UFRJ

DOCUMENTO DE PROJETO E DESIGN: Robots

Carlos Henrique Bravo Serrado
Francisco José Taam Santos de Oliveira
Gustavo Silva Araújo
Lucas Araujo Carvalho
Lucas Moreno
Iago Rafael Lucas Martins
Markson Arguello

O Projeto

Nome do Projeto: Robots
Desenvolvedores:

- Francisco José Taam Santos de Oliveira (Game Designer)
- Gustavo Silva Araújo (Artista)
- Carlos Henrique Bravo Serrado (Programador)
- Iago Rafael Lucas Martins (UX/UI Designer)
- Lucas Araujo Carvalho (Programador/Game Designer)
- Lucas Moreno Silva (Programador)
- Markson de Viana Arguello (Programador)

Data de Início: 22/12/2021
Plataforma: PC

Design do Projeto

Visão geral

Introdução

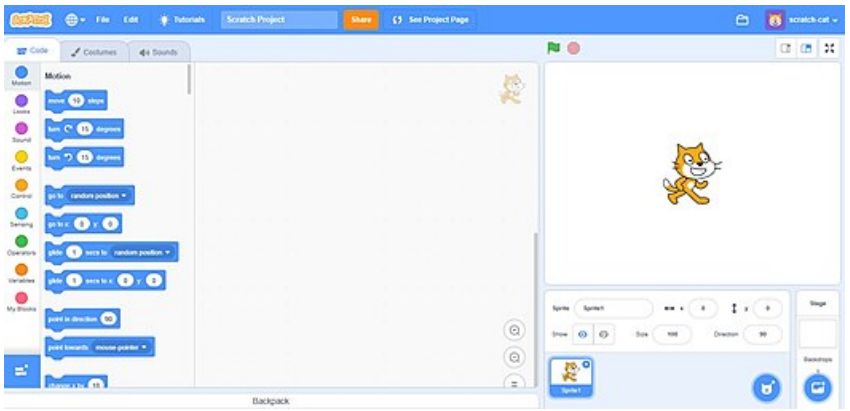
Trata-se de um jogo que visa ensinar o ensino ludificado do Pensamento Computacional, onde o jogador deve observar o funcionamento de um código e identificar como o mesmo é formado, descobrindo assim o problema e permitindo o desenvolvimento de uma solução com uma linguagem de programação em blocos.

Objetivo do jogo

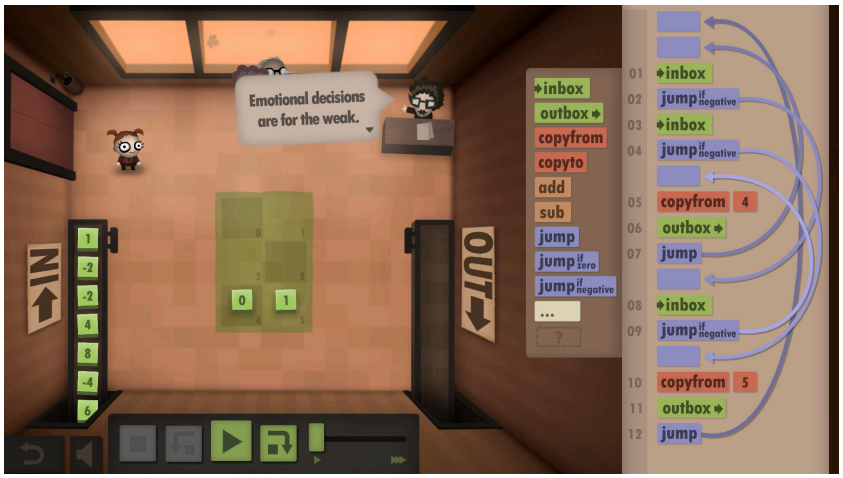
Introduzir lógica de programação de forma lúdica para discentes e interessados no aprendizado de programação, facilitando o processo de ensino por meio de um alicerce praticado de forma melhor absorvida pelo indivíduo.

Referências

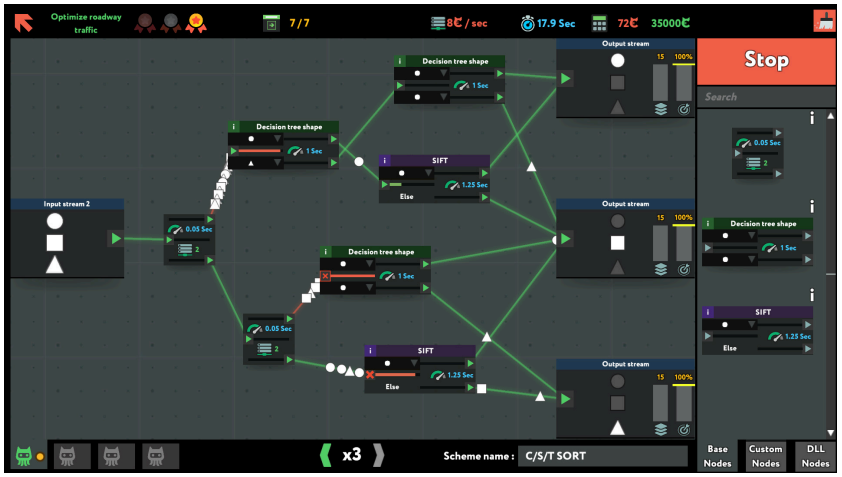
- Scratch;



- Human Resource Machine;



- While True Learn();



- Fire Emblem;

Mecânicas do jogo

Medalhas

Nas lutas, será possível desenvolver inúmeros algoritmos capazes de vencê-la. Contudo, alguns serão mais eficientes pela quantidade de passos necessários ou pelo tamanho do algoritmo construído. Por conta disso, cada fase possui uma meta de rodadas mínimas e quantidade de blocos máxima para a conquista dessas medalhas. Dessa forma, o jogador poderá refazer a fase, caso queira alcançar uma resolução de maior excelência.

Debug Notebook (Caderno de Depuração)

Haverá uma espécie de registro acerca de todo o combate em execução. Ao abrir, você poderá acessar rodada por rodada. Nos detalhes, será possível ver cada atributo seu e do seu inimigo naquele ponto, além das decisões tomadas. A ideia é que o Debug Notebook sirva para auxiliar na identificação de padrões do adversário, condicionando uma solução viável.

Variáveis

Em cada luta, teremos variáveis para serem utilizadas nas estruturas de programação para que seja possível completar seus sentidos. Elas também empenham uma função de sugerir padrões do adversário, visto que é obrigatório que exista uma solução possível com esses parâmetros.

Estruturas de Programação

Serão utilizadas para montar o código ao longo das lutas. Para esse projeto, teremos WHILE, FOR, IF e ELSE.

Comparadores

Utilizados em blocos de IF e WHILE. Utilizado para comparar duas variáveis em blocos condicionais. A comparação, se verdadeira, desencadeará o código que estiver inserido em seu bloco.

Funções

Ataca()

Golpeia o alvo selecionado. Caso o adversário defenda, nenhum dano será causado. Caso contrário, se houver carga, o dano será igual ao seu valor somado ao dano base do robô. Senão, o dano será apenas o valor base.

Defende()

Na presença de pontos de defesa, uma entidade pode chamar esta função para anular o dano dado por um golpe, independente de ser carregado ou não. Ao defender um ataque, é retirado um ponto de defesa do atributo de quem foi atacado.

Carrega()

Esta função possibilita aplicar golpes mais fortes. Sua dinâmica se dá em 2 etapas: na primeira rodada, a entidade somará um ponto de carga. Na segunda rodada consecutiva, ela somará dois pontos, ou seja, irá obter um extra. Essa lógica não segue se aplicando. Na verdade, ela apenas se repete (1^a = +1, 2^a = +2, 3^a = +1, 4^a = +2, ...)

Cura()

Seguindo a mesma lógica de sequência presente na função carrega, a função cura irá aumentar os pontos de vida da entidade, caso ela possua um valor abaixo do máximo.

Even()

Prossegue para o conteúdo do bloco IF ou WHILE no qual foi inserido, caso a variável em questão retorne verdadeiro para sua paridade.

True()

Prossegue para o conteúdo do bloco IF ou WHILE no qual foi inserido, sempre.

Atributos

Vida

São os pontos de vida. Podem ser recuperados com a função “cura”.

Defesa
Utilizada na função “defender”, serve para anular o dano causado por uma unidade inimiga. Ao receber qualquer ataque, seja ele amplificado com carga ou não, o dano é negado e a entidade que defendeu perde um ponto de defesa. Diferentemente da vida, este não pode ser recuperado ao longo de uma luta.

Carga
Trata-se do acúmulo de carga gerado pela função de carga. Se uma entidade com carga ataca, o dano aplicado será equivalente ao seu total. Uma vez dado o golpe, toda a carga é consumida. O valor da carga atual não pode ultrapassar o valor de carga definido no painel Status.

Dano
Representa o valor base para o dano causado pela função Ataca(). Se uma entidade ataca, os pontos de vida do oponente serão subtraídos pelo valor do dano dela somados ao valor da carga.

Lista de Variáveis/Parâmetros

Round
O valor referente ao atual round no qual a luta se encontra.

Constante
Dígitos de 0 a 9 que podem ser utilizados em blocos de FOR e comparações.

Atributos e Derivados
A vida, defesa e carga do jogador e do adversário. Além disso, também há variantes com relação à operações ou estados iniciais, como vida atual do jogador, metade da vida atual do jogador, dobro da vida máxima do inimigo.

Lista de Comparadores

X > Y (X é maior que Y)

X ≠ Y (X é diferente de Y)

X = Y (X e Y são iguais)

Criador de Fases

Docentes podem projetar desafios personalizados de acordo com temas e aplicações específicas discutidas em aula, ou ainda reforçar o aprendizado de alguma das estruturas de programação abordadas com uma quantidade maior de desafios. O criador permite criar o algoritmo de um robô adversário, além de configurar todos os atributos e métricas de medalhas da fase.

Level Design

(Fases Fundamentais)

(1) Cura e Ataque

Concept: Inimigo ataca, mas imediatamente fica inconsciente por um período de tempo. Nesse intervalo, o jogador precisa se curar para continuar sua luta.

```
while(true) {
att()
def()
def()
}
```

(2) Defesa e Ataque

Concept: Inimigo semelhante ao (1), mas este é letal, isto é, qualquer golpe seu matará o jogador. O jogador precisa usar do intervalo de tempo da inconsciência do inimigo para atacar, precisando defender-se quando ele despertar.

```
while(true) {
att()
def()
}
```

(3) Charge

Concept: Inimigo fraco ataca constantemente. O jogador precisa carregar um golpe poderoso para derrotar o oponente primeiro.

```
while(true) {
att()
}
```

(If, Else e While)

(4) WHILE

Concept: Inimigo letal, que segue a ideia apresentada na fase (2). Agora o jogador tem o while à disposição, o que permite utilizar uma menor quantidade de blocos.

```
while(true) {
   att()
   def()
   def()
   def()
   att()
}
```

(5) IF e ELSE básico

Concept: Inimigo que muda de padrão após determinado tempo. Usa condicionais para determinar qual padrão escolher baseado nos turnos e entra em um ritmo mais acelerado quando a batalha demora.

```
while(true) {
    if ( round > 9 ) {
        heal()
        char()
        att()
    } else {
        if ( even(round) ) {
            heal()
            heal()
        } else {
            char()
            char()
        }
        att()
    }
}
```

(6) IF e ELSE com comparadores

Concept: Inimigo realiza charge constante. Ao ter sua vida reduzida para metade do máximo, ele altera seu comportamento para ataque constante.

```
while(true) {
   if ( Ehp > HEMhp ) {
       char()
   } else {
       att()
   }
}
```

(7) FOR básico

Concept: Inimigo letal realiza muitas curas e sucessivos ataques. No entanto, há uma brecha. Essa brecha, apesar de ser razoavelmente perceptível, exigirá o uso do novo bloco FOR, que viabilizará a solução em poucas linhas.

```
while (true)
    for(4) {
        heal()
    }
   def()
   for(5) {
       att()
   }
}
```

(8) FOR avançado

Concept: Diante de um inimigo portador de enorme defesa, o jogador precisará compreender a lógica adversária para localizar uma brecha. No entanto, diferente do adversário anterior, este terá uma técnica variada. O jogador precisará identificar esse padrão complexo.

```
while(true) {
    for(3)
        def()
        for(2) {
            def()
            if(PChar = 0) {
                char()
            }
            att()
        }
    }
}
```

Interações jogador-jogo

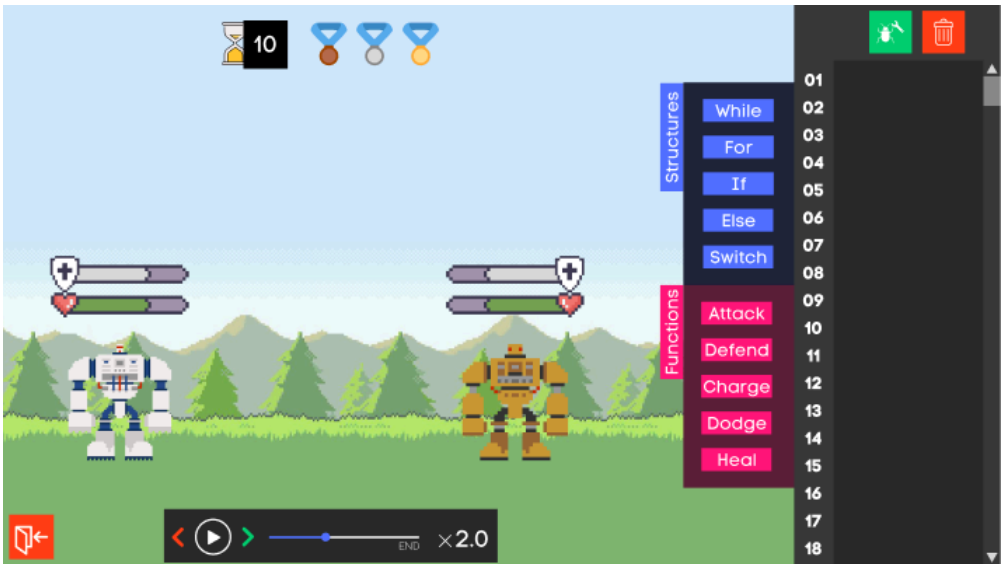
Mapas de ação (verbos do jogador)

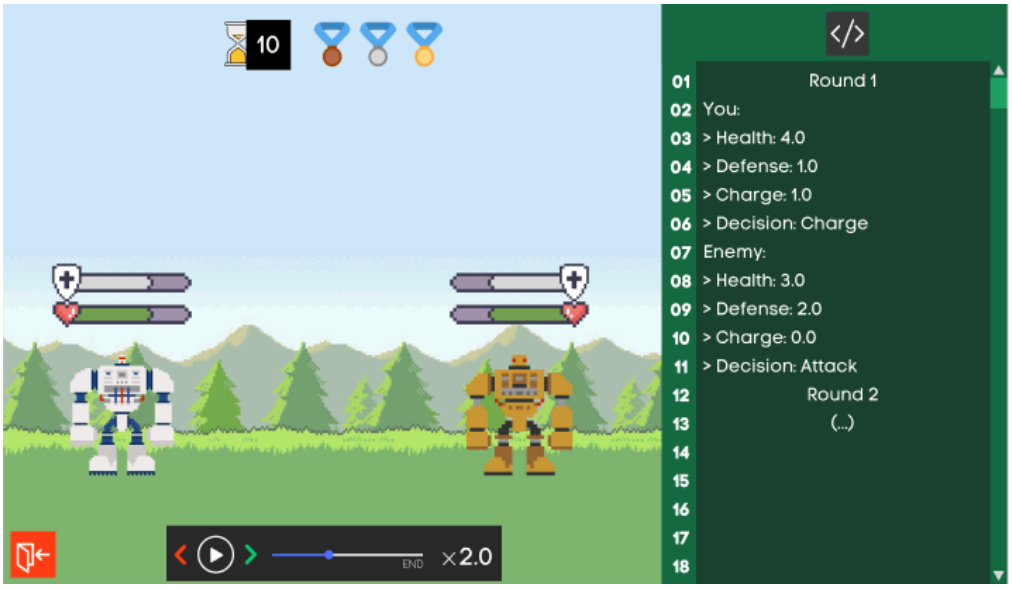
O jogador poderá montar blocos de comando para formar o algoritmo de luta do robô, arrastando os blocos para o campo de código, utilizando o cursor. Assim que estiver pronto, ele poderá executar a luta para ver o resultado. Além disso, ele poderá obter informações por meio do uso do Debug Notebook.

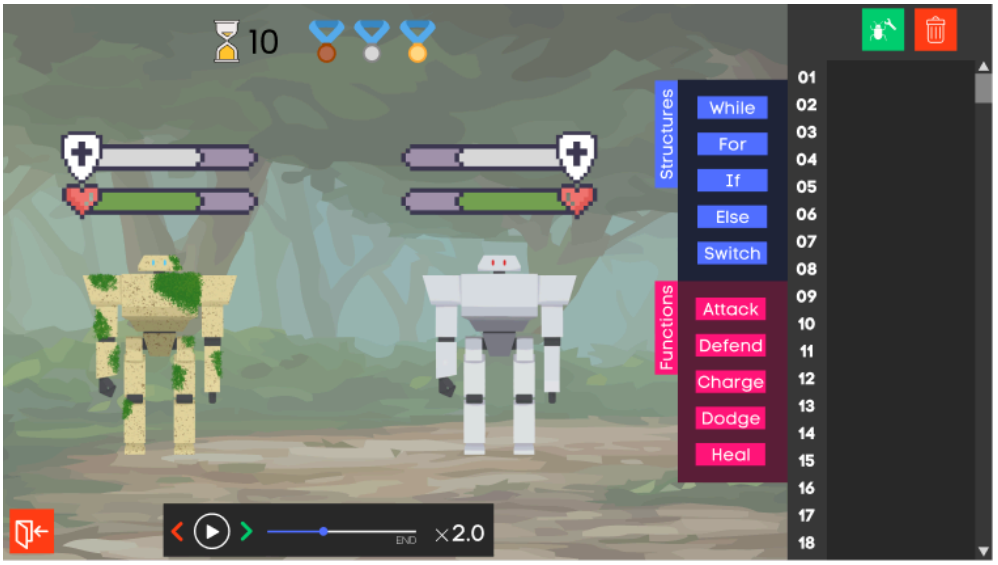
Controles

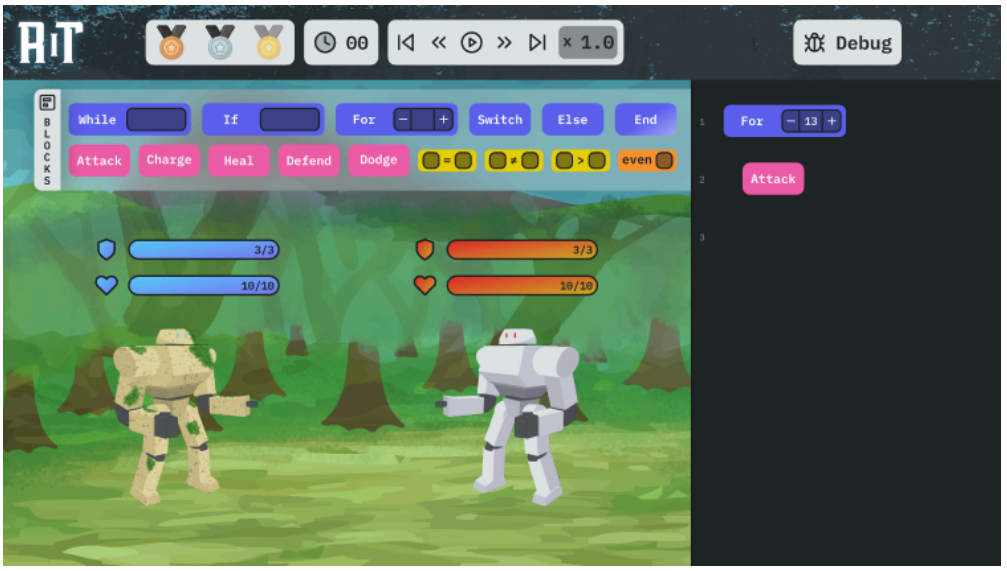
O jogador utiliza o mouse para realizar todas as ações.

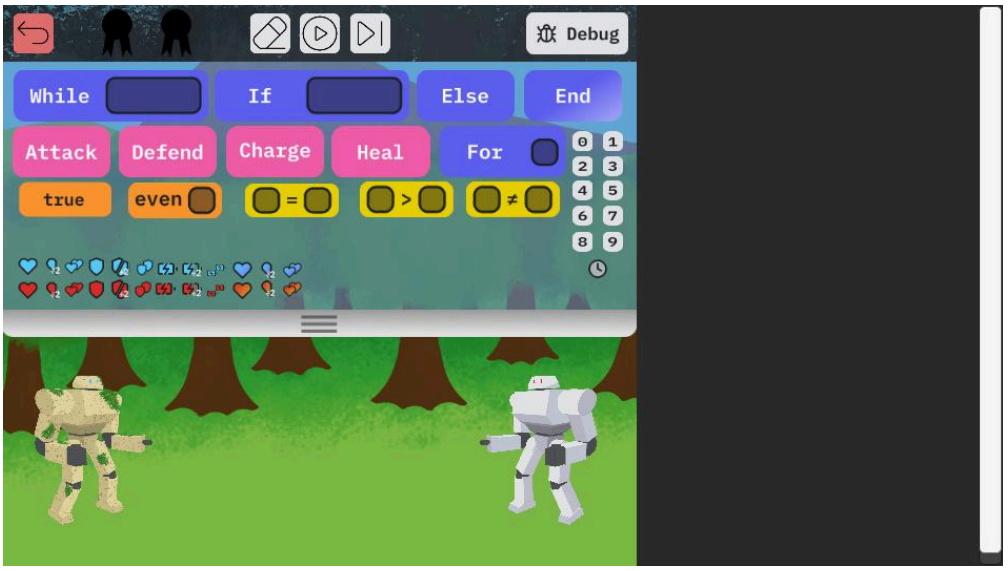
GUI (Graphical User Interface)

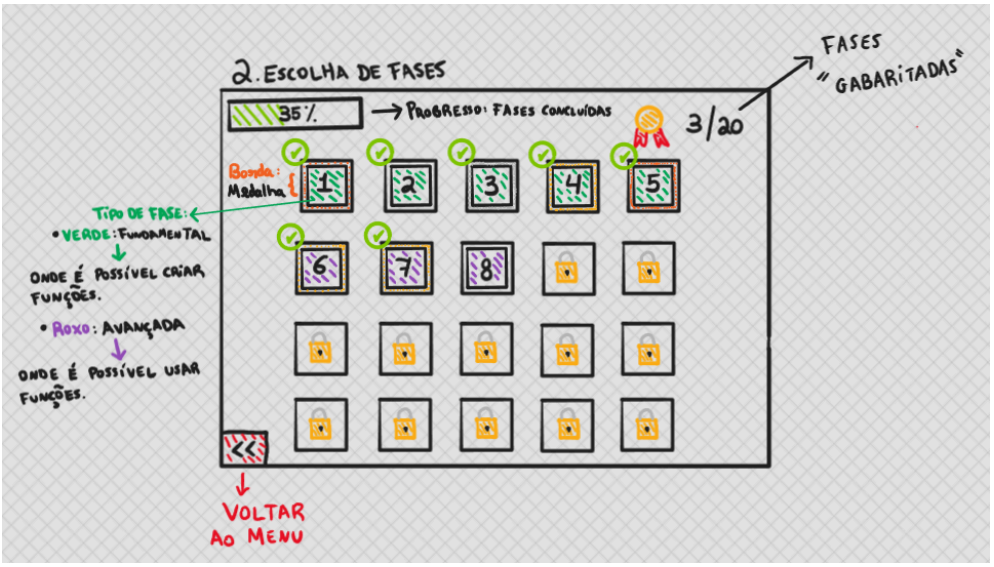





10
Structures
While
For
If
Else
Switch
Functions
Attack
Defend
Charge
Dodge
Heal
01
02
03
04
05
06
07
08
09
10
11
12
13
14
15
16
17
18
×2.0


RiT
00
× 1.0
Debug
BLOCKS
While
If
For
Switch
Else
End
Attack
Charge
Heal
Defend
Dodge
even
3/3
10/10
3/3
10/10
For
13
Attack


Debug
While
If
Else
End
Attack
Defend
Charge
Heal
For
true
even
0
1
2
3
4
5
6
7
8
9



Direção artística

Estilo de arte

Arte 2D animada por rigging, uma técnica utilizada para criar animações em que personagens ou objetos são manipulados através de um esqueleto virtual. Esse esqueleto é composto por uma série de articulações e ossos que são aplicados sobre a imagem bidimensional do personagem.