UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
BACHARELADO EM ENGENHARIA DA COMPUTAÇÃO

GUILHERME ALBUQUERQUE, HADRYAN SALLES, RICKY LEMES HABEGGER

SNAKE COM REDE NEURAL

RELATÓRIO FINAL

CURITIBA
2019

GUILHERME ALBUQUERQUE, HADRYAN SALLES, RICKY LEMES HABEGGER

SNAKE COM REDE NEURAL

Relatório final apresentado como avaliação parcial da disciplina de Oficina de Integração.

Orientador: Gustavo Benvenutti Borba e Ronnier Frates Rohrich

CURITIBA
2019

RESUMO

Abordando os temas de inteligencia artificial, protocolos de comunicação e processamento paralelo, o projeto visa desenvolver uma rede neural capaz de jogar o jogo *Snake*. Para haver uma integração de software e hardware, o projeto todo foi pensando para ser executado em microcontroladores. Estão presentes três módulos com funções divididas entre processamento da rede neural, do jogo *Snake* e da exibição de imagem. O jogo foi implementado como um simples clone do jogo original, podendo ser controlado por um usuário ou pela rede neural. A inteligencia artificial foi feita com uma rede neural aprimorada por meio de algoritmo genético. A interação entre os módulos se da por um protocolo de comunicação criado para transferir dados com auxílio da portas seriais. O processamento paralelo ocorre entre a exibição da imagem e o processamento de dados da rede neural.

Palavras-chaves: snake, jogo da cobrinha, aprendizado de maquina, evolução, inteligencia artificial, redes neurais, processamento paralelo, protocolos de comunicação, algoritmo genético, processamento de imagem.

ABSTRACT

Approaching the subjects of artificial intelligence, communication protocols and parallel processing, the project aims to develop a neural network able to play *Snake* game. To have an integration of hardware and software, the project was planned to be executed only on microcontrollers. There are three modules with functionalities divided between processing the neural network, the *Snake* game and the image displaying. The game was implemented like a simple clone of the original game *Snake* and can be controlled by the user or the neural network. The artificial intelligence was developed and trained by a genetic algorithm. The interaction between modules is made by a communication protocol that transfer data with serial ports. The parallel processing occours between the image displaying and the data processing of the neural network.

Key-words: *Snake*, *Snake* game, machine learning, evolution, artificial intelligence, neural networks, parallel processing, communication protocols, genetic algorithm, image processing.

SUMÁRIO

1 INTRODUÇÃO

Redes neurais são conhecidas por serem parte dos tópicos avançados de computação. Isto ocorre pois, frequentemente, tais estruturas estão relacionadas à expressões "quentes"da área como inteligência artificial e aprendizado de máquina. Porém, até mesmo pessoas sem nenhum conhecimento de computação podem entender o funcionamento de uma rede neural simples. Simultaneamente, existe o conceito de redes de computadores, que utilizam uma série de regras (protocolos) para conectar diferentes dispositivos. Uma vez que estes dispositivos estão conectados eles são capazes de interagir, trocar informações ou realizar tarefas paralelas. Sendo assim, com foco nos assuntos de redes neurais e protocolos de comunicação, o projeto é construir uma aplicação que utilize três micro-controladores, um deles fazendo uso de redes neurais. De forma prática, a aplicação será um jogo executado por um dos micro-controladores, jogado por outro e exibido graficamente por um terceiro.

1.1 OBJETIVOS

Este projeto tem como objetivos:

- clone do famoso jogo *Snake* (jogo da cobrinha) capaz de ser executado e jogado através de um Arduino.
- rede neural atuando como inteligência artificial capaz de jogar *Snake*, processada por um micro-controlador Tiva-C (substituído por um Raspberry Pi após problemas).
- converter dados digitais em sinal analógico utilizando uma conexão VGA, processo este que será realizado por um segundo Arduino.

Após a implementação destes três recursos, conectar os micro-controladores utilizando um protocolo de comunicação e fazer com que a rede neural controle o jogo da cobrinha exibido em um monitor de vídeo.

1.2 MARCOS

- Conteúdos programáveis básicos para o projeto. Implementar o jogo *Snake* por completo e uma rede neural genérica e aleatória com funcionalidades essenciais de criação, ativação e importação/exportação de dados. Além disso, transferir a execução do jogo e da rede neural para Arduino Mega e Tiva C, respectivamente.

- Protocolo de comunicação entre os controladores. Construir as APIs de cada módulo para realizar a comunicação entre eles. Neste marco haverá uso de bibliotecas externas.
- Algoritmo genético e treinamento da rede neural. Implementar o Algoritmo Genético e, através de um computador pessoal, aperfeiçoar a rede neural. Após a Rede Neural estar otimizada, com a melhor configuração possível para comandar o jogo, exportar seus dados para o Tiva Launchpad, podendo recriar o “cérebro"após o treinamento.

(O terceiro marco cita o microcontrolador Tiva como parte do projeto, porém para resolver alguns problemas enfrentados ele foi substituído por um Raspberry Pi.)

2 HARDWARE

O hardware deste projeto não é excepcionalmente complexo, uma vez que não conta com sensores, motores ou estruturas de composição. Porém, ao desenvolver uma integração entre os diferentes micro-controladores, foi necessário se aprofundar na forma como estes trabalham. Toda a informação sobre a direção em que "cobra"deve se mover ou onde ela deve ser desenhada, transita entre os controladores através de pulsos elétricos.

2.1 COMPONENTES UTILIZADOS

- Arduino Mega (2x)
- Tiva-C
- conector VGA (macho - fêmea)
- Raspberry Pi 3 B+

2.2 UTILIZAÇÃO

Um Arduino Mega será responsável por executar o jogo *Snake* e se comunicar com o segundo Arduino e o Raspberry. O segundo Arduino fará a conversão dos dados para saída analógica do monitor, com o envio sendo feito pelo conector VGA. O Raspberry terá que processar os dados da rede neural e enviar uma resposta para o Arduino (que executa o jogo) contendo um comando para Snake.

2.3 COMUNICAÇÃO SERIAL

Para integrar os diferentes microcontroladores foi criado uma logica de comunicação. A conexão dos microcontroladores esta especificada na figura 1

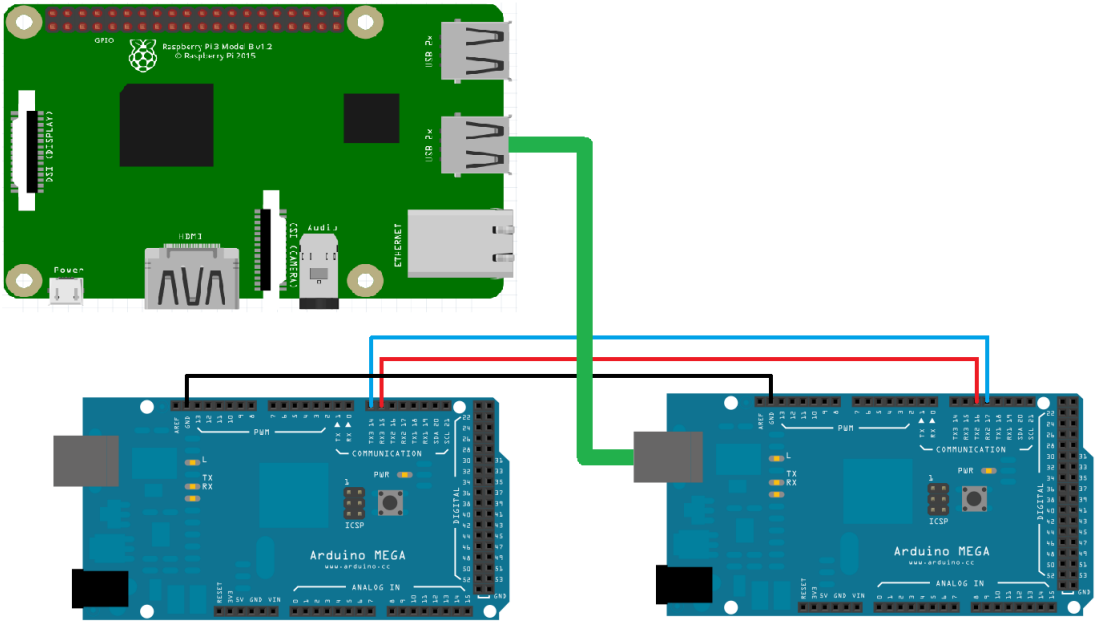

Figura 1 – Diagrama esquemático de ligações entre Arduinos Mega e Raspberry Pi

2.4 CONEXÃO ARDUINO - VGA

Para realizar a conexão entre o Arduino e o cabo VGA foi utilizado o tutorial apresentado na biblioteca "VGAX library for Arduino UNO and MEGA". O diagrama de conexões esta contido na figura 2

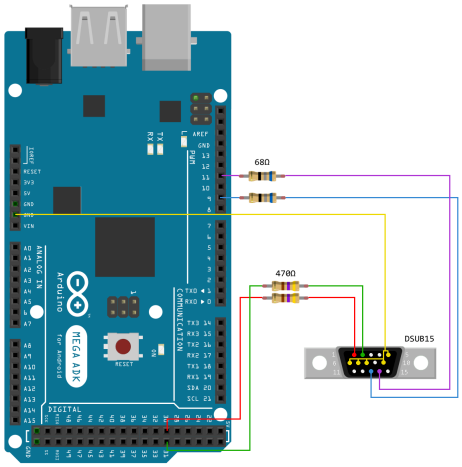

Figura 2 – Diagrama esquemático de ligações entre Arduino Mega e conector VGA

3 SOFTWARE

3.1 JOGO *SNAKE*

Foi implementado um clone do jogo *Snake*, com todas as características essenciais do jogo original. Nesse jogo, o usuário controla a direção de movimento da "cobra", podendo alterar entre esquerda, direita, cima e baixo. Há também uma fruta, elemento no mapa que aumenta a pontuação do jogador quando é capturado. Para capturar uma fruta é necessário atinga-la com a cabeça da cobra, fazendo com que uma nova fruta apareça em uma posição aleatória. O jogador perde a rodada se atingir as bordas do mapa ou colidir com o próprio corpo da cobra.

3.2 BIBLIOTECA MATRICIAL

Operações matriciais são essenciais para desenvolver uma Rede Neural. Dessa forma, o primeiro passo para a implementação da rede neural foi construir uma biblioteca de operações matriciais em linguagem C. As operações implementadas foram:

- Multiplicação de matriz
- Adição de matrizes
- Aplicar uma função(sigmóide) para cada elemento da matriz

3.3 REDE NEURAL

Inspirada no funcionamento do sistema nervoso central, estabeleceu-se na área de Inteligência Artificial uma estrutura de dados chamada Rede Neural. Esta estrutura tenta assimilar aspectos de um cérebro biológico à operações matemáticas. De forma geral, uma rede neural tem como objetivo fabricar uma resposta partindo de dados de entrada. Porém, diferentemente de uma simples função com entradas e saídas, uma rede neural pode ser treinada para responder/prever algo sem a especificação de como fazer isso. Existem diversas arquiteturas de redes neurais, entre elas, uma muito simples é chamada de Perceptron.

3.3.1 Perceptron

A rede neural Perceptron possui uma camada com um conjunto de neurônios (vértices), onde cada neurônio pode ter conexões para outros neurônios (arestas). Sendo assim, essa rede pode ser representada por matrizes, onde uma camada com N neurônios é também uma matriz [1][N] ou [N][1]. Além disso, cada conexão entre neurônios possui um valor chamado de peso sináptico.

Os dados de entrada de uma rede neural são escolhidos de acordo com a funcionalidade desejada para a rede. No caso de uma rede com objetivo de retratar a função XOR (ou-exclusivo), os dados de entrada são dois valores binários/booleanos. Quando estes dados chegam aos neurônios da camada de entrada, eles são multiplicados pelos pesos sinápticos, gerando um valor de saída denominado potencial de ativação.

Este processo, onde os valores de entrada são ativados estão representados na figura 3:

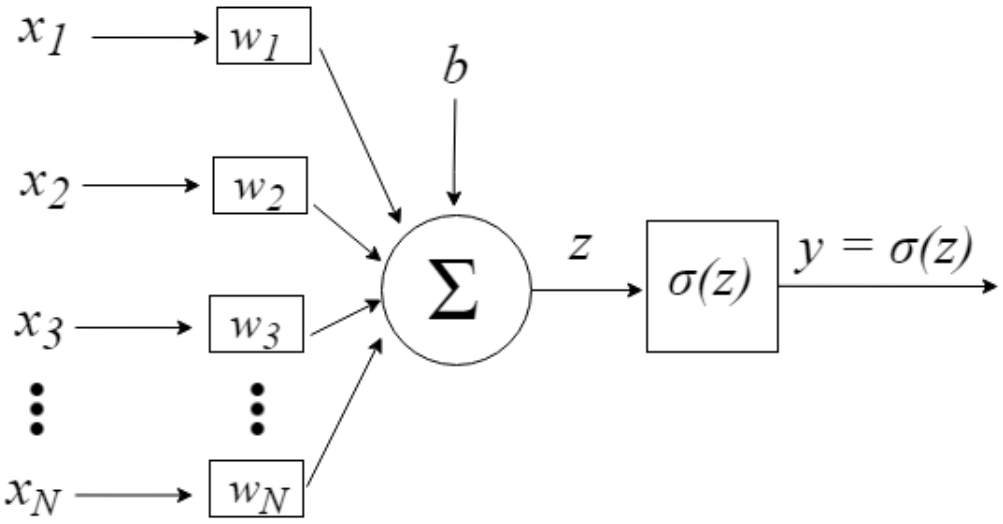

Figura 3 – Exemplo de uma rede perceptron simples onde x são as entradas, w são os pesos sinápticos, b é a bias, z é o potencial de ativação e σ é a função de ativação

As funções utilizadas na ativação dos valores de entrada são:

$$z = b + \sum_{i=1}^{i=n} x_i w_i \tag{3.1}$$

$$y = \sigma(z) \tag{3.2}$$

onde σ é a função sigmóide:

$$\sigma(x) = 1/(1 - e^{-x}) \tag{3.3}$$

3.3.2 Arquitetura utilizada - Perceptron Multicamadas

É possível combinar diversas camadas de neurônios para gerar a Rede Perceptron Multicamadas. Este foi o modelo de Rede Neural utilizado para cumprir o objetivo de jogar *Snake*. Em uma rede Perceptron Multicamadas (MLP — *Multi Layer Perceptron*) o valor de saída de cada neurônio é utilizado como entrada para uma camada seguinte. Portanto, uma forma simples de construir uma MLP é utilizando três tipos de camadas: entrada, saída e intermediarias (conhecidas como camadas ocultas). Dessa forma, quando os dados chegam na camada de entrada, eles são ativados. Então, se propagam pelas camadas ocultas até atingirem a camada de saída, onde são retornados como resposta da rede neural. A rede neural deste projeto foi implementada seguindo as descrições MLP, de forma que possa ser inicializada com os tamanhos de camada desejados. Por isso, sua estrutura é genérica, podendo ser utilizada em outros projetos que façam uso de uma rede neural MLP em C.

3.3.3 Modelagem do problema

Ao jogar *Snake*, notamos a presença de três objetos diferentes: a fruta, item que eleva a pontuação do jogador, as paredes e as partes do corpo da cobra. O corpo da cobra e as paredes são considerados obstáculos, pois levam ao fim do jogo quando são atingidos. Assim sendo, foram utilizadas características destes três tipos de objetos para modelar os valores de entrada da rede neural. Então foram criados sensores (como antenas), e colocados na cabeça da cobra, permitindo a detecção de frutas, paredes e partes do corpo. Os sensores estão distribuídos como vetores que apontam para sete direções diferentes como representado na figura 4. Essas direções são escolhidas de acordo com a orientação de movimento da cobra, de forma que não existam antenas na direção anti paralela ao seu movimento. A distância máxima que um objeto pode ser detectado é limitada, portanto, uma partícula muito distante é invisível para o "cérebro artificial". Os valores dos sensores também são ajustados de forma que um objeto detectado próximo da cabeça da cobra seja mais relevante que um distante. A resposta da rede neural é dada em três valores, que significam as possíveis escolhas de jogadas. Estas escolhas são: não alterar a direção da cobra, alterar a direção para direita ou alterar para esquerda. Toda vez que a cobra muda de direção seus sensores a acompanham. O algoritmo criado para fazer a rede neural decidir qual escolha tomar consiste em 3 passos principais:

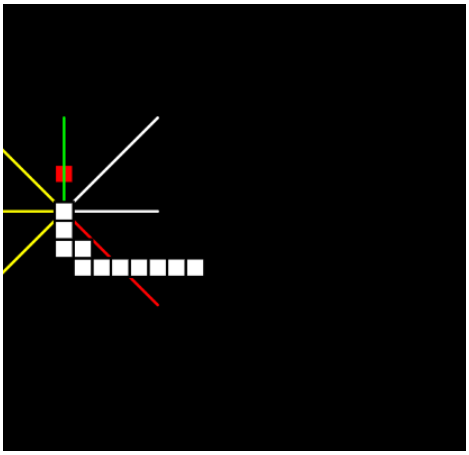

Figura 4 – Representação dos sensores da cobra. Amarelo indicando a detecção de uma parede, vermelho indicando corpo e verde fruta

- Em cada direção, utilizar os três tipos de sensores (de parede, corpo e fruta) para conferir se existem objetos naquela determinada direção.
- Caso o sensor encontre algum objeto dentro da determinada distancia máxima, seu valor é alterado para o inverso da distancia até o objeto.
- Caso não encontre, seu valor é considerado negativo, representando uma baixa relevância.
- Utilizar os valores dos sensores como entradas para a ativação da rede neural.
- Receber o retorno da rede neural, e transformar as respostas de direita, em frente ou esquerda para um comando que possa ser enviado ao jogo. Esse comando pode significar, por exemplo, que o jogador apertou a seta para cima.

3.4 TREINAMENTO DA REDE NEURAL

Uma rede neural precisa passar por um processo chamado “treinamento” para conseguir ser útil em sua determinada aplicação. No caso da rede neural deste projeto, ela tem como objetivo aprender a tratar os dados dos sensores da cobra como algo que indique, por exemplo, que as paredes devem ser evitadas e a cobra não deve colidir com elas. Sendo assim, toda vez que o sensor detectar uma parede muito próxima à cabeça da cobra, ela mudara sua direção para não colidir.

O treinamento pode ser feito de diversas formas, em casos que o resultado desejado já é conhecido, é usado o *Back Propagation*. Com o *Back Propagation*, sabendo a saída esperada para certos dados de entrada, os pesos de cada neurônio são ajustados. Dessa forma, sempre que dados "parecidos"forem inseridos na rede neural, ela responderá da maneira correta. Porém, o *Back Propagation* não é a melhor maneira de treinar uma rede neural para jogar *Snake*. Visto que, os movimentos certos para cada jogada dependem de fatores pseudo aleatórios, como a

posição da fruta. Além disso, o jogo avança e o estado muda completamente para cada fruta que a cobra come. Portanto seria necessário conhecer um número extremamente grande de estados e suas respectivas respostas corretas para utilizar o *Back Propagation*. Então, o treinamento foi feito utilizando um algoritmo genético que permite o desenvolvimento de várias redes neurais simultaneamente, de forma que as características das melhores redes sejam propagadas para as próximas gerações.

3.4.1 Algoritmo genético

Um algoritmo genético tem como base uma população de seres com uma mesma característica, neste caso uma rede neural. Porém, cada unidade desta população é diferente, a exemplo dos pesos sinápticos de cada rede. Estes pesos sinápticos são gerados aleatoriamente no início de um treinamento por algoritmo genético. Portanto, por pura aleatoriedade algumas redes neurais serão geradas melhores que outras, refletindo em decisões melhores, como por exemplo fazer com que a cobra desvie do próprio corpo.

O objetivo de um algoritmo genético é propagar a característica de uma unidade que se sobressaiu em relação às outras, para futuras gerações. Sendo assim, são geradas várias cobras com “cérebros artificias” (redes neurais) e ao fim de cada rodada (quando todas as cobras morrem), suas performances são avaliadas. Então, conhecendo as melhores cobras da geração, seus pesos são usados para a criação de novos seres. Essa “reprodução” é feita utilizando um ou mais antecessores, de forma que as cobras da nova geração herdem traços (pesos sinápticos semelhantes) das melhores redes neurais passadas. Entretanto, nenhuma unidade é gerada com total semelhança a outras. Isto ocorre pois todo novo cérebro artificial sofre uma mutação, processo em que alguns de seus pesos sofrem alterações aleatórias.

A avaliação das cobras é chamada de *fitness* e é feita de maneira que para cada uma das cobras:

- sua performance seja diretamente proporcional ao numero de frutas capturadas.
- sua performance seja diretamente proporcional ao numero de jogadas sobrevividas (sem colidir).

Um fato interessante sobre redes neurais em conjunto com algoritmos genéticos é que muitas vezes o comportamento da população converge para falhas logicas na avaliação dos indivíduos. No caso das cobras, elas "aprenderam"a burlar o sistema de avaliações, onde elas passaram a se movimentar em círculos sem dar importância para as frutas. Desta maneira, as cobras poderiam viver infinitamente, fazendo suas performances serem muito elevadas. No entanto, o objetivo do treinamento é fazer com que a rede neural aprenda a jogar *Snake* da maneira correta, visando sempre aumentar a pontuação. Então, para resolver este problema, foi adicionado a

funcionalidade de eliminar todas as cobras que passassem determinado numero de jogadas sem comer alguma fruta.

Após todas as cobras da rodada morrerem (por colisão ou por estarem a muito tempo sem capturar uma fruta), a criação da nova geração é iniciada. Essa criação começa com a avaliação de todas as cobras eliminadas, onde a nota de cada uma passa por uma normalização, e em seguida, por uma função de distribuição acumulada (neste caso, foi usada a sigmóide descrita na função 3.3). Esse processo é utilizado para ressaltar os indivíduos com as melhores pontuações, de forma que quanto maior sua nota, maior seja sua "importância" dentre o conjunto de cobras. Então acontece a reprodução, onde para cada nova cobra a ser gerada, o seguinte processo é realizado:

- sortear duas cobras da geração passada com base na probabilidade e performance (*fitness*) das melhores.
- combinar os genes (pesos sinápticos) dos dois indivíduos sorteados a fim de criar uma nova cobra com características semelhantes.

Como redes neurais são estruturas de dados que fornecem uma saída ao tratarem determinada entrada, o processo de treinamento da rede neural foi realizado em *JavaScript*. Desta maneira, uma rede neural idêntica a que seria usada no projeto (em C), com os mesmos números de camadas de entrada, intermediaria e de saída, foi treinada até atingir o comportamento desejado. Na decorrência disso, os pesos sinápticos de um "cérebro artificial" que havia aprendido a jogar foram copiados e inseridos na rede neural feita em C, onde o resultado esperado foi obtido.

3.5 PROTOCOLO DE COMUNICAÇÃO

Na intenção de enviar os dados referentes ao jogo *Snake* para os outros dispositivos, foi desenvolvida uma biblioteca baseada no modelo TCP/IP representado na figura abaixo.

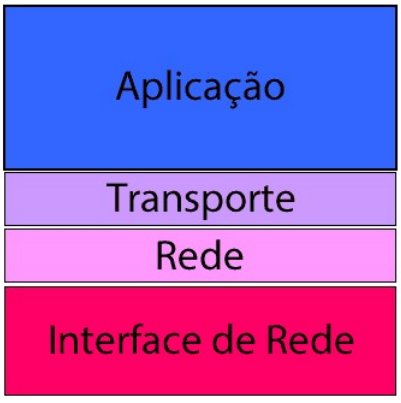

Figura 5 – Diagrama do modelo TCP-IP

Como um paralelo com as camadas desse modelo podemos dividir a implementação em 4 partes:

- Aplicação: Recebe os dados da cobra organizados em um vetor de bytes e repassa para a camada de transporte.
- Transporte: Empacota os bits de maneira a utilizar um bit de controle com cada byte enviado. No Receptor, desfaz o procedimento.
- Rede: Como a comunicação é ponta a ponta, não é necessário um endereço para o envio.
- Interface de Rede: Envia os dados através de conectores.

Referente ao transporte de dados, os bits foram organizados com uma codificação 7/8, ou seja, para cada 8 bits transferidos, 7 bits fazem parte do conjunto de dados do emissor e 1 bit auxilia na própria comunicação indicando se o dado recebido tem a finalidade de executar um comando ou é apenas informação a ser repassada ao destinatário.

Para interface de rede foi utilizado as portas seriais executando o protocolo de comunicação *Universal asynchronous receiver/transmitter* (UART), com isso, fica claro que o modelo TCP/IP foi apenas uma referência para a arquitetura do software desenvolvido. Como a biblioteca desenvolvida não realiza nenhum tipo de verificação para saber se os dados recebidos estão integros, o paralelo com esse modelo só é estabelecido no processo de preparação dos dados para o envio e do inicio da comunicação (enviado bytes de sincronia).

Inicialmente foram utilizadas as funções nativas da classe *Hardware Serial* do Arduino e do Tiva-C porém, após os problemas enfrentados com o Tiva-C (detalhados na seção 4.1), foi implementada uma classe HardwareSerial que fosse capaz de executar as mesmas funcionalidades da classe do Arduino no Raspiberry Pi, para isso, foi utilizada biblioteca externa *unistd.h*. Com essa classe foi possível utilizar a mesma biblioteca de comunicação no Arduino e no Raspiberry Pi com sistema operacional Linux Raspbian.

3.6 CONVERSÃO DE IMAGEM PARA VGA

Para realizar a transmissão de dados entre o Arduino e o monitor VGA, foi utilizada a biblioteca aberta “VGAX library for Arduino UNO and MEGA”. Com esta biblioteca é possível enviar uma imagem de até 120x60 de resolução numa paleta de 4 cores para um monitor VGA. A implementação foi feita seguindo o tutorial contido na pagina do *GitHub* dos criadores.

4 RESULTADOS

4.1 HARDWARE

Durante integração dos microcontroladores não foi possível estabelecer uma comunicação entre os Arduinos e o Tiva-C. Suspeita-se que o modelo de gerenciamento de passo do Tiva-C seja diferente, impossibilitando uma sincronia entre os dois modelos de microcontroladores. Portanto, para substituir o Tiva-C e atuar no processamento dos dados da rede neural foi utilizado um Raspberry Pi. Este novo componente, em conjunto com o sistema operacional Raspbian, tornou possível a comunicação entre os microcontroladores. O Raspberry é utilizado por meio de uma interface que simula o processamento de um microcontrolador. Desse modo, o mesmo protocolo de comunicação dos microcontroladores pode atuar no Raspberry Pi.

Um problema verificado no projeto foi o aparecimento de falhas na imagem projetada no monitor. Após diversas buscas por maneiras para amenizar essas falhas e sincronizar devidamente o sinal analógico do VGA foi constatado que o problema não era da comunicação em si, mas da plataforma na qual foi desenvolvido o projeto. Internamente, quando um Arduino precisa executar operações com precisão em um determinado tempo, ele emula *threads* que parecem estar sendo executadas em paralelo utilizando interrupções de hardware. Nesse projeto, temos no Arduino ligado ao monitor, duas operações que necessitam de precisão no tempo de execução. Estas são, a comunicação serial com o Arduino que executa o jogo e a comunicação com o monitor VGA. Assim, quando o Arduino está enviando sinais para o monitor, ao mesmo tempo ele está sendo interrompido por dados chegando na porta serial o que resulta em interferências que causam as falhas na imagem. É interessante notar também que essas falhas se agravam conforme a cobra cresce, uma vez que existe um maior fluxo de dados na porta serial.

4.2 SOFTWARE

Por parte do software, não houveram muitos problemas. Apesar da rede neural resultante do algoritmo genético não ser uma eximia jogadora de *Snake*, ela cumpre o objetivo de tomar decisões corretas em situações de fácil escolha. Outro ponto impactante na “habilidade” da rede neural é que boas decisões no inicio do jogo são diferentes daquelas que deveriam ser tomadas em um estágio avançado. Isso implica em cérebros artificiais que lidam muito bem com o jogo até uma etapa mas após certo ponto do jogo, as decisões tornam-se falhas. O fato de haver um limite de processamento e memória nos microcontroladores fez com que a rede neural utilizada não pudesse ser muito complexa. De certa forma, uma rede neural mais complexa poderia apresentar resultados melhores, como por exemplo um comportamento menos cíclico. Esse comportamento cíclico é notado na movimentação adotada pelas cobras, onde elas circu-

lam o mapa até encontrar uma fruta. Talvez uma abordagem onde houvesse uma entrada na rede neural para cada posição do mapa fosse mais efetiva, pois o "cérebro artificial"conheceria todo o conteúdo do cenário sem estar limitado aos sensores. Porém essa abordagem não seria possível, uma vez que esgotaria o processamento disponível em micro-controladores.

5 CONCLUSÃO

Apesar da grande quantidade de assuntos abordados neste projeto, o mesmo poderia ser executado de uma forma muito mais simples, econômica e eficiente. O uso de múltiplos controladores serviu apenas como meio de estudo do protocolo de comunicação e de processos paralelos. É notável que um único mini-microcomputador (Raspberry Pi, por exemplo) poderia realizar o trabalho de executar todos os módulos, inclusive a exibição da imagem, com uma qualidade superior à apresentada no projeto.

REFERÊNCIAS

NEAT, NEUROEVOLUTION. AGATETEPE, disponível em: <https://www.agatetepe.com.br/neat-uma-abordagem-incrivel-para-neuroevolution/>

Acesso em: 5 jul. 2019

VGAX LIBRARY. SMAFFER, disponível em: <https://github.com/smaffer/vgax>

Acesso em: 4 jul, 2019

USANDO NEAT PARA JOGAR MARIO. A, C, Araujo, UFMA, disponível em: <http://nca.ufma.br/~geraldo/vc/20171/trab/How_to_train_your_Mario.pdf>

Acesso em: 5 jul, 2019

REDE NEURAL PERCEPTRON MULTICAMADAS, Medium, disponível em: <https://medium.com/ensina-ai/redes-neurais-perceptron-multicamadas-e-o-algoritmo-backpropagation-eaf89778f5b8>

Acesso em: 6, jul, 2019

PERCEPTRON, DEEP LEARNING BOOK, disponível em: <http://deeplearningbook.com.br/o-perceptron-parte-1/>

Acesso em: 6, jul, 2019

SOLVING SNAKE WITH EVOLUTION. P, Binggeser, disponível em: <https://becominghuman.ai/designing-ai-solving-snake-with-evolution-f3dd6a9da867>

Acesso em: 6, jul, 2019