca de 70 segundos em modo normal e aproximadamente 0,2 segundo em modo Turbo.

17. BLOCOS NO SCRATCH

Nesta seção, você conhecerá os diferentes tipos de blocos disponíveis no Scratch, seus nomes e os usos pretendidos. Você pode retornar a esta seção à medida que progredir, se for necessário refrescar a sua memória. Como está sendo mostrado na figura 17, o Scratch possui quatro tipos de blocos: blocos de comando, blocos de função, blocos de trigger e blocos de controle. Os blocos de comando e os blocos de controle (também chamados de blocos de pilha) têm saliências na parte inferior e/ou nas reentrâncias na parte superior. Esses blocos podem ser unidos na forma de pilhas. Os blocos de trigger, também chamados de hats (chapéus), têm a parte superior arredondada porque são colocados
no topo de uma pilha. Os blocos de trigger conectam eventos a scripts. Eles esperam por um evento – por exemplo, um pressionamento de tecla ou um clique do mouse – e executam os blocos que estão abaixo deles quando esse evento ocorre. Por exemplo, todos os scripts que começarem com o bloco when green flag clicked (quando clicar em bandeira verde) serão executados quando o usuário clicar no ícone da bandeira verde.

Os blocos de função [também chamados de reporters (informantes)] não têm saliências nem reentrâncias. Eles não podem constituir uma camada de um script sozinhos; em vez disso, são usados como entradas para outros blocos. Os formatos desses blocos indicam o tipo de dado que eles retornam. Por exemplo, os blocos com extremidades arredondadas informam números ou strings, enquanto os blocos com as extremidades pontiagudas informam se algo é verdadeiro ou falso. Isso está sendo mostrado na figura 18. Alguns blocos de função apresentam uma caixa de seleção ao lado deles. Se a caixa for selecionada, um monitor aparecerá no Stage para mostrar o valor corrente do bloco informante. Selecione um sprite e marque a caixa de seleção do bloco x position [coordenada x da posição, da paleta Motion (Movimento)]. Em seguida, arraste o sprite pelo Stage e observe esse monitor. Ele deverá ser alterado à medida que o sprite for movido para trás e para a frente.
PARTINDO DOS CONHECIMENTOS ADQUIRIDOS NESSE MÓDULO, ELABORE UMA ANIMAÇÃO/SIMULAÇÃO NO SCRATCH UTILIZANDO MAIS DE 2 OPÇÕES DE BLOCOS DISPONÍVEIS. COMECE DE FORMA SIMPLES, TESTE E EM SEGUIDA VÁ APRIMORANDO.
19. REFERÊNCIAS
