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

DOUGLAS YAMASAKI CUNHA
GUSTAVO LEARDINI MONTANHEIRO
LUCAS PERIN SILVA LEYSER

RELATÓRIO FINAL - OFICINA DE INTEGRAÇÃO I

RELATÓRIO DE PROJETO

CURITIBA
2019

DOUGLAS YAMASAKI CUNHA
GUSTAVO LEARDINI MONTANHEIRO
LUCAS PERIN SILVA LEYSER

RELATÓRIO FINAL - OFICINA DE INTEGRAÇÃO I

Relatório de Projeto apresentado como requisito parcial para avaliação final da disciplina de Oficina de Integração I.

CURITIBA
2019

SUMÁRIO

1 INTRODUÇÃO

Neste trabalho, é descrito o desenvolvimento de uma plataforma para o jogo Snake (1997) (Wikipedia, 2019). O processo começa desde a implementação do jogo, feita em linguagem C++ e adaptada para a execução na IDE Arduino, esta que é responsável por todo o processamento da plataforma. A exibição do jogo é realizada em uma matriz de LED RGB (16 x 32) (Adafruit, 2019). A interação humana com o sistema é dada através de um smartphone rodando o sistema operacional Android e executando um aplicativo responsável pelo controle do jogo (App-Arduino-Bluetooth, 2019). Tal comunicação se dá por meio de um módulo bluetooth (HC-05) integrado.

1.1 OBJETIVOS

Os objetivos deste trabalho são:

- Implementação do jogo em Linguagem C++
- Adaptação do código C++ para a IDE Arduino
- Exibição do jogo na matriz de LED
- Integração do módulo bluetooth à IDE Arduino
- Controle do jogo através de um aplicativo para sistema Android

1.2 JUSTIFICATIVA

O projeto aqui apresentado foi escolhido por se tratar de uma idéia já conhecida e simples mas que integra bem ambos os lados da computação (*Software e Hardware*). É levemente voltada mais para a parte a parte de programação, e não necessita de um profundo conhecimento em eletrônica, uma vez que nenhum dos envolvidos possui grandes habilidades nessa área até então. No entanto, é uma boa oportunidade de aprendizado, utilizando o conhecimento dos autores em programação para adentrar na área de eletrônica e microcontroladores.

A motivação para escolher o jogo Snake foi devido à sua grande compatibilidade com um dos elementos de hardware envolvidos, a matriz de LEDs.

2 HARDWARE

2.1 LISTA DE COMPONENTES

- Arduino Uno
- Matriz de LED RGB (16 x 32) Adafruit
- Módulo Bluetooth HC-05

2.2 MATRIZ

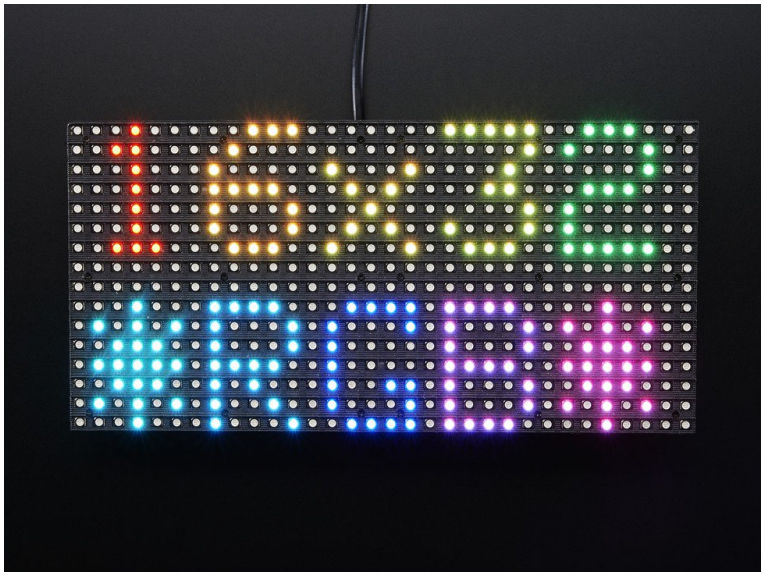

Figura 1 – Matriz de LEDs

A matriz é o componente de hardware de saída do jogo, isto é, ele mostra ao usuário as saídas relevantes do programa.

2.2.1 Energia

Embora LEDs sejam fontes luminosas muito eficientes, um grande número deles exige uma corrente considerável. Neste projeto, foi utilizada uma fonte de 2 amperes e 5 volts. Um cabo tipo Molex é responsável por fornecer energia à matriz.

2.2.2 Conexões

A matriz é ligada ao Arduino por um cabo flat de 16 vias com a seguinte disposição:

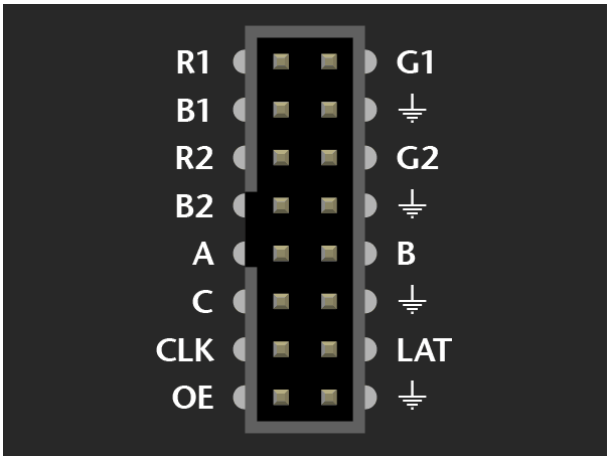

Figura 2 – Configuração dos pinos de entrada da matriz. Fonte: (Adafruit-Guide, 2019)

As conexões foram feitas em uma proto shield com a seguinte configuração:

Conexão da Matriz	Porta do Arduino
R1	2
G1	3
B1	4
R2	5
G2	6
B2	7
CLK	8
OE	9
LAT	10
A	A0
B	A1
C	A2

Tabela 1 – Tabela de Conexões

2.2.3 Funcionamento da Matriz

Uma vez que não existe documentação explicando como as matrizes funcionam, nem mesmo datasheets disponibilizados pela fabricante, segue uma explicação superficial do funcionamento desse componente.

Primeiramente, a matriz é composta de 512 LEDs e é impossível controlar todos eles simultaneamente. Principalmente por dois motivos: seria necessário uma corrente muito grande

para isso e, o custo elevado para incluir tamanha quantidade de pinos. Em vez disso, a matriz é dividida em 8 seções intervaladas (0 à 7), cada uma delas controlando 64 LEDs. A primeira seção corresponde às linhas 1 e 9, a segunda corresponde às linhas 2 e 10 e assim por diante até a última seção, responsável pelas linhas 7 e 16. Essa divisão foi feita dessa maneira para que o processo de atualização da matriz fosse mais suave, impedindo que as linhas fiquem extremamente visíveis.

No PCI existem 12 chips controladores de LED, modelo FD9802 (RaysLogic, 2019) porém, eles possuem 16 saídas e trabalham com corrente contínua. 16 saídas * 12 chips = 192 LEDs que podem ser controlados simultaneamente e, 64 * 3 (R G e B) = 192. Isso explica o motivo da partição da matriz de duas em duas linhas, com 192 saídas disponíveis, é possível controlar 2 linhas ao mesmo tempo, com cada um dos 192 R, G e B LEDs ligados ou desligados. O controlador, nesse caso o Arduino Uno R3, através dos pinos A, B e C seleciona qual seção será "desenhada" iterando até a seção 7 e, em seguida retornando para a seção 0.

A única desvantagem dessa técnica é que, embora simples e rápida, ela não possui controle PWM (Wikipedia-PWM, 2019) embutido. O controlador pode apenas ligar ou desligar os LEDs. Então, para mudar a cor de um LED, é preciso desliga-lo, atribuir uma nova cor a ele e liga-lo novamente com a essa nova cor. Um processo parecido é utilizado para deixar a cor já atribuída ao LED mais brilhante ou esmaecida, e envolve piscar o LED com frequências diferentes. Ou seja, a matriz é constantemente redesenhada em intervalos de tempo extremamente curtos a fim de realizar, manualmente, um controle PWM (GrantWinney, 2019). Por isso, é necessário um controlador rápido (pelo menos 50 Hz) para exibição de muitas cores e movimentos com boa fluidez.

2.3 BLUETOOTH

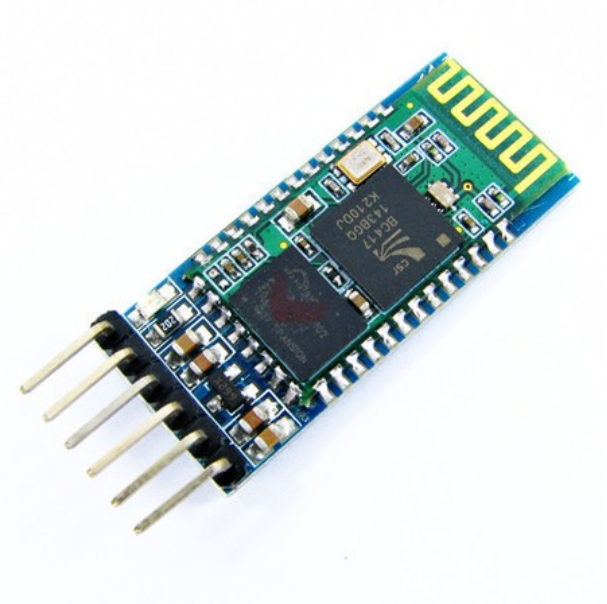

Figura 3 – Módulo Bluetooth HC-05

2.3.1 Conexões

O Bluetooth envia informações a partir de seu pino TX, e o Arduino recebe através de seu pino RX. O pino do GND é conectado ao GND do Arduino, o VCC à linha +5v do Arduino, e o pino RX (que conecta-se com o pino TX do arduino) é conectado a um divisor de tensão feito com duas resistências: a primeira de 1 kOhm, e a segunda de 2 kOhm (feito com uma serie de 2 resistores de 1 kOhm), para diminuir a tensão recebida (o módulo suporta receber somente até 3,3v).

Abaixo uma figura representando as conexões (nota-se que utilizamos o Arduino Uno, e conectamos aos pinos padrões RX e TX, não criando nossos próprios para tal uso (para simplificação).

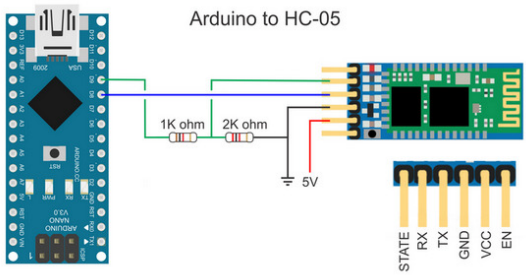

Figura 4 – Conexões do Módulo Bluetooth. Fonte: (MartynCurrey, 2019)

2.3.2 Funcionamento

Para ser feita a comunicação com o jogador, escolhemos a plataforma Bluetooth. Utilizamos nesse projeto o módulo HC-05. O aplicativo do Android transmite dados pela porta serial, que são recebidos pelo módulo HC-05, que por sua vez transmite através da porta serial para o Arduino.

3 SOFTWARE

O desenvolvimento do software, que é a parte mais central desse projeto, foi feita em duas etapas principais. A primeira etapa consistiu de uma implementação do jogo no console do Windows, para fins de teste do algoritmo. Devido à relativa simplicidade desse tipo de implementação, essa parte é um teste abstrato de como o fluxo do programa deve acontecer, já que as tecnicalidades de se operar com o Arduino e a matriz de LEDs não estão presentes nessa etapa. Com o sucesso desse teste inicial, bastaria a "tradução"desse código para uma forma que leve em consideração os detalhes técnicos da implementação, tanto no que diz respeito à matriz de LEDs quanto o que diz respeito ao Arduino.

A segunda etapa da implementação do software, efetivamente a mais importante, foi feita utilizando-se as bibliotecas presentes em (Adafruit-Guide, 2019). Uma delas implementa as funções "baixo-nível"da matriz, enquanto a outra implementa as que realmente precisamos nos preocupar em usar ("alto-nível"), que executam comandos como acender ou apagar leds, desenhar certos formatos na matriz, e apagar os leds da mesma. Houve a presença de algumas dificuldades nessa etapa, devido ao próprio funcionamento da matriz. Um exemplo disso é a necessidade de apagar a matriz totalmente a cada movimento, e reacender os leds relevantes a cada "tick"do jogo. Outra dificuldade presente aqui foi a questão da memória limitada do Arduino, e a necessidade de incluir várias bibliotecas que poderiam consumir muito essa memória. No entanto, fazendo certas economias de código, pudemos contornar essa dificuldade e fazer com que houvesse memória suficiente para uma execução satisfatória do código.

3.1 FLUXOGRAMA DO ALGORITMO

O fluxograma a seguir foi feito para ilustrar de forma abstrata o funcionamento do programa:

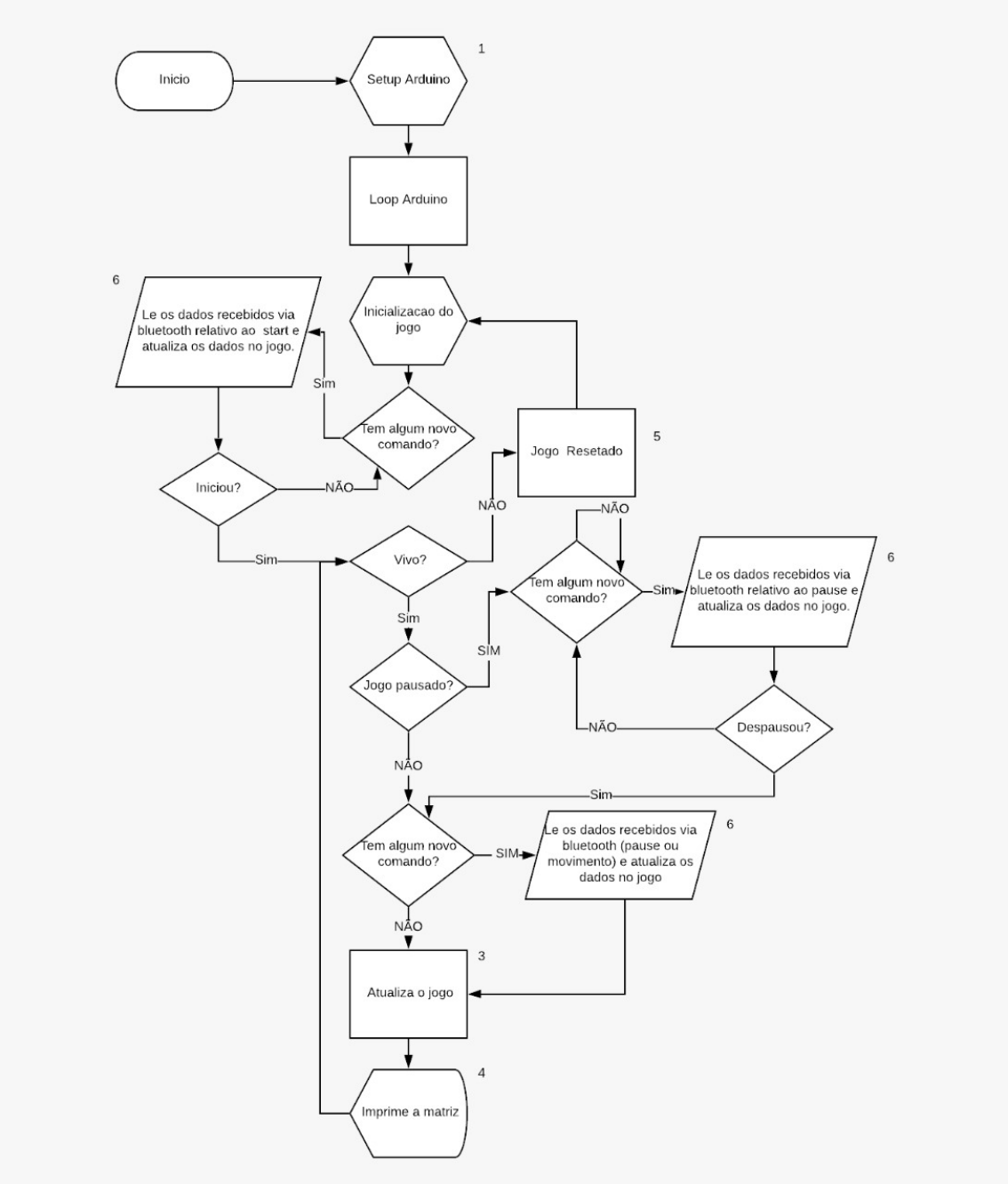

Figura 5 – Fluxograma do algoritmo

O jogo é feito dentro do loop do Arduino seguindo uma forma estrutural genérica do tipo:

```
iniciar;
while (ativo){
processos do jogo;
}
finalizar;
```

O laço principal , no exemplo while(ativo) que controla em si o jogo é dado pelo laço while (alive) que representa a presença da cobra.

São criadas as classes cobra e comida , a fim de compôr as peças membro relativas à cobra, e fazer o controle via objeto da comida. A estrutura da cobra é composta por uma lista duplamente encadeada implementada via STL (importada do C++ via biblioteca). O uso de uma lista duplamente encadeada facilita a programação do jogo. Visto que nao é necessário fazer uma função que mova peça por peça da cobra, mas apenas atualizar as posições da cabeça e cauda.

Por termos utilizado o Arduino de intencionalmente em forma *slave* é necessária apenas atenção na comunicação com a porta serial (feita através do próprio Arduino pelas portas de comunicações, rx e tx). Ao Arduino receber algum comando via Bluetooth do dispositivo Android, ele registra no serial e torna a função Serial.available ter o valor $true$. Dessa forma, sabemos que um novo comando foi inserido, e com a função Serial.read() é possível ler tal comando, e fazer com que a funcao Serial.available retorne, dessa vez, $false$, simulando assim um comportamento idêntico a um buffer. Este fato ajuda na implementação, e no próprio controle do movimento, uma vez que permite empilhamentos de comandos, assim possibilitando movimentos rápidos e precisos. Todos os comandos são registrados e atualizados utilizando a variável char buf.

Abaixo está uma descrição aprofundada dos passos desse fluxograma:

1. O setup do Arduino é feito de forma que iniciamos a porta serial, a seed aleatória a ser utilizada naquela execução e algumas variáveis essenciais, como a matriz de LEDs com os atributos já iniciados via construtor.

   A declaração da função que configura os atributos via construtor da matriz é:

   ```
   RGBmatrixPanel(int a, int b, int c,int clk, int lat, int oe, boolean dbuf).
   ```

   Onde a, b, c, clk, lat, e oe são os números da pinagem do Arduino correspondente às suas funções, conforme a tabela 1

   A última variável é sobre a utilização de um buffer duplo, o qual melhoraria o aspecto visual (animações e desenhos), porém o consumo de RAM seria muito elevado, e como as animações são simples, isso foi tomado como desnecessário.

A matriz é então iniciada com a função begin(). Esta tem a função de configurar o próprio microcontrolador, a fim de poder transmitir os seus dados via a pinagem do Arduino. São empregadas uma série de tecnicas para diminuir a latência tanto do processo de output do próprio Arduino como a da utilização dos próprios registradores.

Variáveis essenciais para o controle do jogo são também inicializadas aqui, como o delay inicial de atualização, e booleanos para o fim mencionado.

2. Este item representa uma série de processos responsáveis por setar o estado inicial do jogo. Primeiramente é resetado o valor do delay inicial (para casos em que há uma nova tentativa do jogo), é setada a direção padrão (a cobra tem direção para baixo), e chama-se a funão setJogo e setComida. a primeira tem a única função de criar os três primeiros nós da cobra e colocá-los numa posição previamente fixada (posição inicial) , e é utilizada exclusivamente aqui. O seu funcionamento consiste em alocação dos nós e empilhamento em um objeto do tipo list (da biblioteca STL), parametrizado com a classe PecaCobrinha. Já a segunda é utilizada no decorrer do código para setar as posições da comida. Ela é feita de maneira simples e direta com finalidades de se ter uma economia grande em memória RAM. Ela recebe a lista da cobrinha, e em seguida sorteiam-se casas, que são sempre comparadas com todas as peças da cobra. Caso aconteça um conflito de casas selecionadas, sorteia-se outra logo em seguida, até ser encontrada uma posição válida. Todos os tipos de retornos são feitos por via de ponteiros , evitando assim a alocação de cópias.

   São então redefinidos os booleanos de controle do jogo (alive , hasStarted , hasPaused) com o intuito de resetar a instância , limpar a matriz e desenhar as posições iniciais, e logo em seguida é iniciada a espera pelo comando de START do jogo.

3. A atualização do jogo se dá por 3 partes: atualização da cobra, da comida e da matriz. A primeira e a segunda são feitas de forma conjunta pela função update, que recebe ponteiros para a lista da cobra, e para a comida, além de uma direção (com o padrão sendo movimento para baixo , ou previamente determinada pelo input do Bluetooth), e um ponteiro para booleano indicando se há uma comida presente. A função começa tomando a direção em que a cobrinha está se movendo (o que se faz comparando as casas da cabeça com o "pescoço". Por exemplo: ao mover-se para a direita, temos o pescoço à esquerda da cabeça). Ao receber uma nova direção em que a cobrinha quer se mover (passada por referência), caso a direção escolhida seja a direção atual em que a cobra está se movendo, esta parte da função ignora a direção escolhida, pois um dos requisitos do jogo é que a cobra não possa tomar a direção reversa à que está se movendo. Em seguida, é feita a verificação de colisão da cobra com as paredes (comparando a posição da cabeça com os limites definidos pelo espaco de jogo), e com a própria cobra (comparando a posição futura da cabeça com a posição do corpo da cobra). Caso seja encontrada colisão, a função termina e a variável alive é atribuída com o valor false. Para finalmente fazer o movimento

da cobra, é criado um novo nó para a lista, que é em seguida colocado na posição da cobra no próximo tick. Procura-se a colisão da cabeça da cobra com uma comida. Caso sim, isso significa que a cobra comeu, e portanto deve crescer uma unidade (nota-se que ao criar um novo nó e empilhar, a cobra já cresceu). É mudado então o valor de comida_presente para false.

Caso não tenha ocorrido a colisão, é necessário desempilhar a última casa da cobra, e utilizamos a função pop_back relativa à classe list.

A atualização da comida é feita pela função setComida, a qual só será chamada se o booleano comida_presente for de valor false dentro do loop principal do jogo (while (alive)) e o seu funcionamento é explicado no item 1.

4. A última parte da atualização do jogo se dá pela atualização da matriz em si (output). O jogo é estruturado de forma em que cada iteração do loop principal forme um tick ou frame do jogo.

   Temos um comportamento do tipo : processar , limpar , exibir e esperar. Fazendo com que assim se crie o comportamento de troca de frames.

   A matriz então se apaga com a função fillscreen(0), ela é chamada à partir do objeto criado do tipo RGBMatrix. O parâmetro passado ’0’ indica que todos os pixels serão da cor do tipo RGB (0,0,0), ou seja, apagados.

   Após isso, é chamada a função draw, que recebe os seguinte parâmetros: ponteiro para list parametrizado com PecaCobrinha, ponteiro para objeto do tipo Comida e o booleano comida_presente. Essa função faz o uso da função drawPixel, que recebe uma posição (x,y) relativa à matriz, e uma cor no formato RGB de 4 bits, dada pela função Color444. A função drawPixel com este tipo de sobrecarregamento acenderá o pixel na matriz exatamente na posicao (x,y), utilizando o parâmetro de cor mencionado. Logo no comeco é selecionada a primeira posição da lista (a cabeça), e é escolhida a cor azul (0,0,17) para seu desenho. As posições subsequentes da cobra são desenhadas com cores aleatórias para que se obtenha um destaque ao RGB da matriz. Caso o booleano comida_presente seja de valor true, a comida é desenhada, de cor fixa verde.

5. O reset do jogo ocorre após o loop principal (while (alive)), e contém apenas dois passos, visto que na próxima iteração do laço principal do Arduino (void loop()), serão chamadas as funções explicadas no item 2. Os dois passos consistem em limpar a lista da cobrinha e reiniciar a posição da comida.

6. Como já explicado previamente, utilizamos um formato do tipo "buffer"com a porta Serial. Ao chamarmos a função read(), armazenamos seu retorno na variável buf, para imediatamente em seguida chamar uma sequências de instruções condicionais. No caso do loop principal, temos uma estrutura switch/case em que troca a direção escolhida, que é representada pela variável movdirection. É possível também para o jogador trocar o valor da

variável hasPaused via input, entrando assim em um loop de pause em que se aguarda o comando para despausar.

O mesmo mecanismo é aplicado para que o jogo se inicie (nota-se que se o jogador mandar para o serial comandos direcionais esses serão ignorados, assim não permitindo que o jogador possa trapacear no jogo armazenando instruções no buffer enquanto o jogo está pausado).

3.2 APP DE CONTROLE

Para o controle do jogo foi cogitado o desenvolvimento de um app próprio, criado pelos autores. Entretanto, por limitações de tempo hábil, e por atrasos relativos à entrega da matriz, não foi possível completá-lo. Portanto, foi necessário o uso de um app já pronto, disponível na Play Store (App-Arduino-Bluetooth, 2019).

4 RESULTADOS

Para os resultados, era importante que a atualização do jogo a cada passo da cobra fosse feita rápido o suficiente para que não houvesse um atraso muito grande entre cada um. Se o delay fosse muito grande, o jogo poderia se tornar enfadonho, o que derrotaria o propósito de sua própria existência. O Arduino consegue executar as instruções rápido o suficiente para que esse atraso seja adequado, e, inclusive, configurável. O funcionamento do jogo se dá exatamente como idealizado, com a cobrinha podendo se mover em qualquer uma das casas da matriz de LED, crescendo um pedaço extra ao comer, e "morrendo"ao colidir com si própria ou as "paredes", que são os limites da matriz. Além disso, o jogo também se acelera levemente ao comer uma comida, aumentando a dificuldade ao longo do tempo.

Quanto aos controles, eles funcionam também perfeitamente como esperado, ou seja, com as direções corretas e um delay imperceptível entre emissão do controle (ao pressionar o botão) e recepção pelo jogo.

5 CONCLUSÃO

O processo de desenvolvimento desse projeto foi uma grande oportunidade de aprendizado para os autores. Através da aplicação de conhecimentos já adquiridos, foi possível não só criar este projeto, como também aprender mais e desmistificar o desenvolvimento de algo de forma independente.

Houveram, naturalmente, dificuldades envolvidas no processo, tanto de responsabilidade dos autores, quanto advindas de razões além do controle destes. As dificuldades de responsabilidade dos autores foram principalmente relacionadas a tempo e coordenação das ações dos membros da equipe, mas foram superadas com bastante trabalho e dedicação ao projeto. As que não foram de responsabilidade dos autores foram relacionadas à entrega da matriz pelos correios, e problemas técnicos em relação à disponibilidade de boa documentação de código das bibliotecas utilizadas, e limitações da placa Arduino.

Quanto às conclusões do projeto em si, foi possível compreender os meios de se lidar com a implementação de código em um Arduino, assim como os problemas associados. O projeto original teve que ser levemente alterado, para que pudesse ser completado com resultados satisfatórios. No entanto, a ideia original foi mesmo assim cumprida, tendo um jogo perfeitamente funcional e controlado da maneira proposta inicialmente.

REFERÊNCIAS

Adafruit. 2019. Disponível em: <https://www.adafruit.com/product/420>.

Adafruit-Guide. 2019. Disponível em: <https://learn.adafruit.com/32x16-32x32-rgb-led-matrix/overview>.

App-Arduino-Bluetooth. 2019. Disponível em: <https://play.google.com/store/apps/details?id=com.giumig.apps.bluetoothserialmonitor&hl=pt_BR>.

GrantWinney. 2019. Disponível em: <https://grantwinney.com/how-to-use-an-rgb-multicolor-led-with-pulse-width-modulation-pwm-on-the-raspberry-pi/>.

MartynCurrey. 2019. Disponível em: <http://www.martyncurrey.com/arduino-to-arduino-by-bluetooth/>.

RaysLogic. 2019. Disponível em: <http://www.rayslogic.com/propeller/Programming/AdafruitRGB/FD9802.htm>.

Wikipedia. 2019. Disponível em: <https://en.wikipedia.org/wiki/Snake_(video_game_genre)>.

Wikipedia-PWM. 2019. Disponível em: <https://en.wikipedia.org/wiki/Pulse-width_modulation>.