UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
ESCOLA DE INFORMÁTICA APLICADA

TuKolk:
Um jogo de console utilizando a plataforma Arduino

André Alves Oliveira
Fabrício Kolk Carvalho

Orientador
Leila Cristina Vasconcelos de Andrade

RIO DE JANEIRO, RJ – BRASIL
AGOSTO DE 2014

TuKolk:
Um jogo de console utilizando a plataforma Arduino

André Alves Oliveira
Fabrício Kolk Carvalho

Projeto de Graduação apresentado à Escola de Informática Aplicada da Universidade Federal do Estado do Rio de Janeiro (UNIRIO) para obtenção do título de Bacharel em Sistemas de Informação.

Aprovada por:

_____
Leila Cristina Vasconcelos de Andrade (UNIRIO)

_____
Luiz Amâncio Machado de Sousa Júnior (UNIRIO)

_____
Gleison dos Santos Souza (UNIRIO)

RIO DE JANEIRO, RJ – BRASIL.
AGOSTO DE 2014

Agradecimentos

Fabrício gostaria de agradecer as professores Geiza e Leila por toda a ajuda durante o período acadêmico, em inúmeras situações. Gostaria também de agradecer a minha família. Em especial à mamãe Lilian que sempre me deu força e amor, tia Tetê, vovó Sônia, e a inesquecível e sempre presente Renata. Agradecimentos ao amigo André, grande Tuka, parceiro de muitas aventuras, e ao Fábio pela amizade quântica. Aos amigos distantes das estrelas, Landell, Eduardo e Antonella.

André gostaria de agradecer ao professor Tanaka, por ser um profissional em excelência, o qual o inspirou em sua vida profissional. Gostaria de agradecer também a toda minha família, em especial aos meus amorosos pais, Cláudio e Manoela e minha linda irmã Luiza. Agradeço também aos amigos que me apoiaram durante o curso acadêmico, em especial o Fabrício Kolk, que apesar de anos sofridos de trabalhos e provas, provocou risadas e compartilhou comigo esta jornada de sucesso.

Nosso agradecimento conjunto à Unirio, seu corpo docente, discente e técnico administrativo por permitir a realização de todos esses estudos e o presente trabalho.

RESUMO

O presente trabalho é o resultado da pesquisa dos orientandos, que passando por diversos obstáculos, confeccionaram e programaram um console de videogame baseado em arduino. Utilizando tela touchscreen e display de LCD, o console é utilizado na jogabilidade de um viciante jogo programado pelos orientandos.

Tendo por objetivo ser uma introdução ao desenvolvimento de jogos, este estudo analisa cada nível com saberes e competências vistas no ambiente acadêmico, relacionando de forma lúdica conceitos teóricos com um projeto prático.

Palavras-chave: arduino, lcd, shield, jogo, mega.

ABSTRACT

The present work is the result of the mentees research, that through various obstacles, programmed and manufactured a video game console based on Arduino. Using touch screen and LCD display, the console is used in a gameplay of an addictive game programmed by the mentees.

With the objective to be an introduction to game development, this study analyzes each level with knowledge and skills seen in the academic environment, relating playfully theoretical concepts with practical design .

Keywords: arduino, lcd, shield, game, mega.

Índice

1 Introdução

1.1 Motivação

O projeto de graduação visa demonstrar a capacidade dos discentes de construir sistemas, atestando os conhecimentos angariados na faculdade e justificando os estudos e matérias passados, provando que estão aptos a ingressar no mercado, por demonstrar a principal capacidade proposta, que é o de propor soluções tecnológicas.

O Curso de Sistemas de Informação da Unirio tem ênfase no desenvolvimento de software, de forma que os alunos se encontram afinados com as tendências do mercado, que demanda cada vez mais desenvolvedores.

Porém, a ênfase demasiada no software cria um distanciamento do hardware e diminui a capacidade de inovação dos alunos, que tenderão a pensar em soluções via software, dado o ensinamento recebido na vida acadêmica. Em contramão dessa tendência, podemos observar a quantidade de aparelhos eletrônicos lançados no mercado e sua popularização, desde celulares Iphones, impressoras 3D, google glasses, consoles de video game, mostrando que grande parte do mercado está dirigido não só pelo software, mas pelo hardware que os embarca.

Comprovando-se a importância do universo do hardware e o distanciamento dos estudantes com relação a esse universo, propomos um projeto final, utilizando a plataforma de microcontrolares Arduino, de forma a nos apropriarmos de uma visão global da produção de Sistemas, aliando o universo do desenvolvimento de software com o da confecção do hardware.

O nosso projeto se propõe a ser um console de jogos portátil, que terá jogos por nós programados, utilizando tela LCD. Esse projeto irá demonstrar capacidades de desenvolvimento de software já tradicionalmente esperadas, além de acrescentar o diferencial da confecção da plataforma dos jogos, que são um traço cultural do século 21 e que estão na vanguarda da indústria tecnológica no mundo contemporâneo.

Portanto, o projeto de console com o uso do Arduino nos permitirá ter uma visão diferenciada sobre inovação, fazendo uso não só as habilidades tradicionalmente esperadas, como também demonstrando um potencial de abstração aliado como uma visão concreta de soluções via hardware.

1.2 Objetivos

O objetivo da presente monografia é o de criar um console de videogame, montando o hardware necesssário, assim como programando o software de seu jogo, deixando como legado um estudo teórico e prático sobre os fundamentos eletrônicos, físicos, matemáticos e programáveis de consoles de videogame e seus jogos.

Pretendemos atingir esse objetivo realizando um estudo teórico do tema multidisciplinar e uma implementação prática de um protótipo, com os conceitos explanados, utilizando a plataforma de hardware livre, Arduino.

1.3 Organização do texto

O presente trabalho está estruturado em capítulos e, além desta introdução, será desenvolvido da seguinte forma:

· Capítulo II: Nesta seção será explorado o background tecnológico do projeto, estudando todos os conceitos eletrônicos, físicos, matemáticos e programáveis relevantes.

· Capítulo III: Nesta seção as tecnologias efetivamente utilizadas serão apresentadas, ressaltando os dados técnicos que sejam relevantes para o projeto.

· Capítulo IV: Nesta seção será demonstrado o passo a passo do desenvolvimento do projeto, com as dificuldades encontradas e as soluções desenvolvidas.

· Capítulo V: Conclusões – Reúne as considerações finais, assinala as contribuições da pesquisa e sugere possibilidades de aprofundamento posterior.

2 Background Tecnológico

Nesta seção, abordaremos a infra-estrutura do Arduino, as unidades fundamentais da eletrônica e seus componentes (transistores, portas lógicas), a formação de blocos lógicos e organização de computadores. Falaremos sobre microcontrolares e microprocessadores, de forma a situar o background técnico sobre o arduino e o seu sistema computacional.

Nesta seção também serão introduzidas noções de programação de jogos, falando de tópicos pertinentes, como inteligência artificial, engines, entradas e saídas do usuário, laços de repetição, física, computação gráfica, matemática computacional e demais tópicos relevantes para o projeto.

2.1 Elétrons e eletricidade

Nesta seção abordaremos tópicos que giram em torno de elétrons, corrente elétrica, medidas elétricas e alguns componentes como resistores, capacitores e transistores.

Todo o estudo de eletrônica começa com a concepção de elétron, passando pelos modelos atômicos de Demócrito, pelos de Bohr e demais evoluções. Para o nosso estudo basta compreendermos que os átomos possuem elétrons de carga negativa, que com o seu deslocamento são capazes de gerar eletricidade. Uma grande variedade de componentes foi criada para poder manipular a eletricidade e fazer uso dela das mais diversas formas.

Qualquer sistema eletrônico é alimentado por uma fonte de energia externa. No caso do projeto do Arduino alvo desta monografia, a fonte vem do próprio computador, acoplado ao Arduino via entrada USB.

As unidades fundamentais na eletrônica são a corrente, a tensão e a resistência. Uma diferença de potencial elétrico, gerada pelo acúmulo de elétrons em uma seção, e sua consequente ausência em outra, gera polaridades e com isso uma corrente elétrica com determinado sentido. Essa corrente enfrenta os obstáculos dos materiais que percorre, gerando resistência.

A definição de corrente elétrica é a quantidade de carga que passa entre 2 pontos, matematicamente sendo a quantidade de carga (medida em Coulombs) por unidade de tempo. Ela pode ser alternada, que oscila entre os pólos, ou contínua, que mantém os seus polos constantes.

A equação que relaciona essas 3 unidades fundamentais da eletrônica, é conhecida como Lei de Ohm, sendo matematicamente escrita como: V= R*I, aonde V é uma unidade de tensão medida em Volts, R é uma medida de resistência medida em Ohms e I é uma unidade de corrente elétrica, medida em ampéres.

Outra equação importante para o estudo da eletrônica é a de potência, que é compreendida como a capacidade de se realizar trabalho. Matematicamente usa-se a equação: P = V * I.

Para podermos manipular essas grandezas elétricas, fazemos uso de diversos componentes. Entre os principais, temos:

- Resistores: A resistência, como já dito, é medida em Ohms e apesar de geralmente possuirmos componentes específicos, onde pode-se fazer a leitura de valores, assim como as faixas de tolerância, é importante lembrar que qualquer material, inclusive os fios, também apresenta níveis de resistência.

- Capacitores: Capacitores são dispositivos que permitem o armazenamento de energia, graças a um conjunto de placas que possui uma certa distância entre si. Eles possuem uma medida de capacitância, medida em Farads.

- Transistores: São utilizados principalmente para aplicações de amplificação. Possuem 3 contatos, a base, o emissor e o coletor. Os mais famosos são os que possuem as junções PNP e NPN. Com transistores, podemos controlar grandes correntes usando pequenas correntes.

2.2 Eletrônica digital

Aqui falaremos sobre algebra booleana, portas lógicas e demais conceitos.

O universo da eletrônica digital possui apenas 2 valores possíveis, *true* ou *false*, sendo que como estamos falando de dispositivos eletrônicos, esses valores são traduzidos respectivamente como valores de tensão elétrica positivos ou nulos. Para se desenhar circuitos digitais, transistores são agrupados formando as chamadas portas lógicas, que nada mais são do que dispositivos capazes de operar respostas lógicas diferenciadas, podendo manipular bits, de acordo com as necessidades dos projetos. Essa manipulação é feita utilizando-se a algebra booleana, veremos a seguir sobre cada um destes conceitos:

As portas lógicas, podem ser de diversos tipos, notadamente temos:

- AND: A porta AND realiza uma multiplicação lógica de suas 2 entradas, de forma que apenas obtém um valor true quando as duas entradas são true.

- NAND: A porta NAND é a negação lógica da porta AND, de forma que apenas obtém o valor false quando as suas duas entradas são true.

- OR: A porta OR funciona de acordo com o princípio da soma de suas duas entradas, de forma que basta apenas uma entrada ser true, para a saída (output) ser também true.

- NOR: A porta NOR é a negação lógica da porta OR, portanto a única forma de se obter um valor true é a partir de duas entradas false.

- XOR: A porta XOR retorna valores true quando as suas entradas são opostas, ou seja, possui uma entrada true e a outra false.

- XNOR: A porta XNOR é o oposto lógico da porta XOR, retornando TRUE apenas quando as duas entradas são iguais, ou seja, ou quando ambas são false, ou quando ambas são true.

A álgebra booleana é utilizada para converter para o domínio matemático as operações lógicas desenvolvidas com as portas lógicas e demais componentes. Existem 3 classes, a aritmética, as identidades algébricas e finalmente as propriedades avançadas.

- Aritmética

As propriedades aritméticas garantem a lei aditiva (operação OR) e a lei multiplicativa (operação AND).

- Identidades algébricas

Identidades aditivas:

- Soma com zero: x+0 = x
- Soma com 1: x+1 = 1
- Soma com si próprio: x+x = x
- Soma com o complemento: x+/x = 1

Identidades multiplicativas:

- Multiplicação com zero: 0*x = 0
- Multiplicação com 1: 1*x = x
- Multiplicação com si: x*x = x
- Mutiplicação com o complemento: x*/x = 0

- Propriedades avançadas:

- Propriedade comutativa da adição: x+y = y+x
- Propriedade comutativa da multiplicação: x*y = y*x
- Propriedade associativa da adição: x+(y+z) = (x+y) + z
- Propriedade associativa da multiplicação: x*(y*z) = (x*y)*z
- Propriedade distributiva: x*(y+z) = x*y + x*z

Outra ferramenta muito importante é o teorema de De Morgan. Basicamente ele transforma logicamente uma adição (OR) em uma multiplicação (AND). Unindo o ferramental matemático junto com o eletrônico, podemos construir circuitos capazes de manipular bits de informação de diversas maneiras.

2.3 Blocos lógicos

Neste seção apresentaremos um breve resumo sobre decodificadores, multiplexadores, flip-flops, buffers, entre outros. Vamos discorrer sobre conceitos importantes também, dos chamados blocos lógicos.

Decodificadores decodificam entradas de bits, mapeando-as. Eles são muito úteis quando queremos mapear as entradas para todas as possíveis saídas. São circuitos combinacionais que convertem entradas de N bits em M linhas de saída, de forma que cada linha de saída é apenas ativada por uma combinação específica de entradas.

Multiplexadores são circuitos que possuem N entradas e nos permitem selecionar uma dessas N entradas e jogá-la na saída (output).

Buffers são dispositivos que armazenam dados ou os amplificam.

Os blocos funcionais aritméticos que são importantes para o nosso tipo de projeto são os comparadores, somadores e ALUs, melhor detalhados abaixo:

- Comparadores: São dispositivos que comparam dois inputs, resultando em outputs "*tais quais maior que"*, "*menor que"* ou *"igual"*.

- Somadores: São dispositivos que pegam duas entradas binárias, realizam a soma e disponibilizam o resultado no output.

- ALUs: As unidades lógicas e aritméticas somam, subtraem, comparam e aplicam operações lógicas aos operandos. São o centro nervoso dos processadores, sua essência. Um processador em última instância apenas move memória e realiza operações matemáticas e lógicas.

Flip-Flops são circuitos que possuem dois estados estáveis que podem ser usados para armazenar a informação. A memória é a base de toda lógica sequencial.

Outro conceito importante é o de contadores, que são dispositivos que contam sequencias binárias repetidas.

Uma máquina de estados é um dispositivo que se baseia em comportamentos diferentes, executados por um número de ciclos, até um outro comportamento (estado) ser selecionado. Cada comportamento é um conjunto de instruções de código, que em determinadas condições muda de estado.

Com base nos conceitos descritos acima, podemos compreender melhor como o nosso console funciona em baixo nível. Um microprocessador é apenas uma máquina de estados que busca, decodifica e executa instruções. As entranhas de um processador são apenas portas lógicas, buffers, flip flops, ALUs e conectores. Todos estes controlados por sinais, que são binários, TRUE ou FALSE, que controlam o fluxo de dados, que nos permitem controlar a execução de um programa. Estes programas nada mais são do que drivers de máquinas de estados que são mutáveis via memória.
(Lamothe, André. 2005.)

2.4 Arquitetura de computadores

Nesta seção discorreremos sobre a teoria da computação, microprocessadores, microcontrolares, Assembly, registradores, processadores, arquiteturas, entre outros.

Microprocessadores são dispositivos que executam programas, acessam memória e dispositivos de entrada e saída. Microprocessadores são constituídos por unidades de controle, unidades de lógica aritmética, registradores, flags, contadores, e trabalham com o auxílio de memórias RAM, ROM e dispositivos de entrada e saída.

Microcontroladores são microprocessadores com hardware adicional para memória, entradas e saídas e etc. O Arduino é, portanto, um microcontrolador.

O design de microprocessadores pode ser no estilo RISC ou CISC, vejamos cada um deles:

- CISC: Processadores CISC possuem instruções longas e complexas. A idéia é dar o máximo de liberdade para os programadores, possuindo uma instrução para praticamente tudo.

- RISC: Com o objetivo de focar nas instruções que realmente são utilizadas, nasceu o RISC. Possuem um pequeno conjunto de instruções, que são executadas em alta velocidade.

Cada processador possui o seu conjunto de instruções. As instruções consistem em um número de códigos de operação que um processador pode executar. Cada instrução é representada por um padrão binário de bits. A representação binária de cada código de operação é o que chamamos de código de máquina, e a representação desses códigos de operação em mais alto nível é a linguagem assembly.

A maior parte dos processadores tem uma arquitetura baseada na chamada máquina de Von Neumann. Nessa arquitetura, temos 3 segmentos interligados:

- Unidade de processamento: Possui unidades de controle, Unidade lógica e aritmética e registros.
- Entrada e saída: Periféricos de entrada e saída
- Memória

As tarefas que um processador executa são as seguintes:

- Buscar instruções: Ação de buscar a próxima instrução da memória, a ser executada.
- Decodificar: Ação de decodificação pelo sequenciador interno.
- Executar: Aqui a unidade de controle executa os comandos de microcódigo para as partes competentes.
- Reescrever: Ação de escrever o resultado das instruções na memória ou registradores.

A forma como se escrevem as microoperações, descreve o funcionamento do processador. As microoperações são compostas de:

- Registros e memória
- As operações dos registros
- Unidade de controle

As operações em alto nível com registradores, memória, multiplexadores, entre outros, são descritas pela lógica de transferência de registros.

Em processadores há os “datapaths” e as unidades de controle. Os “datapaths” são os registros e a lógica que os une. As unidades de controle são responsáveis por essas operações.

A unidade de processamento precisa de dois tipos de memória: RAM e ROM. Memórias RAM são memórias voláteis que são limpas a partir do momento que a fonte de alimentação é desligada. Memórias ROM são memórias de leitura, aonde armazenamos o firmware com o seu conjunto de instruções programadas em hardware.

A interface de entrada e saída (I/O) começa com o mapeamento dos espaços de entrada e saída, utilizando decodificação de memória.

Decodificação de memória baseia-se na lógica combinacional que utiliza as vias de endereçamento e controle para selecionar memórias diferentes ou elementos de entrada e saída no sistema. As entradas e saídas dos dispositivos de memória são mapeados como sendo parte da memória de endereço.
(Lamothe, André. 2005.)

2.5 Arduino

Arduino é uma plataforma de prototipagem eletrônica baseada em hardware livre e software colaborativo e fácil de usar. Foi feito para entusiastas, artistas, designers, programadores e qualquer um que esteja interessado em criar objetos e/ou ambientes interativos. Os fundadores do Arduino são: Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino, and David Mellis.

O arduino é também um microcontrolador, conforme explicado nas seções acima, que nos permite criar uma grande variedade de projetos. O movimento DIY (*do it yourself* - "faça você mesmo", em tradução literal), conforme explica Chris Anderson em seu livro "Makers: A nova revolução industrial" se apropriou do Arduino por conta de seu baixo custo e grande gama de possibilidades, criando desde Drones até próteses robóticas.

O presente projeto visa utilizar a plataforma Arduino para construir um console de video game. Uma iniciativa parecida foi feita por James Bowman, em seu projeto "Gameduino", tendo arrecadado por volta de 40 mil dólares no site de *Crowdfunding* "Kickstarter", tendo criado uma plataforma Arduino dedicada aos games. Até o momento da escrita dessas linhas, James Bowman estava na segunda edição, com o Gameduino 2. O projeto pode ser encontrado no link a seguir:

https://www.kickstarter.com/projects/2084212109/gameduino-2-this-time-its-personal

Diversos arduinos foram lançados, cada um com especificidades próprias. Dentre os mais famosos, temos os da linha Uno e Mega.

O site oficial do Arduino possui um espaço chamado "Arduino Playground", que funciona no formato Wiki, isto é, um espaço colaborativo aonde os desenvolvedores podem postar seus projetos, códigos, tutoriais, diagramas, circuitos entre outros. Esta wiki está disponível em italiano, espanhol, francês, alemão, mandarim e português.
(Arduino Playground link: http://playground.arduino.cc/)

O Arduino possui preços acessíveis, porém no Brasil estes preços são inflados por conta dos custos de importação, frete, impostos, câmbio e demais taxas. Uma lista completa e tecnicamente detalhada dos Arduinos disponíveis, pode ser encontrada no link a seguir: https://www.sparkfun.com/arduino_guide

Graças ao Arduino, os entusiastas ganharam uma poderosa ferramenta de criação, que aliada a internet tornou possível que o conhecimento fosse compartilhado e expandido.

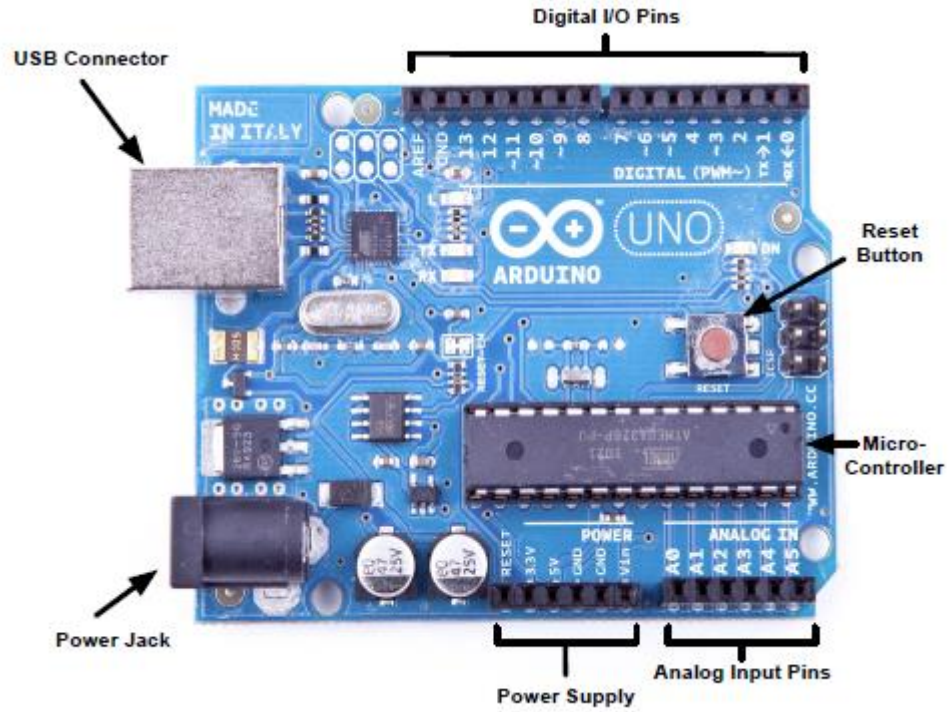


Figura 1 - Foto do Arduíno UNO, visto de cima.
Fonte: “Arduino a quick-start guide”, Maik Schmidt, p 26.

Conforme podemos ver no exemplo da Figura 1, um Arduino geralmente segue esse padrão, possuindo uma entrada USB, uma entrada para uma fonte de alimentação externa, pinagem para controle de sinais de inputs e outputs, pinagem de terra e +Vcc, assim como inputs analógicos. O coração do arduino é o seu chip microcontrolador, explicitado na Figura 1.

Cada Arduíno precisa estar conectado à uma fonte de energia. O Arduino UNO pode ser energizado por um cabo USB vindo do computador e se conectar ao USB Connector do aparelho, ou utilizar uma entrada de energia específica que conecte ao Power Jack. Lembrando que a energia utilizada não pode passar dos 20 Volts, senão poderá danificar os componentes da placa Arduino. A voltagem recomendada para a maioria dos modelos Arduino é de 6 a 12 Volts.

A conexão USB com o computador também é uma forma de despejar o código na placa do Arduino. A partir desta conexão com o computador, o Arduino pode ser configurado facilmente e de forma bem descomplicada.

O botão de Reset do Arduino funciona assim como o console de Nintendo original. Ao apertar o botão, o Arduino irá conectar o “reset pin” ao “ground” e isto acarretará em uma reinicializão do código que foi carregado no Arduino. Este procedimento pode ser muito útil para testes de código.

Um shield é uma extensão de hardware de arduino, que se acopla modularmente a ele de forma a converter sua pinagem original, traduzindo seus bits e ampliando suas capacidades, permitindo diversos usos.

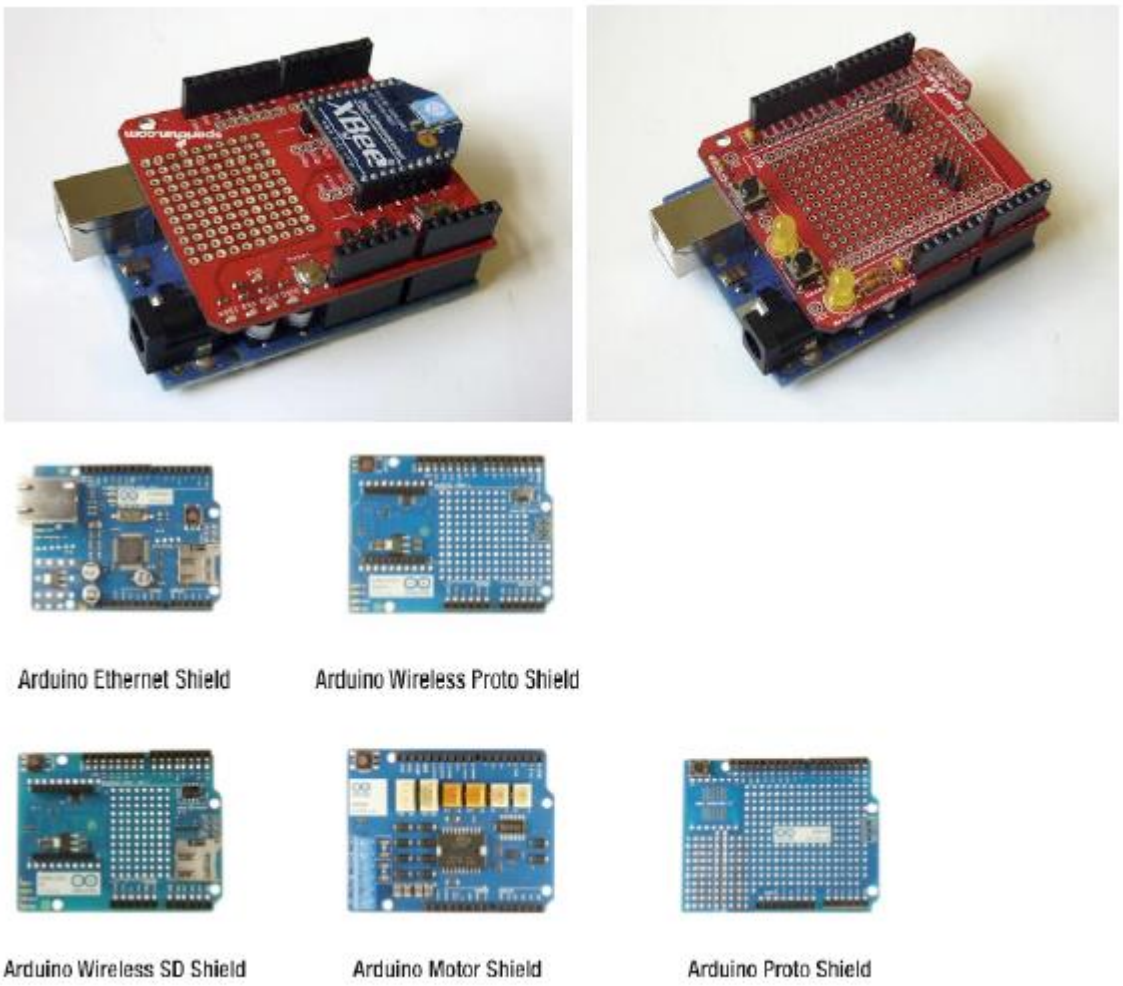


Figura 2 - Fotos de diferentes Shields.
Fonte: “Arduino and Kinect projects: Design, Build, blow their minds”, de Enrique Ramos Melgar e Ciriaco Castro Diez, p. 9.

O arduino possui uma *IDE* (Ambiente Integrado de Desenvolvimento) própria, que precisa ser baixada e instalada, com o seu ambiente de programação. Ela é composta de um editor, um compilador, um carregador e um monitor serial. A linguagem do arduino é implementada usando as linguagens C/C++ e a biblioteca *Wiring*, que é um framework de programação open source para microcontroladores. A linguagem Wiring por sua vez foi inspirada na *processing*, o que fez com que o Arduino herdasse a estrutura da linguagem processing. (Wikipedia, http://pt.wikipedia.org/wiki/Arduino)

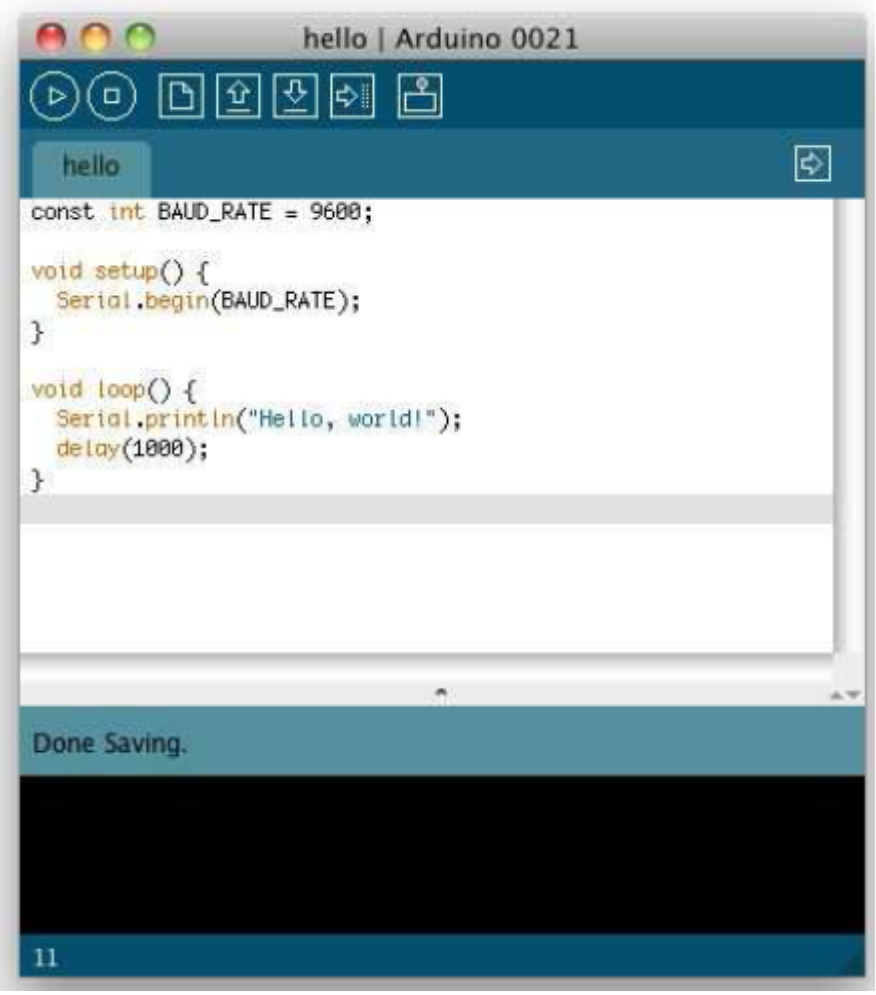


Figura 3 - Tela de compilação do IDE Arduino.

Para dar os primeiros passos com o arduino, segue-se o seguinte algoritmo:

1 | Pegar uma placa Arduino e um cabo USB

2 | Fazer o download do IDE Arduino

3 | Conectar a placa no computador

4 | Instalar os drivers (automático)

5 | Executar a aplicação do Arduino

6 | Abrir o exemplo ‘blink’

7 | Selecionar seu modelo de placa

8 | Selecionar a Porta Serial

9 | Fazer Upload do programa no Arduino

2.6 Programação

Nesta seção abordaremos tópicos relativos a programação de jogos no Arduino.

Como foi dito em outra seção, o Arduino possui muitas estruturas derivadas da linguagem C/C++. Os programas criados são chamados de *Sketchs*. Existem duas funções principais, a setup() e a loop(), que precisam estar presentes, mesmo que não sejam usadas.

- setup() - A função setup() é apenas executada uma vez, logo após o seu *Sketch* ter sido carregado, ou quando você energiza o seu Arduino. Essa função é utilizada para inicializar variáveis, pinagens, bibliotecas, etc.

- loop() - A função loop() é executada continuamente enquanto o arduino estiver sendo alimentado por energia. Ela é a essência da maior parte dos programas.

A programação no Arduino pode ser dividida em 3 grandes grupos: Estruturas, variáveis e funções. Podemos ver esses grandes grupos melhor descritos nos anexos, que contém a lista completa constituinte destes, conforme descritos no guia de referências oficial do Arduino. (Arduino homepage, 2014)

As funcionalidades do Arduino também são extensíveis com o uso de bibliotecas, podendo o programador importá-las, criá-las ou usar as que já vem junto com a *IDE* padrão. A lista completa pode ser vista na referência do site oficial do Arduino.
(Arduino, Bibliotecas. 2014)

Para os propósitos desse projeto, podemos entender a linguagem de programação do Arduino como a linguagem C combinada com as bibliotecas Arduino. A sintaxe, estrutura, operadores, fluxo de controle e funções, são derivadas de C. As bibliotecas são geralmente escritas em C++.

O Arduino possui um firmware chamado Firmata que permite o uso de outras linguagens de programação para a comunicação com sensores, motores, e outros dispositivos.

Toda linguagem de programação é baseada em 4 elementos:

- Expressões: Expressões são criadas combinando operandos e operadores.
- Statements: Todas as statements (“afirmações”) em C terminam com “;”.

- Blocos de statements: Composições de um ou mais “afirmações” que são vistas pelo compilador como sendo apenas uma.

- Blocos de funções: São blocos de código que tem como objetivo completar uma tarefa.

Todo programa pode ser dividido em 5 passos:

- Inicialização
- Inputs
- Processamento
- Outputs
- Término

O arduino por ser também implementado em C++ , possui princípios de programação orientada a objetos, especialmente em suas bibliotecas. Vejamos agora os principais fundamentos da P.O.O. (programação orientada a objetos) e como isso se aplica no Arduino.

As unidades fundamentais da POO são as classes e os objetos. O conceito de classe é uma abstração que lida com funcionalidades determinadas, e suas instâncias, que geram objetos. Classes nos permitem criar abstrações, com atributos e métodos, de forma a termos um modelo que possui estruturas em comum e é capaz de gerar versões diferentes com essas mesmas estruturas, que são os objetos. Usando uma analogia, poderíamos ter a classe “pessoa”, com os seus atributos básicos, dois olhos, cor de cabelo e etc, além de suas ações ou métodos, que são as suas interações, como andar, dirigir, e etc. Cada pessoa possui os mesmos atributos, porém a cor dos cabelos, o tamanho dos olhos e a forma como cada um dirige, são informações particulares de cada pessoa. Assim, apesar de terem uma mesma estrutura, as pessoas podem ter versões diferentes. Assim são as classes e os objetos, sendo as classes as abstrações que compartilham de estruturas em comum e os objetos as derivações dessas estruturas, podendo receber diversos valores para os seus atributos e métodos.

A utilização de classes também é um forte aliado para a modularização do código. Classes lidam geralmente com domínios específicos, de forma que as boas práticas do código limpo nos fazem designar uma classe para cada função/ação específica. Nas classes, os seus comportamentos e variáveis são inicializados por estruturas conhecidas como construtores, que são as primeiras funções a serem executadas em uma classe, realizando as declarações iniciais de seus atributos.

Em C++, existem dois tipos de arquivos principais, quando lidamos com classes, o “.cpp” e o “.h”. Os arquivos “.h” são arquivos de definição, onde declaramos as nossas classes. Implementamos as nossas classes nos arquivos “.cpp”.

Todas as classes possuem dois campos, privados e públicos. Campos públicos são informações que outras classes e trechos do código podem utilizar. Campos privados são informações relativas apenas a aquela classe.

Outro conceito importante é o de herança. Na herança, uma classe herda os seus atributos e métodos de uma outra classe, adicionando a classe original seus próprios atributos e métodos, se assim o quiser.

C++ também nos permite usar ponteiros e referências. Referências nos permitem endereçar e trabalhar diretamente com regiões de memória, acessando os bytes salvos naquela região endereçada. Ponteiros nos permitem apontar para regiões de memória, possuindo um tipo. Quando criamos vetores, estamos trabalhando com ponteiros, pois um vetor nada mais é do que um ponteiro para uma seção da memória ou um ponteiro para vários outros ponteiros. O vetor em si é apenas um ponteiro para a localização na memória para a qual o primeiro elemento do vetor é armazenado.

2.7 Engine

Nesta seção abordaremos sobre o que é uma engine, como funciona e como será aplicada no jogo em questão.

Jogos eletrônicos geralmente são programados com o apoio de uma engine. Engines são os chamados "motores de jogos", isto é, softwares que agregam diversas bibliotecas, recursos e soluções, de forma a otimizar o desenvolvimento de jogos. As engines geralmente possuem módulos específicos para otimizar os processamentos de física e computação gráfica. Temos alguns exemplos de engines famosas, como a Unity3D, Unreal, Blender Game Engine *(BGE)*, entre outras. (Game engines, 2014)

A partir das limitações do nosso projeto de Arduino, por não possuir módulos específicos para outros tipos de processamento, como uma *GPU* (Unidade de processamento gráfico), onipresente em computadores e consoles, não contamos com uma engine pronta para o desenvolvimento do nosso jogo. Entretanto teremos que criar uma engine própria, isto é, programar um código que evoque as bibliotecas do Arduino para desenhar na tela LCD os gráficos que desejamos, além de fazer uso da tela TouchScreen, ao capturar a localização matricial dos pontos X,Y que se encontra pressionada pelo jogador, simulando os comportamentos e lógica esperados pelo jogo, entre outros.

2.8 Computação gráfica e matemática

Nesta seção abordaremos a matemática aplicada à impressão de imagens em tela.

O nosso jogo será apresentado em duas dimensões. Porém, forçosamente, mesmo figuras 3D no final são convertidas para 2D, já que são apresentadas em uma tela de duas dimensões. E toda imagem nada mais é do que um conjunto de pontos que se agrupam sobre determinadas regras e cores, em uma tela.

Muito da computação gráfica utiliza a Álgebra Linear, a manipulação de matrizes, a projeção de imagens, entre outras técnicas. Esses estudos de álgebra linear foram depois implementados em *APIs* (“um conjunto de rotinas e padrões estabelecidos por um software para a utilização das suas funcionalidades por aplicativos que não pretendem envolver-se em detalhes da implementação do software, mas apenas usar seus serviços” (WIKIPEDIA, API. Disponível em: http://pt.wikipedia.org/wiki/API . Acesso em: 30 set 2014)), como a *OpenGL*. Faz-se válido portanto estudar um pouco mais esse assunto, para termos uma visão mais aprofundada sobre como são geradas as imagens que produzimos nos consoles, e sua implementação no nível de instrução de máquina.

Matematicamente, podemos calcular vetores 2D pela fórmula ($V = \sqrt{x^2 + y^2}$) e 3D pela fórmula ($V = \sqrt{x^2 + y^2 + z^2}$). Em computação gráfica temos a idéia de vetores e de pontos que descrevem geometrias. Essas geometrias depois são convertidas em matrizes para serem representadas em telas de computadores.

Todas as transformações geométricas podem ser representadas na forma de equações. As matrizes são utilizadas nessas operações porque são mais fáceis de serem usadas e entendidas do que equações algébricas.

Vetores e matrizes são parecidos com o modelo organizacional da memória dos computadores e com isso suas representações se relacionam diretamente com essas estruturas de armazenamento. As coordenadas de 2 dimensões (x,y) e as de 3 dimensão (x,y,z) são facilmente representadas com o uso de matrizes de 2 (2x2) ou 3 (3x3) elementos. Através de matrizes e sua multiplicação podemos representar todas as transformações lineares 2D e 3D. Várias transformações podem ser combinadas resultando em uma única matriz denominada matriz de transformação.

Na aritmética matricial temos um conjunto de operações. Na soma de vetores adicionamos os seus respectivos elementos (quando tiverem a mesma dimensão). Na multiplicação por um escalar, multiplicamos cada elemento pelo escalar.

Vetores ou matrizes transpostos são aqueles que trocam os elementos de suas linhas pelos elementos de suas colunas, e para multiplicarmos matrizes, o número das colunas da primeira matriz deve ser igual ao número das linhas da segunda matriz. O resultado será uma matriz com o número de linhas da primeira e o número de colunas da segunda matriz. O valor de cada elemento é dado pela soma dos produtos dos elementos das linhas da primeira matriz pelos elementos das colunas da segunda matriz de mesma ordenação. A matriz transposta torna sempre possível a multiplicação de 2 vetores, de forma que o resultado seja um número (produto interno).

Existe um conjunto de sistemas de coordenadas para nos orientar em computação gráfica. Temos circunscrito a esses sistemas de coordenadas o sistema de referência do Universo, o sistema de referência do objeto (pivô), o sistema de referência normalizado (0<=x<=1 e 0<=y<=1) e o sistema de referência do dispositivo (geralmente referentes aos sistemas de vídeo).

Muitas vezes temos que transformar um sistema de coordenadas em outro. As operações lineares de rotação e translação de objetos são chamadas de operações ou transformações de corpos rígidos dentro do jargão de computação gráfica.

Transladar significa movimentar o objeto. Transladamos um objeto somando todos os seus pontos, ao adicionarmos componentes em suas coordenadas x,y e z, dependendo se o objeto é de 2 ou 3 dimensões.
x' = x + Tx
y' = y + Ty
Z' = Z + Tz
Podemos também entender a translação à luz dos vetores, adicionando um vetor de deslocamento à posição do ponto:
P' = P + T = [x', y'] = [x,y] + [Tx Ty]
Com a notação matricial ficamos com o seguinte:
[x', y', z'] = [x, y, z] + [Tx Ty TZ]
(Conci, Aura. 2003.)

- Transformação de Escala:
Para escalonarmos (relativo a escala) um objeto, devemos multiplicar cada um de seus elementos por um fator. Isso pode ser entendido pela seguinte operação matemática:

x'= x . Sx
y' = y . Sy

- Na forma matricial:

|x y| |Sx 0|
|0 Sy|

- No espaço 3d:

|Sx 0 0|
|x' y' Z'| = |x y z| * |0 Sy 0| = |xSx ySy zSz|
|0 0 Sz|

- Transformação de rotação
Rotacionar significa "girar" um objeto. Matematicamente temos as equações abaixo.

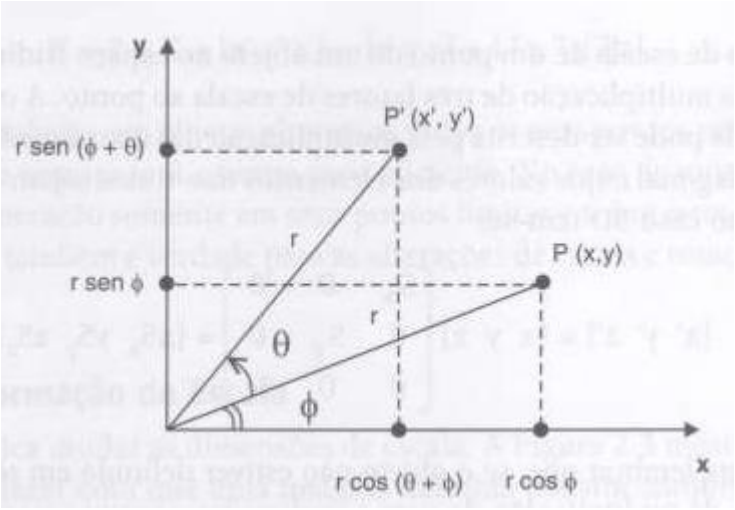


Figura 4 - Técnica de rotação do ponto P em torno da origem.
Fonte: Conci, Aura, "Computação gráfica – Geração de imagens", 2003, p. 42.

x' = r.Cos(O + o) = r * CosO * Coso - r * Sen O . Seno
y' = Y.Sen(O + o) = r * SenO * CosO + r * CosO * SenO

x' = Xcox(0) - ysen(0)
y' = yCos(0) + xSen(0)

(x,y) | Cos0 Sen0|
|-sen0 Cos0|

Toda o ferramental básico derivado da trigonometria, geometria e demais vertentes matemáticas são amplamente utilizados no desenvolvimento de jogos.

2.9 Física

Existem diversas soluções de física disponíveis no mundo dos jogos, como as tecnologias Bullet, Box2D, Havok e o renomado PhysX, que são códigos de programação que simulam efeitos físicos em elementos, tais como os de colisão, corpos rígidos, dinâmica de fluidos, efeitos de roupas, Ragdolls, entre outros.

Dentro do contexto do jogo do nosso projeto, apesar de não termos no nosso hardware Arduino um processamento que permita a utilização de tais tecnologias mais pesadas, podemos fazer uso de princípios de física newtoniana e demais modelos físicos mais rudimentares, para os cálculos de trajetórias de projéteis, efeitos de gravidade, entre outros.

2.10 Inteligência artificial

Qualquer jogo precisa de desafios. Quando não se joga diretamente contra outro jogador, dotado de inteligência, os desenvolvedores de jogos programam e simulam comportamentos inteligentes em agentes, de forma a dotar o jogo de desafio e um certo nível de imprevisibilidade. Diversas técnicas de inteligência artificial foram desenvolvidas, para resolver uma grande gama de problemas.

Os agentes dos jogos dotados de inteligência artificial possuem 3 elementos centrais:

- Percepção

Na fase de percepção estipulamos quais informações os agentes percebem no mundo a sua volta. Pode ser com base em distâncias físicas do campo de visualização dos agentes, ou com base em outras lógicas.

- Pensamento

A partir do momento que os agentes são alimentados com informações do ambiente, precisam tomar ações. Para que uma ação seja tomada, precisa haver uma estratégia inteligente para essa ação, que é decidida na fase de pensamento. Dentre as técnicas mais consagradas, temos a pré-programação de respostas de acordo com certas situações ou a utilização de algoritmos de busca para soluções ideais. Na técnica da pré-programação são utilizados máquina de estados finitos, árvores de decisão, entre outras. Na técnica de busca, utiliza-se um algoritmo de busca para descobrir uma sequência de passos que gere uma solução ideal.

- Ação

É nessa etapa que o resultado da percepção e pensamento é exteriorizado em uma ação percebida pelo jogador, de acordo com a natureza do jogo.

(Rabin, Steve. 2012).

Apesar de existirem diversas técnicas, iremos analisar agora mais profundamente apenas a técnica de máquina de estados finitos, pois é a mais coerente dentro das limitações técnicas do Arduino.

2.11 Máquina de estados finitos

A Figura 5 ajuda a ilustrar o conceito de máquina de estados finitos. Nela, um personagem de jogo parte da estratégia “esperando” para uma grande gama de possibilidades, cada uma possuindo um conjunto diferente de estratégias.

Figura 5 - Exemplo de máquina de estados.
Fonte: Usp, Lsi. Acesso em: 2014.

Cada bloco do desenho representa cada estado formado por um conjunto de instruções pré-programadas que simulam o comportamento do agente naquele estado. Cada transição de estados é captada pelo agente de acordo com a sua implementação, que escolherá para qual estado ele irá ser redirecionado.

2.12 Jogos

Jogos são uma parte universal da experiência humana e estão presentes em todas as culturas, existindo registros que datam em mais de 7 mil anos, como no caso do famoso jogo “Mancala” (Ludus lila , 2014). A principal utilidade do jogo é entreter, divertir, passar o tempo. Porém com o avanço conceitual e tecnológico, os jogos também tornaram-se capazes de melhorar a performance cerebral em áreas como a memória, raciocínio lógico, reflexos, entre outros. O jogador, desde seu primeiro contato com o jogo, pode medir a quantidade de informação retida, a partir de um procedimento chamado curva de aprendizagem.

Os consoles portáteis que inspiraram o presente trabalho surgiram no mercado no final da década de 1970. Desde então diversos modelos surgiram, dos quais podemos destacar o “Game Boy”, “PSP” e “Nintendo DS”. Os jogos destes consoles baseiam-se em todos os fundamentos expostos nas seções anteriores.

Figura 6 - Console portátil Game Boy.

Os jogos eletrônicos atuais são desenvolvidos por equipes multidisciplinares, que geralmente possuem artistas conceituais, artistas 3D e 2D, animadores, game designers, programadores, músicos, sonoplastas, entre outros.

3 Tecnologias utilizadas

Iremos utilizar no projeto a tecnologia arduino, conhecida como “Arduino Mega”, associada a um display LCD com touchscreen utilizando o shield apropriado, além das IDES, bibliotecas e demais funções que o arduino nos provê para programarmos a nossa aplicação. Vejamos abaixo cada um dos módulos empregados:

3.1 Arduino Mega

Estamos utilizando no nosso projeto o microcontrolador Arduino Mega 2560 R3 Classic.

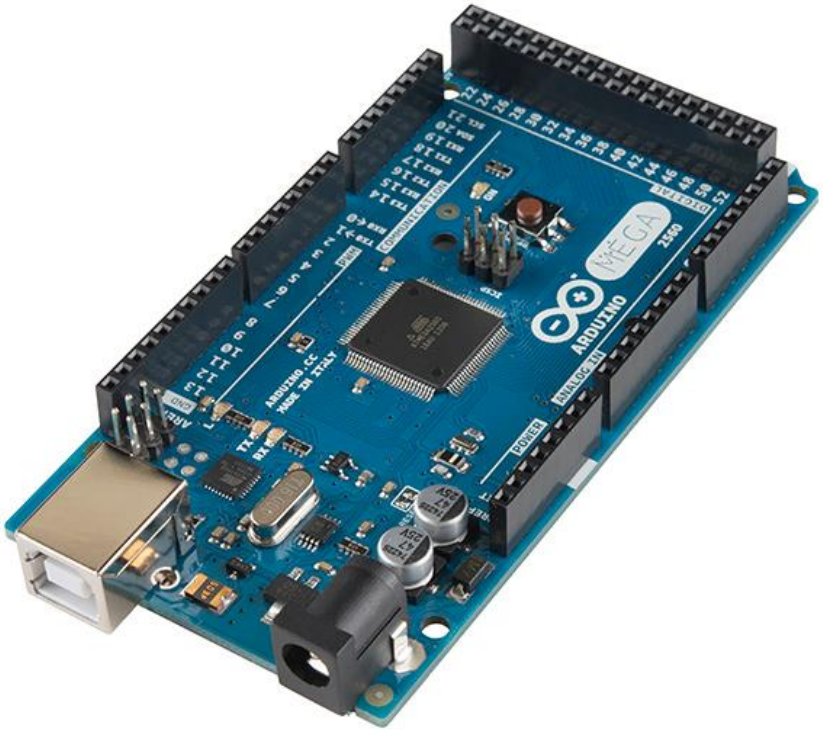


Figura 7 - Placa do Arduino MEGA.
Fonte: http://arduino.cc/en/Main/ArduinoBoardMega2560

Baseado no microcontrolador ATmega 2560, possui 54 pinos de input e output. Destes 54, 14 podem ser utilizadas como saídas *PWM*, que significa pulso com modulação, sendo uma técnica de saída analógica, a partir de entradas digitais. Além disso, possui 16 entradas analógicas, 4 *UARTs* (portas de hardware seriais), um oscilador de cristal de 16 MHz, uma conexão USB, uma entrada de energia, um *ICSP header* e um botão de reset. Ele é compatível com a maior parte dos Shields da linha Duemilanove e Diecimila.

O Arduino ainda tem pinos SDA e SCL, próximos ao pino AREF. Possui um pino IOREF, que permite adaptar o Shield à tensão do Arduino. Possui tensões de entrada de 7 e 12 volts e possui 256k de memória Flash.

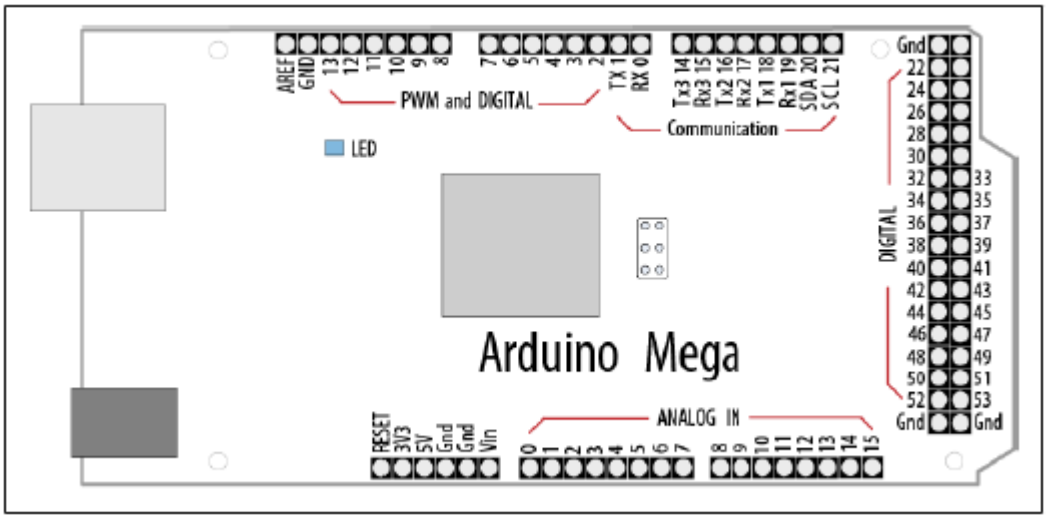


Figura 8 - Diagrama do Arduino Mega.
Fonte: Fonte “Arduino Cookbook” - Margolis, Michael

3.2 TFT LCD MEGA Shield V2.2

O shield, conforme explicado em outra seção, é o módulo que acoplamos ao Arduino para expandir as suas funcionalidades. No projeto do TuKolk estamos utilizando o TFT LCD Mega Shield V2.2 (Compatível com TFT01 e ITDB02).

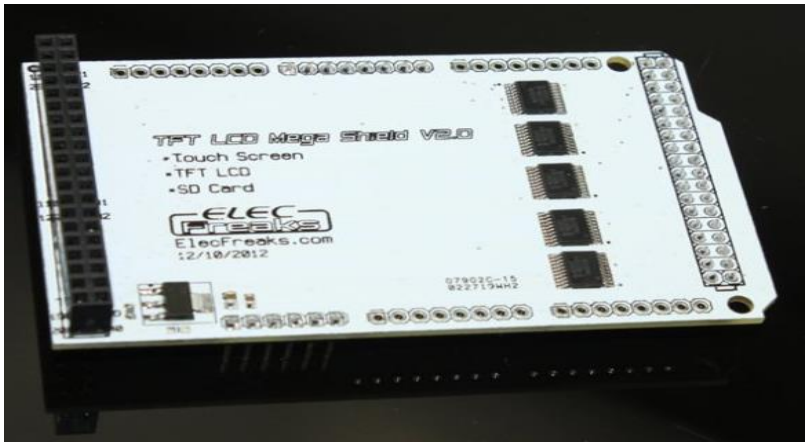


Figura 9 - Placas Shield TFT LCD para Arduino Mega
Fonte: TFT LCD MEGA Shield V2.2. Acesso em 2014.

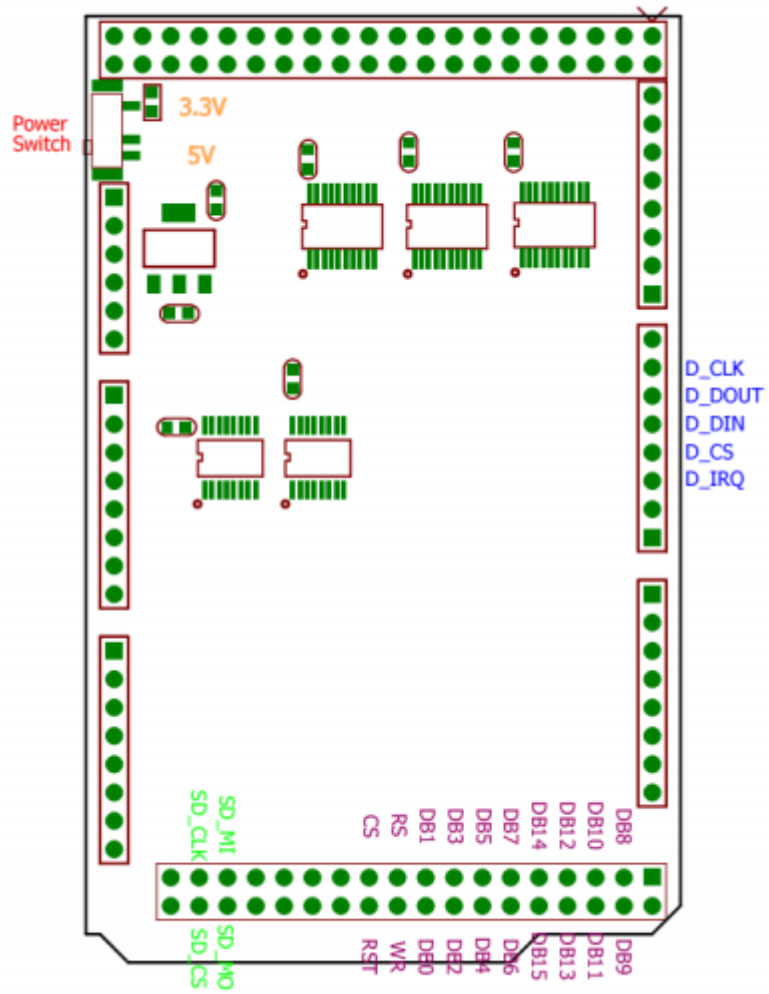


Figura 10 - Diagrama do TFT LCD Mega Shield V2.2.
Fonte: ITDB02 Arduino MEGA Shield. Acesso em 2014.

Um esquema melhor detalhado do Shield encontra-se na seção de anexos, no anexo 7.3. Esse modelo suporta tanto tensões de 3.3 V quanto de 5 V.

3.3 Display LCD / Touch Screen

Adquirimos para o projeto o Display LCD TFT01-2.4 v1.2. Nele, temos uma tela LCD com funcionalidades touchscreen, que utiliza o driver display S6D1121.

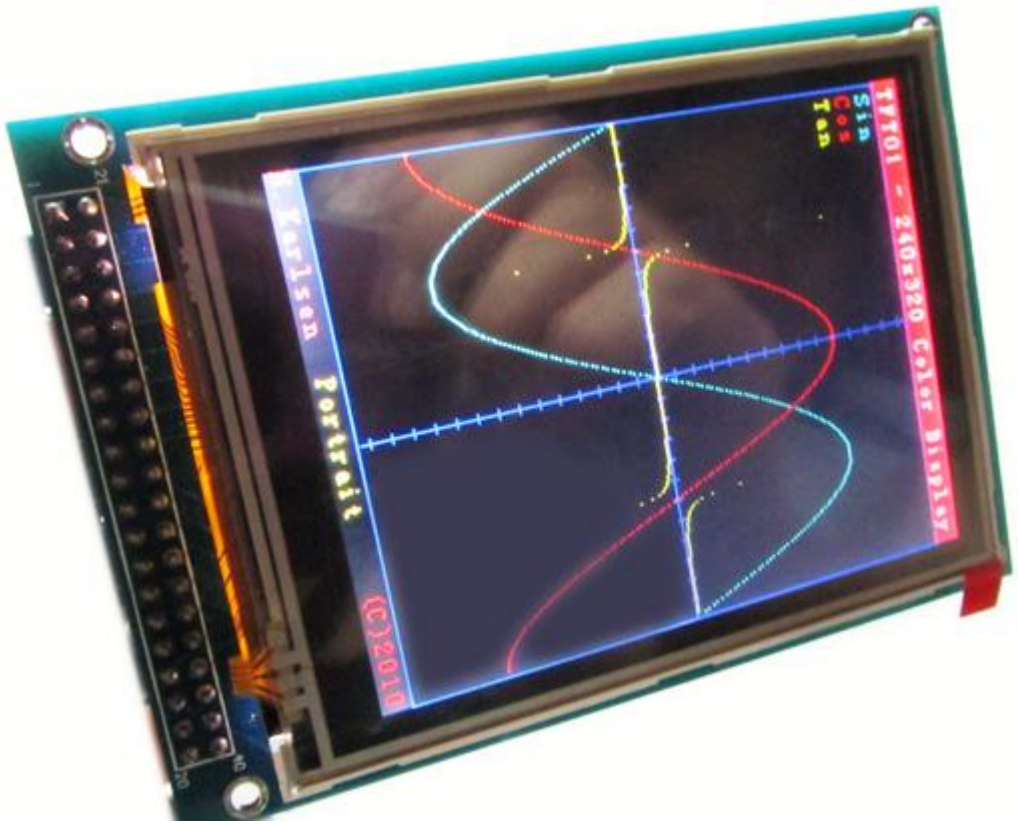


Figura 11 - Display LCD TFT01-2.4 v1.2.
Fonte: ElecFreaks, TFT01 LCD, Graphic Library. Acesso em 2014.

Possui 40 pinos, cartão SD, tela de cristal líquido QVGA TFT LCD display, resolução de 240 X 320 RGB, possibilitando 262.144 cores. Possui dimensões de 530mm por 750mm. Regulador de tensão de 2.8 V / 3.3 V, com alimentação de 5V. Suporta formato 8/16 bits RGB565, Controlador touchscreen ADS7843 e controlador LCD TFT S6D1121.

As funcionalidades de touchscreen, assim como o desenho na tela LCD são feitas via bibliotecas, que serão explicadas em outra seção.

Do ponto de vista físico, as telas LCD funcionam como isolantes elétricos, que separam materiais condutores transparentes como o *ITO(Indium tin oxide)*, de outros condutores. O corpo humano é condutor, então, quando encostamos na superfície da tela, geramos uma distorção no campo eletroestático da tela, gerando uma mudança no efeito capacitivo, formado entre o ITO, o dedo humano e o isolante de vidro. Pode-se então determinar a posição do toque, pela variação da capacitância, e essa posição é enviada a um controlador que a processa.

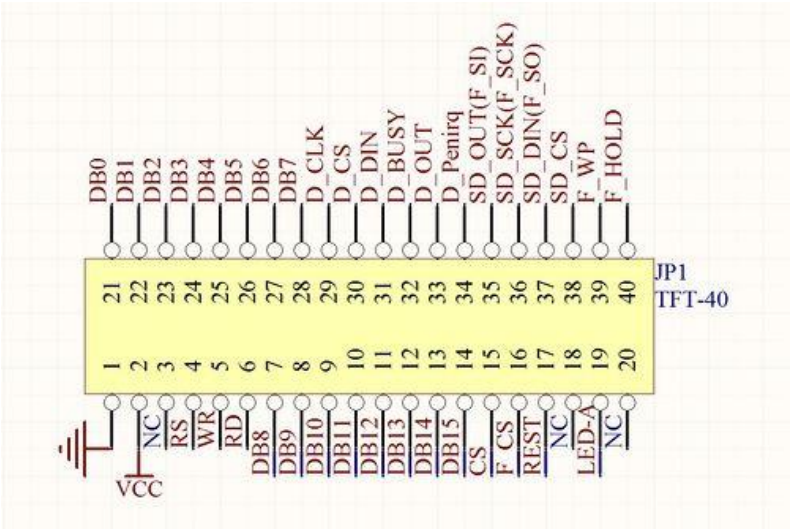


Figura 12 - Diagrama do Display LCD TFT01-2.4 v1.2.
Fonte: ElecFreaks Wiki, TFT01 LCD. Acesso em: 2014.

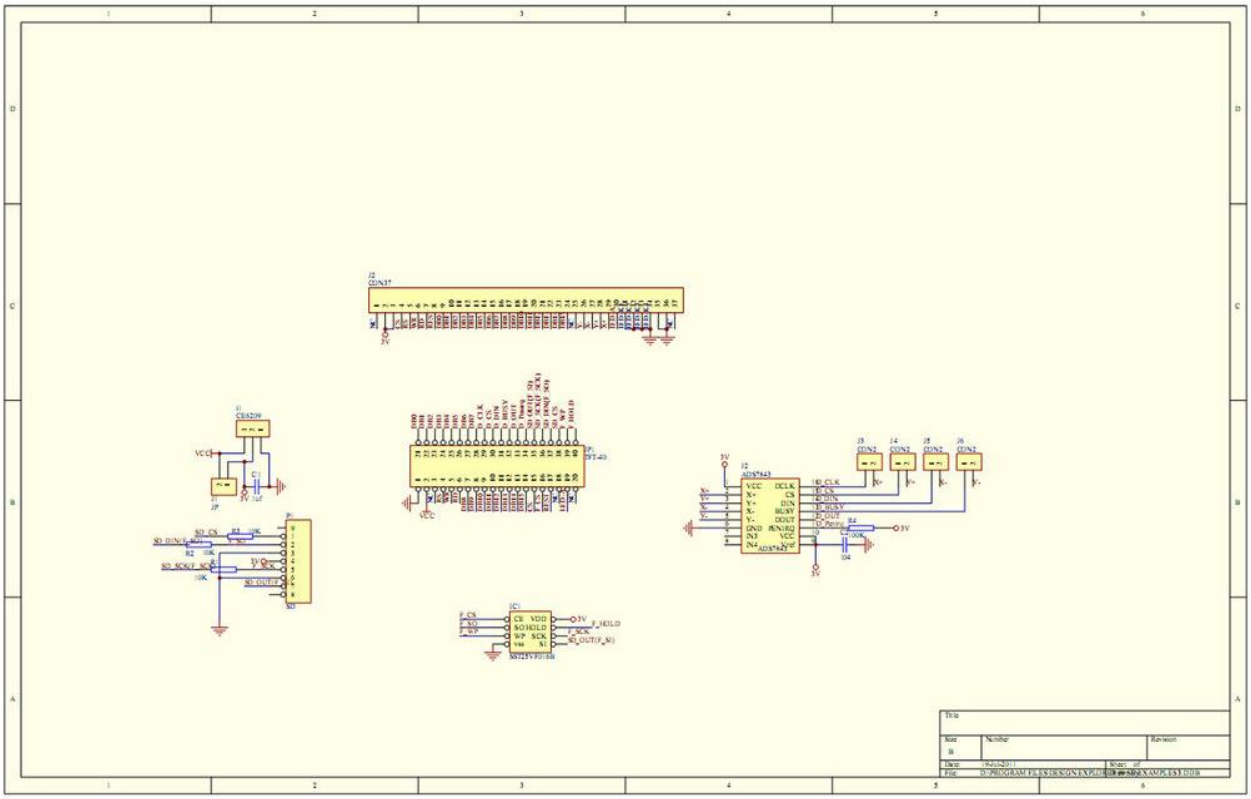


Figura 13 - Esquema LCD.
Fonte: ElecFreaks Wiki, TFT01 LCD. Acesso em: 2014.

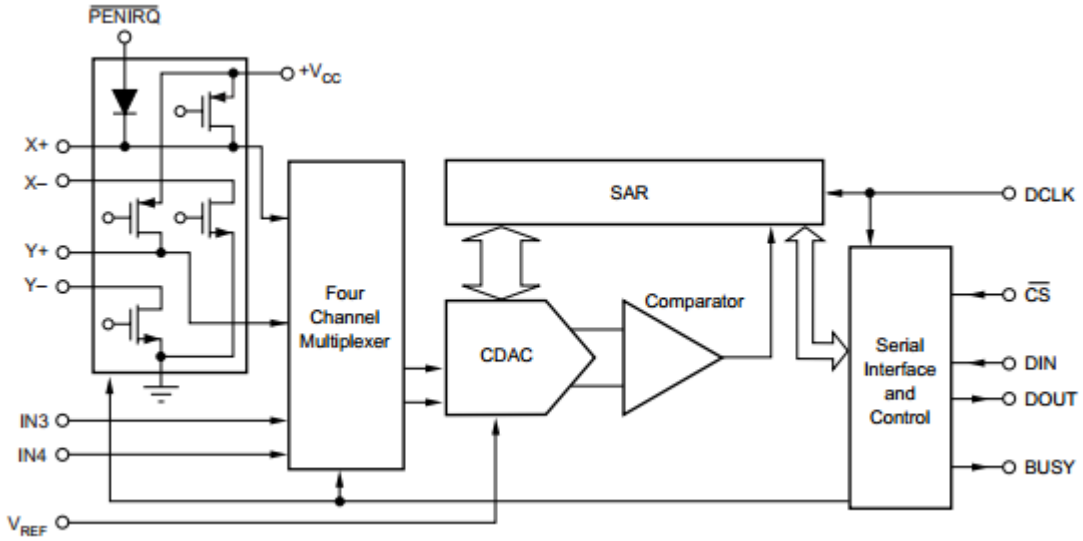
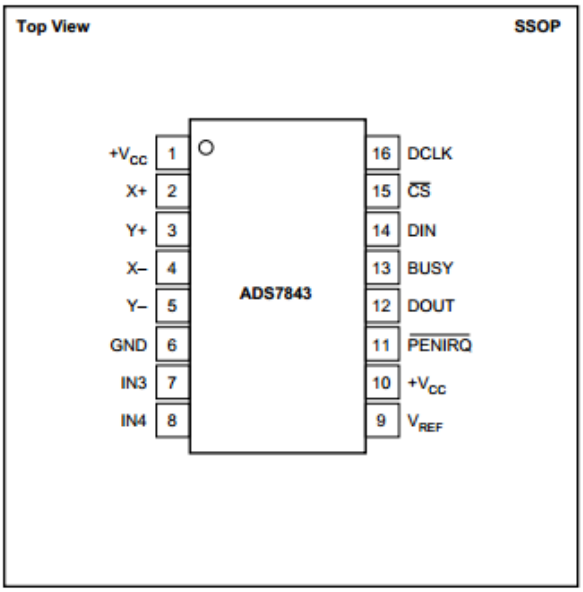


Figura 14 - Esquema do ADS7843 Touch Screen
Fonte: Texas Instruments, TFT01 LCD. Acesso em: 2014.

O ADS7843 é o circuito responsável por capturar as posições da tela touchscreen, criando esse efeito. Basicamente ele é um conversor analógico/digital com interface síncrona serial e baixa resistência.

PIN CONFIGURATION



PIN DESCRIPTION

PIN	NAME	DESCRIPTION
1	$+V_{CC}$	Power Supply, 2.7V to 5V.
2	X+	X+ Position Input. ADC input Channel 1.
3	Y+	Y+ Position Input. ADC input Channel 2.
4	X–	X– Position Input
5	Y–	Y– Position Input
6	GND	Ground
7	IN3	Auxiliary Input 1. ADC input Channel 3.
8	IN4	Auxiliary Input 2. ADC input Channel 4.
9	V_{REF}	Voltage Reference Input
10	$+V_{CC}$	Power Supply, 2.7V to 5V.
11	$\overline{PENIRQ}$	Pen Interrupt. Open anode output (requires 10kΩ to 100kΩ pull-up resistor externally).
12	DOUT	Serial Data Output. Data is shifted on the falling edge of DCLK. This output is high impedance when $\overline{CS}$ is HIGH.
13	BUSY	Busy Output. This output is high impedance when $\overline{CS}$ is HIGH.
14	DIN	Serial Data Input. If $\overline{CS}$ is LOW, data is latched on rising edge of DCLK.
15	$\overline{CS}$	Chip Select Input. Controls conversion timing and enables the serial input/output register.
16	DCLK	External Clock Input. This clock runs the SAR conversion process and synchronizes serial data I/O.

Figura 15 - Pinagem do ADS7843
Fonte: Texas Instruments, TFT01 LCD. Acesso em: 2014.

3.4 Bibliotecas

As bibliotecas são o que nos permitem utilizar as funções da tela LCD, tanto as funcionalidades de touchscreen quanto as de renderização (técnica de desenhar na tela). Utilizamos as bibliotecas UTFT e UTouch para isso. A relação completa pode ser encontrada nos anexos 7.7 e 7.8. Vejamos as principais em maior detalhe:

- UTouch:

A biblioteca UTouch é responsável pelas funções de captura e manipulação do touchscreen. Vejamos as suas funções:

UTouch(TCLK, TCS, TDIN, TDOUT, IRQ);

The main class of the interface.

Parameters:
```
TCLK:  Pin for Touch Clock (D_CLK)
TCS:   Pin for Touch Chip Select (D_CS)
TDIN:  Pin for Touch Data input (D_DIN)
TDOUT: Pin for Touch Data output (D_OUT)
IRQ:   Pin for Touch IRQ (DPenirq)
```
Usage:
```
UTouch myTouch(15,10,14,9,8); // Start an instance of the UTouch class
```

InitTouch([orientation]);

Initialize the touch screen and set display orientation. If the library is used together with UTFT the orientation should be set to the same orientation for both libraries.

Parameters:
```
orientation: <optional>
             PORTRAIT
             LANDSCAPE (default)
```
Returns:
```
Nothing
```
Usage:
```
myTouch.InitTouch();// Initialize the touch screen
```

dataAvailable();

Check to see if new data from the touch screen is waiting.

Parameters:
```
None
```
Returns:
```
Boolean: true means data is waiting, otherwise false
```
Usage:
```
check = myTouch.dataAvailable() // See if data is waiting
```

getX();

Get the x-coordinate of the last position read from the touch screen.

Parameters:
```
None
```
Returns:
```
Integer
```
Usage:
```
x = myTouch.getX(); // Get the x-coordinate
```

getY();

Get the y-coordinate of the last position read from the touch screen.

Parameters:
```
None
```
Returns:
```
Integer
```
Usage:
```
y = myTouch.getY(); // Get the y-coordinate
```

- UTFT:

A biblioteca UTFT é responsável pelas funções da tela LCD. Vejamos as suas funções:

UTFT(Model, RS, WR, CS, RST[, ALE]);

The main class constructor when using 8bit or 16bit display modules.

Parameters:
```
Model:  See the separate document for the supported display modules
RS:     Pin for Register Select
WR:     Pin for Write
CS:     Pin for Chip Select
RST:    Pin for Reset
ALE:    <optional> Only used for latched 16bit shields
        Pin for Latch signal
```

Usage:
```
UTFT myGLCD(ITDB32S,19,18,17,16); // Start an instance of the UTFT class
```

clrScr();

Clear the screen. The background-color will be set to black.

Parameters: None

Usage:
```
myGLCD.clrScr(); // Clear the screen
```

setColor(r, g, b);

Set the color to use for all draw*, fill* and print commands.

Parameters:
```
r: Red component of an RGB value (0-255)
g: Green component of an RGB value (0-255)
b: Blue component of an RGB value (0-255)
```

Usage:
```
myGLCD.setColor(0,255,255); // Set the color to cyan
```

drawRect(x1, y1, x2, y2);

Draw a rectangle between two points.

Parameters:
```
x1: x-coordinate of the start-corner
y1: y-coordinate of the start-corner
x2: x-coordinate of the end-corner
y2: y-coordinate of the end-corner
```

Usage:
```
myGLCD.drawRect(119,159,239,319); // Draw a rectangle
```

drawCircle(x, y, radius);

Draw a circle with a specified radius.

Parameters:
```
x:      x-coordinate of the center of the circle
y:      y-coordinate of the center of the circle
radius: radius of the circle in pixels
```

Usage:
```
myGLCD.drawCircle(119,159,20); // Draw a circle with a radius of 20 pixels
```

fillCircle(x, y, radius);

Draw a filled circle with a specified radius.

Parameters:
```
x:      x-coordinate of the center of the circle
y:      y-coordinate of the center of the circle
radius: radius of the circle in pixels
```

Usage:
```
myGLCD.fillCircle(119,159,10); // Draw a filled circle with a radius of 10 pixels
```

setFont(fontname);

Select font to use with print(), printNumI() and printNumF().

Parameters:
```
fontname: Name of the array containing the font you wish to use
```

Usage:
```
myGLCD.setFont(BigFont); // Select the font called BigFont
```

Notes:
```
You must declare the font-array as an external or include it in your sketch.
```

4 Desenvolvimento do projeto

Nessa seção iremos discorrer sobre o desenvolvimento do projeto, a construção física do hardware, a programação do software, problemas encontrados, desafios e as soluções desenvolvidas.

4.1 O primeiro contato

O primeiro contato com a tecnologia Arduino veio após as aulas da disciplina Tarc I ministradas pelo professor Leonardo Rocha, na Unirio. Houve uma dificuldade inicial em se conseguir os Kits, de forma que utilizamos o Kit do professor.

Após um rodízio com outros alunos, conseguimos por intermédio de um colega em comum um Arduino Uno, para prosseguirmos o desenvolvimento do nosso projeto da disciplina. Realizamos um projeto de manipulação de texto em um display, conforme mostrado na foto abaixo, utilizando a tela LCD do professor, assim como a sua fiação, protoboard e demais periféricos. A experiência com a disciplina nos motivou a aprofundar os estudos e a desenvolver um projeto final utilizando essa tecnologia.

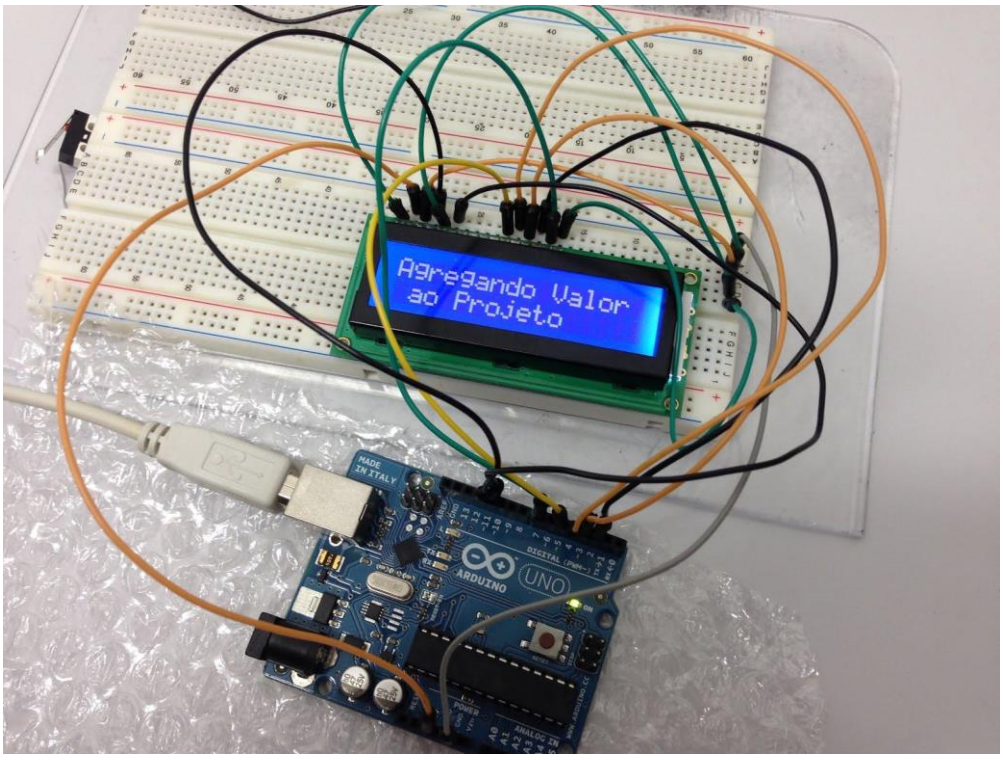


Figura 16 - Primeiras experiências com o Arduino Uno

4.2 Aventuras e novas experiências

Após a experiência com a disciplina, resolvemos então nos inscrever no projeto de conclusão de curso no semestre seguinte. Havíamos definido que iríamos desenvolver um console de videogame utilizando a tecnologia Arduino, algo até então inédito para a gente. De forma a acompanhar a tendência atual da utilização do touchscreen, decidimos comprar uma tela de Nintendo DS com essa funcionalidade. A tela, porém se revelou incompleta, possuindo apenas a tecnologia touchscreen, sem a tela LCD, que até então imaginávamos que viria agregada. Além do mais, a tela não veio com adaptadores para a sua utilização, conforme pode ser visto na a imagem da Figura 17.

Figura 17 - Tela Touch de Nintendo DS

Procuramos então formas para resolver o problema, pesquisando adaptadores que nos permitissem utilizar a pinagem correta da tela, além de termos conjecturado a hipótese de soldarmos as saídas, criando *jumpers*, entre demais soluções artesanais, que foram descartadas.

Decidimos, então, investir em outra aquisição, dessa vez que viesse agregada a tela LCD com as funcionalidades touchscreen, busca que nos levou a compra do display LCD TFT01-2.4 v1.2, melhor descrito no capítulo 3. Esta tela por sua vez se demonstrou incompatível com o Arduino Uno e as nossas pesquisas indicaram que para se encaixar periféricos ao Arduino, iríamos precisar dos chamados Shields, que já foram melhor explicados no capítulo 3. Além disso, o nosso modelo de tela não era compatível com o Arduino Uno, necessitando a compra de um outro modelo de Arduino, no caso, da linha Mega.

Resolvemos pesquisar os Shields e o Arduino Mega no Edifício Avenida Central, referência de eletrônicos na cidade do Rio de Janeiro, porém ninguém sabia nem o que significava a palavra “Arduino”, ignorância que nos surpreendeu. Depois dessa negativa nas lojas físicas e tradicionais, fomos procurar em lojas virtuais. Alguns sites internacionais possuíam diversos produtos e com preços competitivos, porém os longos prazos de entrega nos fizeram descartar essas opções.

Optamos então pelos sites nacionais, pesquisando no famoso “Mercado Livre”. Lá encontramos uma gama razoável de ofertas, porém diversas com especificações técnicas equivocadas ou ausentes, além de diversos tipos de vendedores, alguns com boa e outros com má reputação e conduta. Realizamos uma filtragem dos vendedores que considerávamos que compreendiam as nossas necessidades e que tivessem boa reputação, até que conseguimos um que possuía tanto o Shield quanto o Arduino Mega, a um custo de aproximadamente 250 reais (frete incluído).

Após termos em mãos todo o hardware necessário, houve uma posterior frustração, pois junto com o hardware não veio qualquer tipo de manual, especificações ou detalhes técnicos. O Arduino possui diversos modelos e uma infinidade de Shields, que possuem por sua vez diversas versões, de forma que achar as informações para o nosso caso específico consumiu uma parcela significativa de tempo, tanto em pesquisas em sites nacionais quanto em sites internacionais.

Tendo então finalmente encontrado as informações técnicas, o projeto pode começar. A foto abaixo demonstra um exemplo das capacidades da biblioteca compatível com o display LCD.

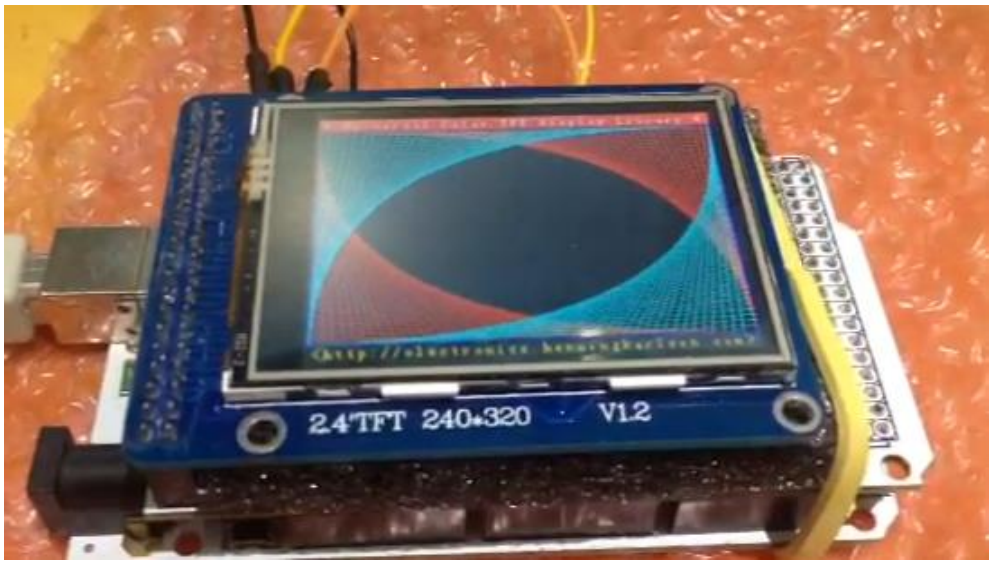


Figura 18 - Testes iniciais.

4.3 Visão geral do jogo

O projeto do jogo desenvolvido possui a tela de menu inicial para se começar o jogo, a tela de jogo e uma tela final de game over, mostrados na Figura 19.

Figura 19 - Telas do jogo.

O jogo criado é bem simples e interessante. Nele, possuímos 1 minutos (60 segundos) para fazermos pontos. Durante esse 1 minuto, diversas bolinhas de diversas cores aparecem na tela, de forma randômica (tanto a posição das bolinhas quanto as suas cores são randômicas). Ao pressionarmos a bolinha, o nosso score é atualizado, nos conferindo pontos.

Figura 20 – Gameplay e crescimento das bolinhas.

As bolinhas possuem 4 estados, o estado inicial de criação, com o tamanho de raio igual a 10 pixels, e 3 outros estados, em que as bolinhas crescem de acordo com o tempo de espera, possuindo raios iguais a, respectivamente, 15, 20 e 25 pixels. Quanto mais tempo se leva para pressionar uma bolinha, maior ela fica, e menos pontos rende. A pontuação das bolinhas é igual a 4, 3, 2 e 1 pontos, de acordo com o seu crescimento, tendo a menor bolinha a pontuação igual a 4 e a maior igual a 1.

4.4 Visão geral do código em alto nível

Vejamos agora uma visão em alto nível da programação do jogo, para depois analisarmos o desenvolvimento de seu código.

Temos 3 possíveis estados de jogo, definidos por 3 funções diferentes de Setup, que são selecionados dentro do corpo do Loop() do código de acordo com as variáveis gameState. São elas “setup()”, “setupGame()” e “setupGameOver()”.

Programamos as funções auxiliares para atualizarem o Score (updateScore()), o tempo (UpdateTimer()), o crescimento das bolinhas (grow()) e para redefinir as novas bolas e funcionalidades dos botões associados à estas bolas (waitForIt()). Fora isso, temos as variáveis de auxílio para a manipulação destes valores, notadamente as “x” e “y” para ler posições, “xPos”, “yPos” e “rad” para definir os valores dos círculos, “r”, “g” e “b” para definir a cor dos círculos, “score” e “highScore” para a pontuação, “timer”, “timerTotal” e “timeLeft” para o tempo e “gameState” para os estados do jogo. O código completo pode ser conferido abaixo, e posteriormente cada seção dele será melhor detalhada, juntamente com as explicações relativas ao seu desenvolvimento.

4.5 O código completo

Nesta seção o código será todo disponibilizado, de forma a ajudar o entendimento das seções seguintes. Ele foi dividido para fins de explicação, mas o código original segue exatamente a ordem demonstrada abaixo.

4.51 Importação de bibliotecas, declaração de variáveis e de fontes.

```
#include <UTFT.h>
#include <UTouch.h>

UTFT    myGLCD(TFT01_24R2,38,39,40,41);
UTouch  myTouch( 6, 5, 4, 3, 2);

// fontes declaradas
extern uint8_t Dingbats1_XL[];
extern uint8_t BigFont[];
extern uint8_t SmallFont[];
extern uint8_t swiss721_outline[];
extern uint8_t arial_bold[];
extern uint8_t UbuntuBold[];

int x, y; //touch
int xPos, yPos, rad; //circle values
int r,g,b; //circle color

int score=0;
int highScore=0;

int timer=0;
int timerTotal=0;
int timeLeft=60;
int gameState=0;
```

4.52 Funções auxiliares

- updateScore()

```
/*************************
**   Custom functions   **
*************************/

void updateScore(int radius)
{//rad vai de 10 a 25
  score+=4-(radius-10)/5;
  myGLCD.setColor(VGA_WHITE);
  myGLCD.print((String)score, 100, 0);
}
```

- updateTimer()

```
void updateTimer()
{
  timeLeft = 60-(timerTotal/10);
  myGLCD.setColor(VGA_BLACK);
  myGLCD.fillRect(288, 0, 319, 15);
  myGLCD.setColor(VGA_RED);
  myGLCD.print((String)timeLeft, 288, 0);
  if(timeLeft==0){
    gameState=0;
    if(score>highScore)
      highScore=score;
    setupGameOver();
  }
}
```

- grow()

```
void grow()
{
  if(rad<25){
    rad+=5;
    myGLCD.setColor(r,g,b);
    myGLCD.fillCircle(xPos,yPos,rad);
    myGLCD.setColor(VGA_WHITE);
    myGLCD.drawCircle(xPos,yPos,rad);
  }
}
```

- waitForIt()

```
void waitForIt()
{
  myGLCD.setColor(VGA_BLACK);
  myGLCD.fillCircle(xPos,yPos,rad+2);
  //myGLCD.drawCircle(xPos,yPos,rad);

  //enquanto houver toque, esperar...
  while (myTouch.dataAvailable())
    myTouch.read();

  //desenha novo circulo
  rad=10;
  //posicao(x,y) tem uma distancia segura da borda
  xPos = random(25, 295);
  yPos = random(16+25, 240-25);
  r = random(0,255);
  g = random(0,255);
  b = random(0,255);
  myGLCD.setColor(r,g,b);
  myGLCD.fillCircle(xPos,yPos,rad);
  myGLCD.setColor(VGA_WHITE);
  myGLCD.drawCircle(xPos,yPos,rad);
}
```

4.53 Funções principais - Setups.

- setup()

```
/*************************
**  Required functions  **
*************************/

void setup()
{
  myGLCD.InitLCD();
  myTouch.InitTouch();
  myGLCD.clrScr();

  myGLCD.setBackColor(0, 0, 0);

  myGLCD.setColor(VGA_GREEN);
  myGLCD.setFont(UbuntuBold);
  myGLCD.print("TuKolk", CENTER, 30);

  myGLCD.setColor(VGA_GREEN);
  myGLCD.setFont(swiss721_outline);
  myGLCD.print("Projeto Arduino", CENTER, 70);

  myGLCD.setColor(VGA_WHITE);
  myGLCD.setFont(Dingbats1_XL);
  myGLCD.print("b", RIGHT, 70);

  myGLCD.setColor(VGA_AQUA);
  myGLCD.setFont(BigFont);
  myGLCD.print("Pressione a Tela", CENTER, 144);

  myGLCD.setColor(VGA_RED);
  myGLCD.setFont(arial_bold);
  myGLCD.print("Andre", LEFT, 220);
  myGLCD.print("&", CENTER, 220);
  myGLCD.print("Fabricio", RIGHT, 220);

  gameState=1;
}
```

- setupGame()

```
void setupGame()
{
// Initial setup
  score=0;
  timer=0;
  timerTotal=0;
  timeLeft=60;
  gameState=0;

  myGLCD.clrScr();
  myTouch.setPrecision(PREC_MEDIUM);
  myGLCD.setFont(Dingbats1_XL);
  myGLCD.print("W",250, 0);
  myGLCD.setFont(BigFont);
  myGLCD.setBackColor (0, 0, 0);
  myGLCD.print("Score:", LEFT, 0);
  myGLCD.setColor(VGA_RED);
  myGLCD.print((String)timeLeft, 288, 0);

  xPos = random(25, 295);
  yPos = random(16+25, 240-25);
  rad = 10;
  myGLCD.setColor(VGA_WHITE);
  myGLCD.drawCircle(xPos,yPos,rad);

  gameState=2;
}
```

- setupGameOver()

```
void setupGameOver()
{
  myGLCD.setColor(VGA_WHITE);
  myGLCD.print("HighScore= "+(String)highScore, 48, 110);
  myGLCD.setBackColor (0, 0, 255);
  myGLCD.setColor(0, 0, 255);
  myGLCD.fillRoundRect (10, 130, 150, 180);
  myGLCD.setColor(255, 255, 255);
  myGLCD.drawRoundRect (10, 130, 150, 180);
  myGLCD.print("Reset", 40, 147);
  myGLCD.setColor(0, 0, 255);
  myGLCD.fillRoundRect (160, 130, 300, 180);
  myGLCD.setColor(255, 255, 255);
  myGLCD.drawRoundRect (160, 130, 300, 180);
  myGLCD.print("Menu", 190, 147);
  myGLCD.setBackColor (0, 0, 0);

  gameState=3;
}
```

4.54 Funções principais - Loop com game states

- game State 1

```
void loop()
{
  while(gameState==1) //Menu
  {
    delay(500);
    myGLCD.setColor(VGA_BLACK);
    myGLCD.setFont(BigFont);
    myGLCD.print("Pressione a Tela", CENTER, 144);
    delay(500);
    myGLCD.setColor(VGA_AQUA);
    myGLCD.setFont(BigFont);
    myGLCD.print("Pressione a Tela", CENTER, 144);
    if (myTouch.dataAvailable())
    {
      myTouch.read();
      gameState=0;
      setupGame();
    }
  }
```

- game State 2

```
while(gameState==2) //Game
{
  if (myTouch.dataAvailable())
  {
    myTouch.read();
    x=myTouch.getX();
    y=240-myTouch.getY();

    if ((y>=yPos-rad) && (y<=yPos+rad))
    {
      if ((x>=xPos-rad) && (x<=xPos+rad))
      {
        int lastRad = rad;
        waitForIt();
        updateScore(lastRad);
        return;
      }
    }
  }
  delay(100);
  ++timer;//10 iterações é um segundo
  ++timerTotal;
  if(timer>5){
    grow();
    timer=0;
  }
  if(timerTotal%10==0){
    updateTimer();
  }
}
```

- game State 3

```
  while(gameState==3) //ResetGame
  {
    if (myTouch.dataAvailable())
    {
      myTouch.read();
      x=myTouch.getX();
      y=240-myTouch.getY();
      if ((y>=130) && (y<=180))
      {
        if ((x>=10) && (x<=150))//Botão Reset
        {
          setupGame();
        }
        if ((x>=160) && (x<=300))//Botão Menu
        {
           setup();
        }
      }
    }
  }
}
```

4.6 Etapas do desenvolvimento do código

Após termos encontrado as bibliotecas corretas, iniciamos o estudo das libraries da UTFT e UTouch, para entendermos como funcionam as suas funções pré-definidas.

Abrimos uma *sketch* nova (arquivo do arduino) para testar as funções vistas nas libraries de exemplo. Estudamos especialmente as funções drawCircle() - que desenha círculos a partir da edição de seus parâmetros X, Y e de raio - e setColor() - que muda a cor de toda a renderização da tela.

Vejamos agora modularmente cada um dos elementos constituintes do jogo.

4.7 Menu

Para se criar o menu inicial do jogo, a primeira etapa foi testar as fontes de texto do site citado nas referências, compatíveis com a biblioteca UTFT, como as funções setFont() e print(). Para se utilizar uma fonte de texto, precisa-se adicioná-la ao Sketch, utilizando a função “extern uint8_t arial_bold[];”. Cada fonte diferente possui um tamanho X e Y de pixels, além de definições de posição X, Y da tela, escritas pela função Print().

A Figura 21 ilustra o menu de nosso jogo, de forma a ajudar a compreensão de cada um de seus elementos:

Figura 21 - Menu do jogo TuKolk.

Adicionamos então um efeito de “pisca pisca” no texto “pressiona a tela”, do centro do menu. Para se criar o efeito alterna-se entre as cores preta e branca, da mesma fonte. Usa-se a função delay(), que recebe o tempo como parâmetro, em milisegundos para se criar esse efeito.

Sabendo que o projeto possui vários estados de telas de jogo, criamos uma variável gameState para definir a situação atual dele.

O menu recebe o valor da variável gameState =1, o que garante com que a tela de menu seja sempre a primeira a ser vista, ao se inicializar o Arduino. Cada tela de jogo possui uma função setup() própria, e assim a variável gameState nos direciona para setups diferentes.

Para se iniciar o jogo, basta tocar na tela. Para isso, usa-se o teste condicional da função myTouch.dataAvailable(). Essa função verifica se houve toque na tela para começar o jogo. Chama então a função setupGame(), e inicializa as configurações iniciais do jogo, que serão analisadas mais a frente.

4.8 O jogo

Após termos desenvolvido o menu, fomos para o desenvolvimento do jogo propriamente dito. Iniciamos o desenvolvimento programando o Score, isto é, a pontuação do jogo, que utiliza a função print() para imprimir os caracteres alfanuméricos na tela.

Abrimos então o sketch de exemplo da UTouch, chamado “UTouch_ButtonTest”, que auxilia a criação de caixas retangulares com funcionalidades de botão, onde definimos qual seria a área clicável do botão. A Figura 22 ilustra esse momento:

Figura 22 - Teste de botão utilizando a biblioteca UTouch.

Testamos então as funções drawCircle() - que imprime o perímetro de um círculo - e fillCircle() - que preenche o círculo desenhado. Primeiro preenchemos um círculo, com a função fillCircle(), e em seguida o imprimimos com a função drawCircle().
A Figura 23 ilustra esse teste.

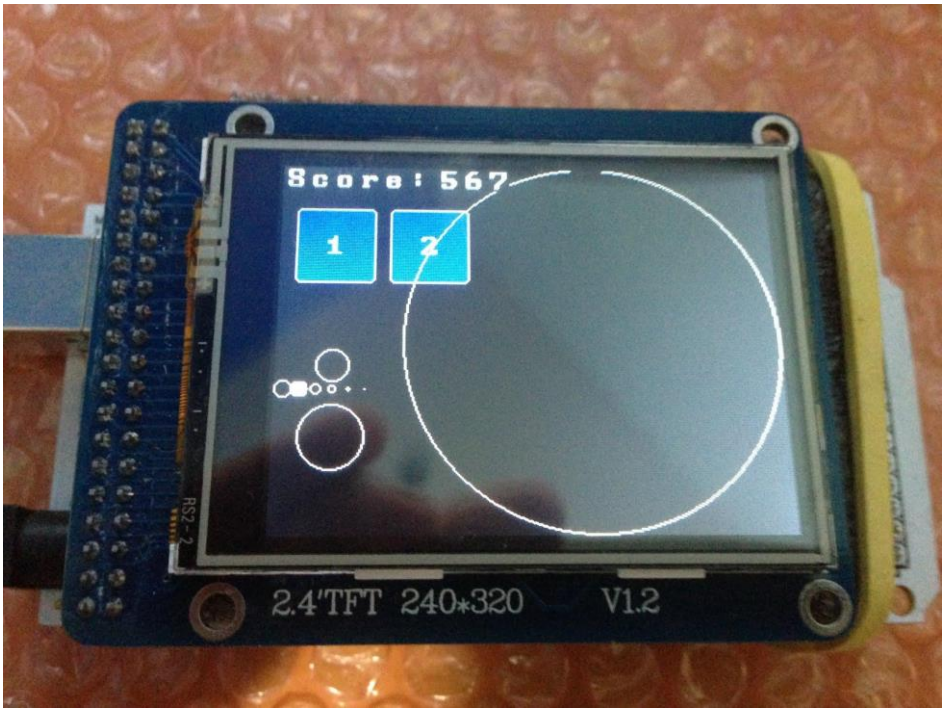


Figura 23- Teste da função drawCircle().

Constatamos então que não haviam mais exemplos úteis nas bibliotecas (Libraries), de forma que daqui para frente as funções teriam que ser desenvolvidas do zero.

Foi criada então uma função nomeada grow(), que é chamada a cada ½ segundo e faz o círculo crescer a partir de seu raio em 10 pixels, não ultrapassando 25 pixels de raio. Mapeou-se a posição do círculo criado para que a sua área fosse transformada em botão. Não criou-se uma função específica para isso pois esta tarefa foi implementada dentro do loop().

Com a criação de círculos, cada vez que um círculo é selecionado com a função de toque, são chamadas duas funções cruciais, que são a waitForIt() e updateScore(). Vejamos cada uma delas abaixo.

- **waitForIt()** é responsável por apagar o círculo selecionado ao pintá-lo da cor do fundo da tela, criando um outro círculo em uma posição randômica que não extrapole as laterais da resolução do LCD. Com estas mesmas posições, renova também a área de toque anterior.
- **updateScore()** é responsável por atualizar o score durante o jogo, de acordo com o tamanho do raio do círculo. Quanto maior for o círculo, menor será a sua pontuação, pois mais fácil será selecioná-lo. Existem 4 variações de círculos, com raios variando entre os valores de 10, 15, 20 e 25 pixels. A expressão que define a pontuação é esta: pontuação+=4-(raio-10)/5.

A Figura 24 captura o momento da implementação do Score.

Figura 24 - Processo de implementação de Score.

O próximo passo foi criar uma contagem regressiva para o nosso jogo. Como não existe uma função pré-definida de cálculo de tempo, foi necessário usar a função delay(1000), que atrasa a execução do programa em 1000 milisegundos, e criar uma váriavel “timerTotal”, para armazenar 1 unidade a mais para cada segundo.

Porém, a função delay(1000) faz com que se paralise por 1 segundo quaisquer outras atividades. Ou seja, o toque só seria reconhecido após 1 segundo, o que atrapalharia a jogabilidade. Para não atrapalhar a função de toque, criamos um método que divide a função delay() em pequenas partes, para que o toque possa ser reconhecido dentre estas pequenas divisões de tempo. Ou seja, o delay(1000) que antes era de 1 segundo, passou a ser delay(100) de 0,1 segundo e a variável chamada “timerTotal” passou a armazenar 10 unidades para cada 1 segundo. Logo, a expressão final para a contagem regressiva ficou: timeLeft = 60 - (timerTotal/10). Assim que o tempo chega a zero, é chamada a função setupGameOver().

Na Figura 25 podemos ver o timer implementado:

Figura 25 - Processo de implementação de Timer.

4.9 Game Over:

Por fim, temos a tela de game over que sobrepõe a tela do jogo com a impressão do highscore e mais 2 botões, “reset” e “menu”, acima da tela de jogo. Se a pontuação for maior que a pontuação anterior, este será o novo high score do jogo. Como pode ser visualizado na Figura 26:

Figura 26 - Processo de implementação de Score.

5 Conclusão

O presente trabalho tem como objetivo realizar um estudo, teórico e prático, sobre os consoles de video games e seus jogos. Com a monografia pronta e o projeto implementado, acreditamos que os objetivos foram alcançados.

5.1 Limitações:

Passamos por diversas dificuldades para podermos começar a realizar o projeto, assim como muitas dificuldades durante o desenvolvimento do mesmo. Por ser um projeto pouco usual, dentro do seu contexto, as informações sobre jogos no universo do Arduino são escassas, para não dizer inexistentes. Portanto, a persistência empregada durante todo o projeto talvez seja uma das maiores lições que essa experiência nos traz, entendida aqui como a habilidade de não desanimar de um projeto frente as dificuldades encontradas, acreditando que a solução virá a partir do momento que se adicionar trabalho, pesquisa e disciplina à essa persistência. Fora isso, a oportunidade de mexer em uma tecnologia em ascenção, o Arduino, que nos permite desenvolver múltiplos projetos em diversas áreas, já é em si uma habilidade valiosa e muito relevante para o mercado, em um mundo cada vez mais automatizado. Vale também citar a experiência com o desenvolvimento de uma aplicação touchscreen, presente nos smarthphones e na criação e manipulação de gráficos em uma tela LCD, também presente hoje em dia em qualquer interface.

5.1 Contribuições:

Falando sobre o mercado de jogos em particular, este também se apresenta como um mercado em ascenção, com o crescimento e surgimento de diversas empresas. Esses conhecimentos portanto, vem muito a agregar para quem tem interesses profissionais com os jogos, pois desvendam a caixa preta que são os consoles, abrangendo o estudo além do alto nível da programação, até o baixo nível dos bits, portas lógicas, componentes eletrônicos e leis da eletricidade.

O trabalho deixa um estudo teórico e prático para quem tiver interesse na área, que poderá partir de um conhecimento acumulado significativo para expandir as possibilidades e melhor aproveitar as capacidades e poder do Arduino ou de outras tecnologias que venham a surgir.

Portanto, os objetivos foram alcançados e esperamos que a partir deste trabalho outros estudantes possam encontrar uma ajuda para alcançar os seus.

5.1 Limitações:

Deste estudo, outros trabalhos podem ser derivados. Destacamos a possibilidade de se utilizar o jogo criado como uma ferramenta para se pesquisar as curvas de aprendizagem, estudos relativos a reflexos, coordenação motora, cognição e níveis de atenção.

6 Referências

- Lamothe, André. **The Black Art of Video Game Console Design**. Sams Publishing. Indianápolis: Indiana, December 2005. cap. 7, p. 380.

- Schmidt, Maik - **Arduino a quick-start guide**. Pragmatic Bookshelf; 1 edition (January 25, 2011)

- Anderson, Chris - **Makers: A nova revolução industrial**. Crown Business (April 8, 2014)

- Conci, Aura. **Computação gráfica – Geração de imagens**. Campus, 2003. cap. 2, p. 38. (Série, nº 7)

- Melgar, Enrique Ramos e Diez, Ciriaco Castro - **Arduino and Kinect projects: Design, Build, blow their minds.** Apress; 1 edition (April 17, 2012)

- Rabin, Steve - **Introdução ao desenvolvimento de games.** Cengage Learning, 2012

- Margolis, Michael - **Arduino Cookbook.** O'Reilly Media; Second Edition edition (December 30, 2011)

- Olsson, Tony - **Arduino Wearables.** Apress; 1 edition (July 3, 2012)

- Di Justo, Patrick e Gertz, Emily - **Atmospheric Monitoring with Arduino.** Maker Media, Inc; 1 edition (December 3, 2012)

- Evans, Brian - **Beginning Arduino programming.** Apress; 1 edition (October 16, 2011)

- Noble, Joshua - **Programming Interactivity.** O'Reilly Media; Second Edition edition (January 30, 2012)

- Purdum, Jack - **Beginning C for Arduino.** Apress; 1 edition (December 8, 2012)

- Trabalho "**Computação gráfica**" de Fabrício Kolk Carvalho e Beatriz Goulart, para a matéria de Álgebra Linear da professora Beatriz Malajovich, do departamento de matemática da Unirio, 2011.

- **Arduino, Bibliotecas**. Acesso em: 30 set 2014. Disponível em: http://arduino.cc/en/Reference/Libraries

- **Ludus Lila**. Dezembro de 2014. Disponível em: http://luduslila.wordpress.com/2012/04/07/mancala-a-origem-de-todos-os-jogos/

- **Usp, Lsi.** Dezembro de 2014. Disponível em: http://www.lsi.usp.br/~rponeves/diss/dissert_files/image025.jpg

- **ElecFreaks, TFT01 LCD Graphic Libray.** Dezembro de 2014. Disponível em: http://www.elecfreaks.com/1249.html

- **ElecFreaks, TFT01 LCD.** Dezembro de 2014. Disponível em: http://www.elecfreaks.com/wiki/index.php?title=2.4%22_TFT_LCD:_TFT01-2.4

- **Texas Instruments, TFT01 LCD.** Dezembro de 2014. Disponível em: http://www.ti.com/lit/ds/symlink/ads7843.pdf

- **Tela de nintendo DS.** Outubro de 2014. Disponível em: http://www.labdegaragem.org/loja/nintendo-ds-touch-screen.html

- **Garagino.** Outubro de 2014. Disponível em: http://www.labdegaragem.org/loja/kit-triplo-garagino-rev-1-conversor-usb-serial.html

- **Conector de Nintendo Ds.** Outubro de 2014. Disponível em: http://www.labdegaragem.org/loja/nintendo-ds-touch-screen-connector-breakout.html

- **UTFT.** Outubro de 2014. Disponível em: http://www.henningkarlsen.com/electronics/library.php?id=51

- **UTouch.** Outubro de 2014. Disponível em: http://www.henningkarlsen.com/electronics/library.php?id=55

- **Aprendendo a programar em Arduino.** Outubro de 2014. Disponível em: http://pt.slideshare.net/Miojex360/apostila-para-programar-arduino

- **Arduino Language reference.** Outubro de 2014. Disponível em: http://arduino.cc/en/Reference/HomePage

- **UTFT Fonts.** Outubro de 2014. Disponível em: http://www.henningkarlsen.com/electronics/r_fonts.php

- **ITDB02 Arduino MEGA Shield.** Outubro de 2014. Disponível em: http://wiki.iteadstudio.com/ITDB02_Arduino_MEGA_Shield

- **ElecFreaks 2.4" TFT LCD: TFT01-2.4.** Outubro de 2014. Disponível em: http://www.elecfreaks.com/wiki/index.php?title=2.4%22_TFT_LCD:_TFT01-2.4

- **GameDuino.** Outubro de 2014. Disponível em: https://www.kickstarter.com/projects/2084212109/gameduino-an-arduino-game-adapter/posts

- **TFT01 LCD TFT Display.** Outubro de 2014. Disponível em: http://www.electrodragon.com/w/index.php?title=TFT01_LCD_TFT_Display

- **LCD TFT01 Arduino Mega Shield v2.0 SHD10.** Outubro de 2014. Disponível em: http://www.elecfreaks.com/store/lcd-tft01-arduino-mega-shield-v10-p-214.html

7 Anexos

7.1 Diagrama do Arduino Mega

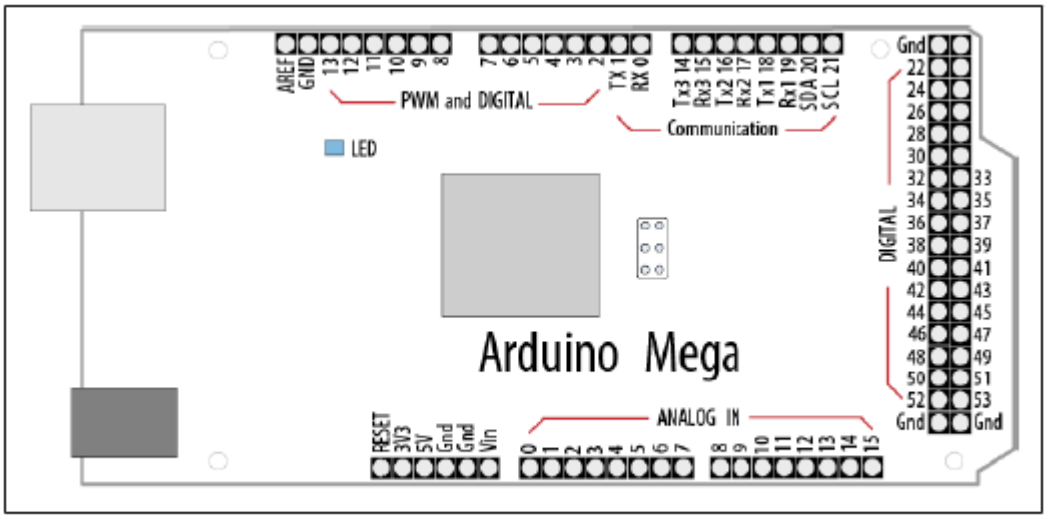
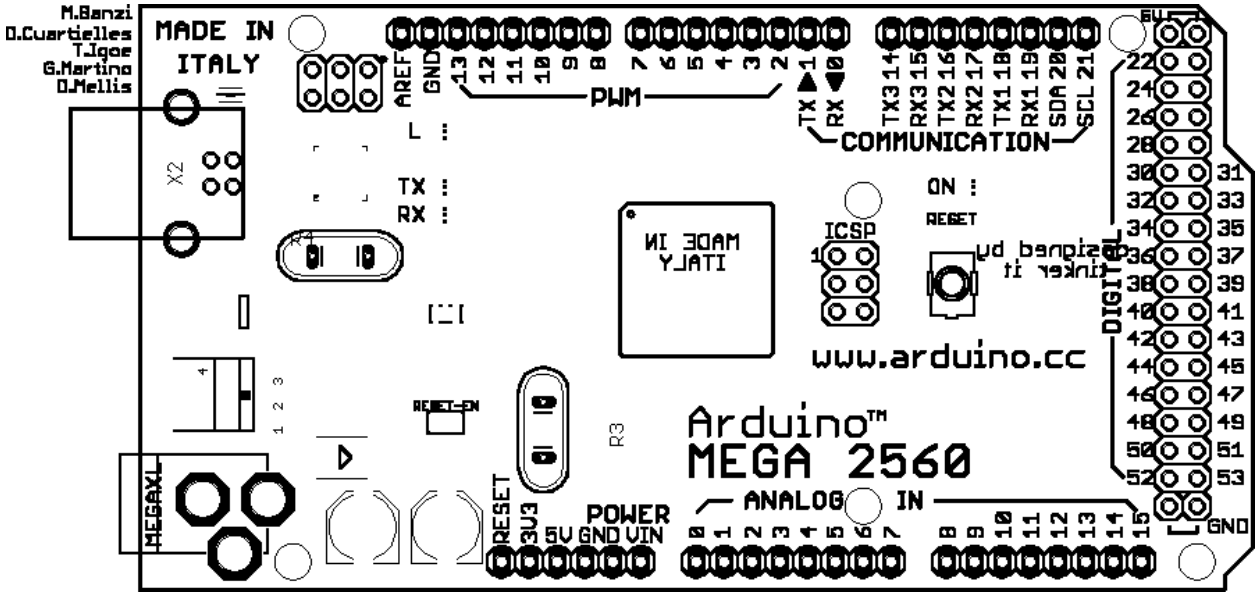


Figure 5-2. Arduino Mega board

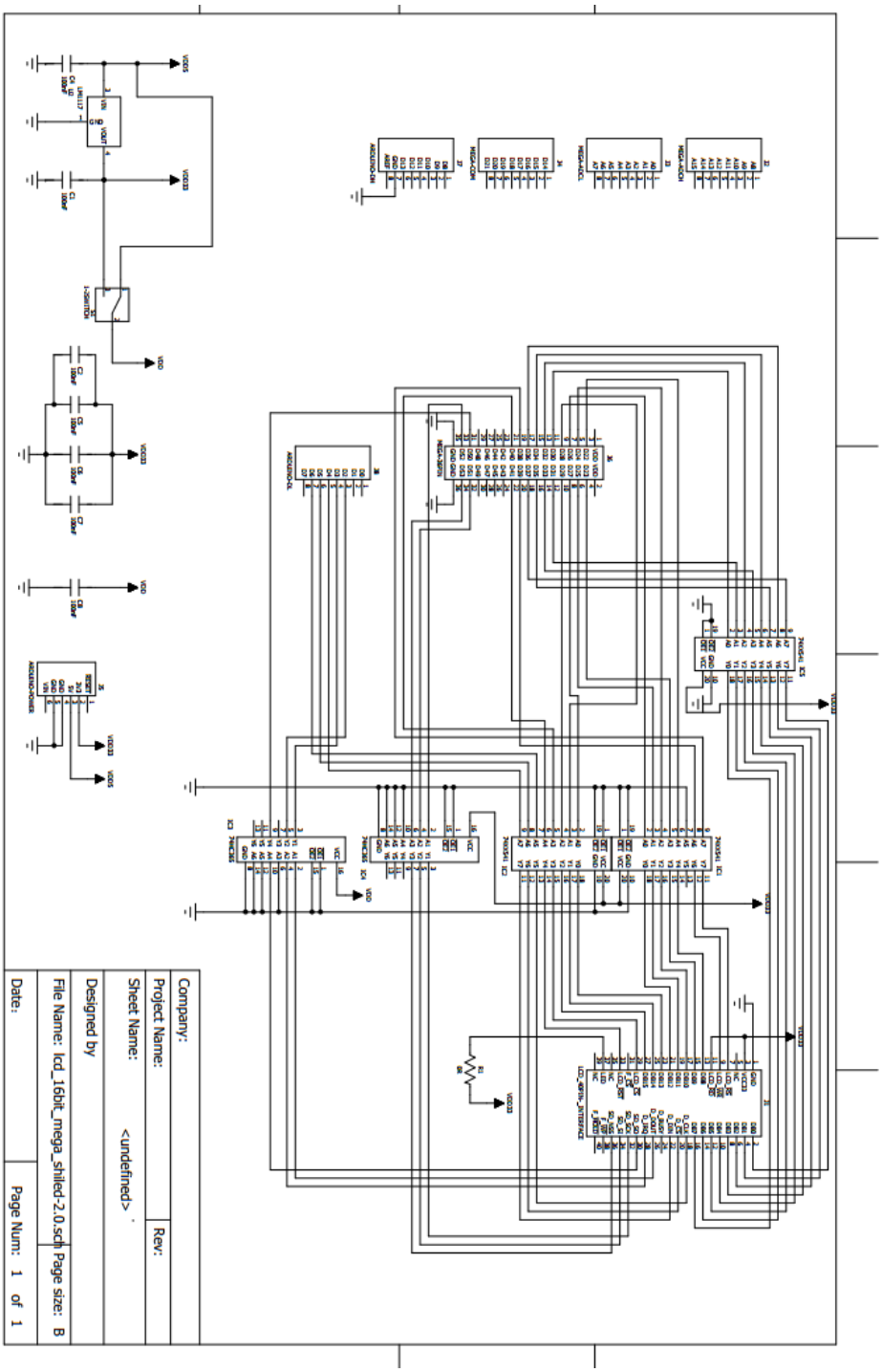
Fonte: “Arduino Cookbook” - Michael Margolis

7.2 Diagrama do Arduino Mega melhor detalhado



Fonte: “Programming Interactivity” - Joshua Noble

7.3 Diagrama eletrônico do Shield



Fonte ITDB02 Arduino MEGA Shield, acesso em outubro de 2014:

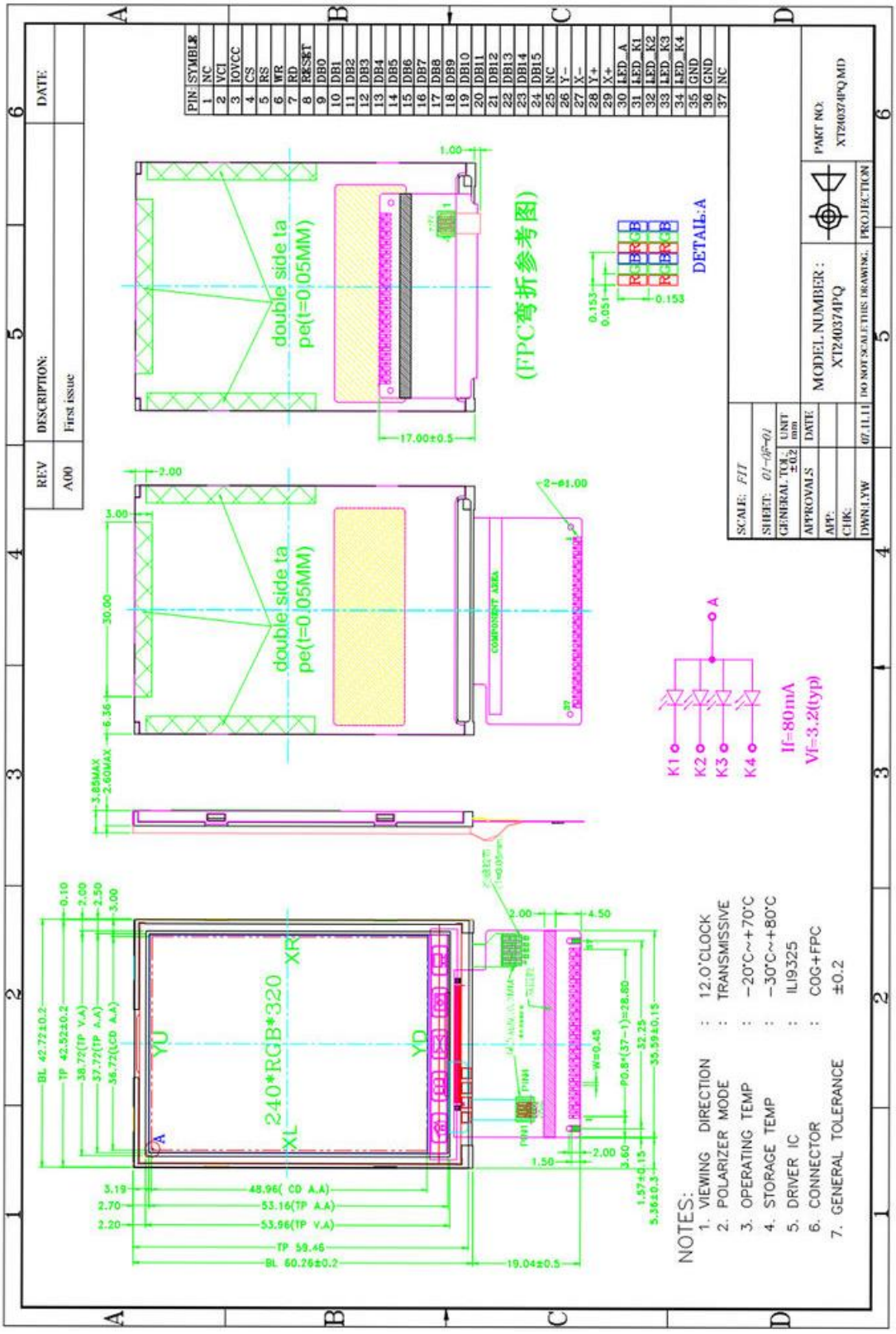
ftp://imall.iteadstudio.com/IM120417024_ITDB02_Arduino_MEGA_Shield/SCH_IM120417024_ITDB02ArduinoMEGAShield.pdf

7.4 Pinagens do Shield

Pin of Arduino	With ITDB02	
3V3	LED_A	RD
GND	GND	
D22(PA0)	DB8	-
D23(PA1)	DB9	-
D24(PA2)	DB10	-
D25(PA3)	DB11	-
D26(PA4)	DB12	-
D27(PA5)	DB13	-
D28(PA6)	DB14	-
D29(PA7)	DB15	-
D37(PC0)	DB0	-
D36(PC1)	DB1	-
D35(PC2)	DB2	-
D34(PC3)	DB3	-
D33(PC4)	DB4	-
D32(PC5)	DB5	-
D31(PC6)	DB6	-
D30(PC7)	DB7	-
D41(PG0)	RESET	-
D40(PG1)	CS	-
D39(PG2)	WR	-
D38(PD7)	RS	-
D50(PB3)	SD_OUT	-
D51(PB2)	SD_IN	-
D52(PB1)	SD_CLK	-
D53(PB0)	SD_CS	-
D6	D_CLK	-
D5	D_CS	-
D4	D_IN	-
D3	D_OUT	-
D2	D_IRQ	

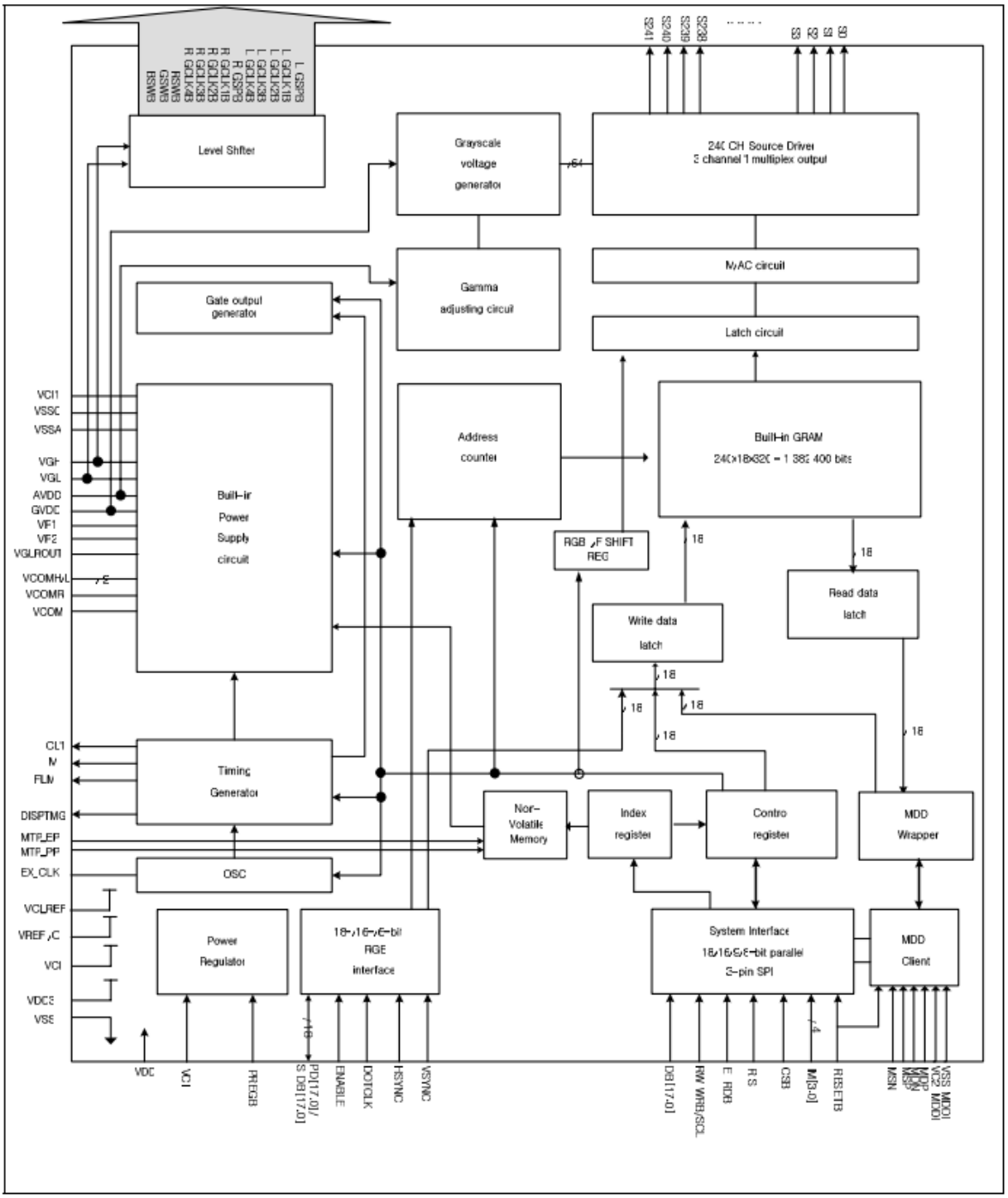
Fonte ITDB02 Arduino MEGA Shield, acesso em outubro de 2014: ftp://imall.iteadstudio.com/IM120417024_ITDB02_Arduino_MEGA_Shield/DS_IM120417024_ITDB02ArduinoMEGAShield.pdf

7.5 Diagrama do LCD



Fonte elecfreaks, acesso em outubro de 2014:
http://www.elecfreaks.com/wiki/index.php?title=2.4%22_TFT_LCD:_TFT01-2.4

7.6 Diagrama de blocos da tela LCD



Fonte Data Sheet S6D1121, acesso outubro 2014:
http://elecfreaks.com/store/download/datasheet/lcd/S6D1121-datasheet.pdf

7.7 Funções da biblioteca UTouch

- UTouch:

A biblioteca UTouch é responsável pelas funções de captura e manipulação do touchscreen. Vejamos as suas funções:

UTouch(TCLK, TCS, TDIN, TDOUT, IRQ);

The main class of the interface.

Parameters:	TCLK: Pin for Touch Clock (D_CLK) TCS: Pin for Touch Chip Select (D_CS) TDIN: Pin for Touch Data input (D_DIN) TDOUT: Pin for Touch Data output (D_OUT) IRQ: Pin for Touch IRQ (DPenirq)
Usage:	`UTouch myTouch(15,10,14,9,8); // Start an instance of the UTouch class`

InitTouch([orientation]);

Initialize the touch screen and set display orientation. If the library is used together with UTFT the orientation should be set to the same orientation for both libraries.

Parameters:	orientation: <optional> PORTRAIT LANDSCAPE (default)
Returns:	Nothing
Usage:	`myTouch.InitTouch();// Initialize the touch screen`

dataAvailable();

Check to see if new data from the touch screen is waiting.

Parameters:	None
Returns:	Boolean: true means data is waiting, otherwise false
Usage:	`check = myTouch.dataAvailable() // See if data is waiting`

read();

Read waiting data from the touch screen. This function should be called if dataAvailable() is true. Use getX() and getY() to get the coordinates.

Parameters:	None
Returns:	Nothing
Usage:	`myTouch.read(); // Read data from touch screen`
Notes:	After calling read(), raw data from the touch screen is available in the variables TP_X and TP_Y. Do not use these if you do not know how to handle the raw data. Use getX() and getY() instead.

getX();

Get the x-coordinate of the last position read from the touch screen.

Parameters:	None
Returns:	Integer
Usage:	`x = myTouch.getX(); // Get the x-coordinate`

getY();

Get the y-coordinate of the last position read from the touch screen.

Parameters:	None
Returns:	Integer
Usage:	`y = myTouch.getY(); // Get the y-coordinate`

setPrecision(precision);

Set the precision of the touch screen.

Parameters:	precision: PREC_LOW, PREC_MEDIUM, PREC_HI, PREC_EXTREME
Returns:	Nothing
Usage:	`myTouch.setPrecision(PREC_MEDIUM); // Set precision to medium`
Notes:	Higher precision data will take longer to read, so take care when using PREC_HI or PREC_EXTREME with fast-moving input.

7.8 Funções da biblioteca UTFT

- UTFT:

A biblioteca UTFT é responsável pelas funções da tela LCD. Vejamos as suas funções:

UTFT(Model, RS, WR, CS, RST[, ALE]);

The main class constructor when using 8bit or 16bit display modules.

Parameters:	Model: See the separate document for the supported display modules RS: Pin for Register Select WR: Pin for Write CS: Pin for Chip Select RST: Pin for Reset ALE: <optional> Only used for latched 16bit shields Pin for Latch signal
Usage:	`UTFT myGLCD(ITDB32S,19,18,17,16); // Start an instance of the UTFT class`

UTFT(Model, SDA, SCL, CS, RST[, RS]);

The main class constructor when using serial display modules.

Parameters:	Model: See the separate document for the supported display modules SDA: Pin for Serial Data SCL: Pin for Serial Clock CS: Pin for Chip Select RST: Pin for Reset RS: <optional> Only used for 5pin serial modules Pin for Register Select
Usage:	`UTFT myGLCD(ITDB18SP,11,10,9,12,8); // Start an instance of the UTFT class`

InitLCD([orientation]);

Initialize the LCD and set display orientation.

Parameters:	Orientation: <optional> PORTRAIT LANDSCAPE **(default)**
Usage:	`myGLCD.initLCD(); // Initialize the display`
Notes:	This will reset color to white with black background. Selected font will be reset to *none*.

getDisplayXSize();

Get the width of the screen in the current orientation.

Parameters:	None
Returns:	Width of the screen in the current orientation in pixels
Usage:	`Xsize = myGLCD.getDisplayXSize(); // Get the width`

getDisplayYSize();

Get the height of the screen in the current orientation.

Parameters:	None
Returns:	Height of the screen in the current orientation in pixels
Usage:	`Ysize = myGLCD.getDisplayYSize(); // Get the height`

lcdOff();

Turn off the LCD. No commands will be executed until a lcdOn(); is sent.

Parameters:	None
Usage:	`myGLCD.lcdOff(); // Turn off the lcd`
Notes:	This function is currently only supported on PCF8833 and CPLD-based displays. CPLD-based displays will only turn off the backlight. It will accept further commands/writes.

lcdOn();

Turn on the LCD after issuing a lcdOff()-command.

```
Parameters:   None
Usage:        myGLCD.lcdOn(); // Turn on the lcd
Notes:        This function is currently only supported on PCF8833 and CPLD-based displays.
              CPLD-based displays will only turn on the backlight.
```

setContrast(c);

Set the contrast of the display.

```
Parameters:   c: Contrast-level (0-64)
Usage:        myGLCD.setContrast(64); // Set contrast to full (default)
Notes:        This function is currently only supported on PCF8833-based displays
```

setBrightness(br);

Set the brightness of the display backlight.

```
Parameters:   br: Brightness-level (0-16)
Usage:        myGLCD.setBrightness(16); // Set brightness to maximum (default)
Notes:        This function is currently only supported on CPLD-based displays
```

setDisplayPage(pg);

Set which memory page to display.

```
Parameters:   pg: Page (0-7) (0 is default)
Usage:        myGLCD.setDisplayPage(4); // Display page 4
Notes:        This function is currently only supported on CPLD-based displays
```

setWritePage(pg);

Set which memory page to use for subsequent display writes.

```
Parameters:   pg: Page (0-7) (0 is default)
Usage:        myGLCD.setWritePage(2); // Use page 2 for subsequent writes
Notes:        This function is currently only supported on CPLD-based displays
```

clrScr();

Clear the screen. The background-color will be set to black.

```
Parameters:   None
Usage:        myGLCD.clrScr(); // Clear the screen
```

fillScr(r, g, b);

Fill the screen with a specified color.

```
Parameters:   r: Red component of an RGB value (0-255)
              g: Green component of an RGB value (0-255)
              b: Blue component of an RGB value (0-255)
Usage:        myGLCD.fillScr(255,127,0); // Fill the screen with orange
```

fillScr(color);

Fill the screen with a specified pre-calculated RGB565 color.

```
Parameters:   color: RGB565 color value
Usage:        myGLCD.fillScr(VGA_RED); // Fill the screen with red
```

setColor(r, g, b);

Set the color to use for all draw*, fill* and print commands.

```
Parameters:   r: Red component of an RGB value (0-255)
              g: Green component of an RGB value (0-255)
              b: Blue component of an RGB value (0-255)
Usage:        myGLCD.setColor(0,255,255); // Set the color to cyan
```

setColor(color);

Set the specified pre-calculated RGB565 color to use for all draw*, fill* and print commands.

```
Parameters:   color: RGB565 color value
Usage:        myGLCD.setColor(VGA_AQUA); // Set the color to aqua
```

getColor();

Get the currently selected color.

Parameters:	None
Returns:	Currently selected color as a RGB565 value (word)
Usage:	`Color = myGLCD.getColor(); // Get the current color`

setBackColor(r, g, b);

Set the background color to use for all print commands.

Parameters:	r: Red component of an RGB value (0-255) g: Green component of an RGB value (0-255) b: Blue component of an RGB value (0-255)
Usage:	`myGLCD.setBackColor(255,255,255); // Set the background color to white`

setBackColor(color);

Set the specified pre-calculated RGB565 background color to use for all print commands.

Parameters:	color: RGB565 color value
Usage:	`myGLCD.setBackColor(VGA_LIME); // Set the background color to lime`

getBackColor();

Get the currently selected background color.

Parameters:	None
Returns:	Currently selected background color as a RGB565 value (word)
Usage:	`BackColor = myGLCD.getBackColor(); // Get the current background color`

drawPixel(x, y);

Draw a single pixel.

Parameters:	x: x-coordinate of the pixel y: y-coordinate of the pixel
Usage:	`myGLCD.drawPixel(119,159); // Draw a single pixel`

drawLine(x1, y1, x2, y2);

Draw a line between two points.

Parameters:	x1: x-coordinate of the start-point y1: y-coordinate of the start-point x2: x-coordinate of the end-point y2: y-coordinate of the end-point
Usage:	`myGLCD.drawLine(0,0,239,319); // Draw a diagonal line`

drawRect(x1, y1, x2, y2);

Draw a rectangle between two points.

Parameters:	x1: x-coordinate of the start-corner y1: y-coordinate of the start-corner x2: x-coordinate of the end-corner y2: y-coordinate of the end-corner
Usage:	`myGLCD.drawRect(119,159,239,319); // Draw a rectangle`

drawRoundRect(x1, y1, x2, y2);

Draw a rectangle with slightly rounded corners between two points. The minimum size is 5 pixels in both directions. If a smaller size is requested the rectangle will not be drawn.

Parameters:	x1: x-coordinate of the start-corner y1: y-coordinate of the start-corner x2: x-coordinate of the end-corner y2: y-coordinate of the end-corner
Usage:	`myGLCD.drawRoundRect(0,0,119,159); // Draw a rounded rectangle`

fillRect(x1, y1, x2, y2);

Draw a filled rectangle between two points.

Parameters:	x1: x-coordinate of the start-corner y1: y-coordinate of the start-corner x2: x-coordinate of the end-corner y2: y-coordinate of the end-corner
Usage:	`myGLCD.fillRect(119,0,239,159); // Draw a filled rectangle`

fillRoundRect(x1, y1, x2, y2);

Draw a filled rectangle with slightly rounded corners between two points. The minimum size is 5 pixels in both directions. If a smaller size is requested the rectangle will not be drawn.

Parameters:
```
x1: x-coordinate of the start-corner
y1: y-coordinate of the start-corner
x2: x-coordinate of the end-corner
y2: y-coordinate of the end-corner
```
Usage:
```
myGLCD.fillRoundRect(0,159,119,319); // Draw a filled, rounded rectangle
```

drawCircle(x, y, radius);

Draw a circle with a specified radius.

Parameters:
```
x:      x-coordinate of the center of the circle
y:      y-coordinate of the center of the circle
radius: radius of the circle in pixels
```
Usage:
```
myGLCD.drawCircle(119,159,20); // Draw a circle with a radius of 20 pixels
```

fillCircle(x, y, radius);

Draw a filled circle with a specified radius.

Parameters:
```
x:      x-coordinate of the center of the circle
y:      y-coordinate of the center of the circle
radius: radius of the circle in pixels
```
Usage:
```
myGLCD.fillCircle(119,159,10); // Draw a filled circle with a radius of 10 pixels
```

print(st, x, y[, deg]);

Print a string at the specified coordinates.
You can use the literals LEFT, CENTER and RIGHT as the x-coordinate to align the string on the screen.

Parameters:
```
st:  the string to print
x:   x-coordinate of the upper, left corner of the first character
y:   y-coordinate of the upper, left corner of the first character
deg: <optional>
     Degrees to rotate text (0-359). Text will be rotated around the upper left corner.
```
Usage:
```
myGLCD.print("Hello, World!",CENTER,0); // Print "Hello, World!"
```
Notes:
```
CENTER and RIGHT will not calculate the coordinates correctly when rotating text.
The string can be either a char array or a String object
```

printNumI(num, x, y[, length[, filler]]);

Print an integer number at the specified coordinates.
You can use the literals LEFT, CENTER and RIGHT as the x-coordinate to align the string on the screen.

Parameters:
```
num:    the value to print (-2,147,483,648 to 2,147,483,647) INTEGERS ONLY
x:      x-coordinate of the upper, left corner of the first digit/sign
y:      y-coordinate of the upper, left corner of the first digit/sign
length: <optional>
        minimum number of digits/characters (including sign) to display
filler: <optional>
        filler character to use to get the minimum length. The character will be inserted in front
        of the number, but after the sign. Default is ' ' (space).
```
Usage:
```
myGLCD.printNumI(num,CENTER,0); // Print the value of "num"
```

printNumF(num, dec, x, y[, divider[, length[, filler]]]);

Print a floating-point number at the specified coordinates.
You can use the literals LEFT, CENTER and RIGHT as the x-coordinate to align the string on the screen.
WARNING: Floating point numbers are not exact, and may yield strange results when compared. Use at your own discretion.

Parameters:
```
num:     the value to print (See note)
dec:     digits in the fractional part (1-5) 0 is not supported. Use printNumI() instead.
x:       x-coordinate of the upper, left corner of the first digit/sign
y:       y-coordinate of the upper, left corner of the first digit/sign
divider: <Optional>
         Single character to use as decimal point. Default is '.'
length:  <optional>
         minimum number of digits/characters (including sign) to display
filler:  <optional>
         filler character to use to get the minimum length. The character will be inserted in front
         of the number, but after the sign. Default is ' ' (space).
```
Usage:
```
myGLCD.printNumF(num, 3, CENTER,0); // Print the value of "num" with 3 fractional digits
```
Notes:
```
Supported range depends on the number of fractional digits used.
Approx range is +/- 2*(10^(9-dec))
```

setFont(fontname);

Select font to use with print(), printNumI() and printNumF().

Parameters:	fontname: Name of the array containing the font you wish to use
Usage:	myGLCD.setFont(BigFont); // Select the font called BigFont
Notes:	You must declare the font-array as an external or include it in your sketch.

getFont();

Get the currently selected font.

Parameters:	None
Returns:	Currently selected font
Usage:	CurrentFont = myGLCD.getFont(); // Get the current font

getFontXsize();

Get the width of the currently selected font.

Parameters:	None
Returns:	Width of the currently selected font in pixels
Usage:	Xsize = myGLCD.getFontXsize (); // Get font width

getFontYsize();

Get the height of the currently selected font.

Parameters:	None
Returns:	Height of the currently selected font in pixels
Usage:	Ysize = myGLCD.getFontYsize (); // Get font height

drawBitmap (x, y, sx, sy, data[, scale]);

Draw a bitmap on the screen.

Parameters:	x: x-coordinate of the upper, left corner of the bitmap y: y-coordinate of the upper, left corner of the bitmap sx: width of the bitmap in pixels sy: height of the bitmap in pixels data: array containing the bitmap-data scale: **<optional>** Scaling factor. Each pixel in the bitmap will be drawn as <scale>x<scale> pixels on screen.
Usage:	myGLCD.drawBitmap(0, 0, 32, 32, bitmap); // Draw a 32x32 pixel bitmap
Notes:	You can use the online-tool "*ImageConverter 565*" or "*ImageConverter565.exe*" in the Tools-folder to convert pictures into compatible arrays. The online-tool can be found on my website. Requires that you *#include <avr/pgmspace.h>* when using an Arduino other than Arduino Due.

drawBitmap (x, y, sx, sy, data, deg, rox, roy);

Draw a bitmap on the screen with rotation.

Parameters:	x: x-coordinate of the upper, left corner of the bitmap y: y-coordinate of the upper, left corner of the bitmap sx: width of the bitmap in pixels sy: height of the bitmap in pixels data: array containing the bitmap-data deg: Degrees to rotate bitmap (0-359) rox: x-coordinate of the pixel to use as rotational center relative to bitmaps upper left corner roy: y-coordinate of the pixel to use as rotational center relative to bitmaps upper left corner
Usage:	myGLCD.drawBitmap(50, 50, 32, 32, bitmap, 45, 16, 16); // Draw a bitmap rotated 45 degrees around its center
Notes:	You can use the online-tool "*ImageConverter 565*" or "*ImageConverter565.exe*" in the Tools-folder to convert pictures into compatible arrays. The online-tool can be found on my website. Requires that you *#include <avr/pgmspace.h>* when using an Arduino other than Arduino Due.