

INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA  
ÁREA DEPARTAMENTAL DE ENGENHARIA DE  
ELECTRÓNICA E TELECOMUNICAÇÕES E DE  
COMPUTADORES  
LICENCIATURA DE ENGENHARIA INFORMÁTICA E  
MULTIMÉDIA

Projeto  
**Consola de Jogo Portátil**



**Alunos:** Francisco Marques (45116)  
Júlio Lima (45115)

**Orientador:** Prof. Manfred Niehus

10 de setembro de 2021



# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>11</b>
1.1	Enquadramento . . . . .	12
1.2	Mercado . . . . .	13
<b>2</b>	<b>Proposta</b>	<b>17</b>
2.1	Motivação e Fundamentos . . . . .	17
2.2	Requisitos . . . . .	18
<b>3</b>	<b>Implementação</b>	<b>19</b>
3.1	Hardware . . . . .	19
3.1.1	Microcontrolador . . . . .	20
3.1.2	Interruptores . . . . .	21
3.1.3	Bateria . . . . .	22
3.1.4	Carregamento e <i>Step-up</i> . . . . .	23
3.1.5	Ecrã . . . . .	26
3.1.6	Altifalante . . . . .	27
3.2	Esquemática . . . . .	28
3.2.1	Placa de Circuitos . . . . .	30
3.2.2	Produção . . . . .	32
3.2.3	Montagem . . . . .	33
3.3	Software . . . . .	35
3.3.1	Projeção Inicial . . . . .	35
3.3.2	Arquitetura . . . . .	38
3.4	Encapsulamento . . . . .	53
3.4.1	<i>SolidWorks</i> . . . . .	53
3.4.2	Caixa . . . . .	54
<b>4</b>	<b>Resultado</b>	<b>57</b>

<b>5 Conclusões</b>	<b>57</b>
5.1 Equipa . . . . .	57
5.2 Projeto . . . . .	58
5.3 Final . . . . .	59
<b>6 Bibliografia</b>	<b>61</b>



## **Lista de Figuras**

1	Sistema Game & Watch . . . . .	11
2	Sistema Playstation Portable 1000 . . . . .	12
3	Emulador Visual Boy Advance . . . . .	13
4	Anbernic RG350M . . . . .	14
5	Ecrã de criação de sprites do sistema PICO-8 . . . . .	14
6	Consola oficial Arduboy . . . . .	15
7	Microcontrolador Arduino Nano . . . . .	20
8	Interruptor tátil . . . . .	21
9	Interruptor deslizante . . . . .	21
10	Esquema de leitura de corrente . . . . .	22
11	Bateria de polímero de lítio . . . . .	22
12	Módulo de carregamento e <i>step-up booster</i> . . . . .	23
13	Esquemática do chip <i>TP4056</i> . . . . .	23
14	Ciclo de carregamento completo . . . . .	24
15	Resistência de programação . . . . .	25
16	Componentes do conversor <i>boost</i> . . . . .	25
17	Ecrã OLED . . . . .	26
18	Comparação entre tecnologias LCD e OLED . . . . .	26
19	Comparação entre comunicações <i>half-duplex</i> e <i>full-duplex</i> . . . . .	27
20	Altifalante <i>buzzer</i> . . . . .	27
21	Ferramenta de pesquisa de componentes . . . . .	28
22	Esquemática elétrica . . . . .	29
23	Janela de desenho da placa de circuito . . . . .	30
24	Exemplo de <i>footprint</i> . . . . .	30

25	Desenho final da placa de circuitos com camadas de cobre . . . . .	31
26	Visualização 3D do desenho final . . . . .	32
27	Passo do procedimento de acabamento HASL . . . . .	32
28	Placa de circuitos final da consola . . . . .	33
29	Passo do procedimento de acabamento HASL . . . . .	34
30	Consola <i>Arcade</i> de jogos . . . . .	35
31	Jogo <i>Pong</i> . . . . .	36
32	Exemplo dos primeiros jogos <i>rogue-like</i> . . . . .	36
33	Placa de circuitos final da consola . . . . .	37
34	Ecrã inicial do jogo . . . . .	38
35	Exemplo de um <i>sprite</i> desenhado usando a função <code>drawBitmap</code> .	39
36	<i>Tileset</i> com os <i>sprites</i> utilizados . . . . .	40
37	Ecrã de escolha de herói . . . . .	40
38	Ecrã de jogo . . . . .	42
39	Ecrã onde se equipam os itens . . . . .	44
40	Algoritmo para a escolha de itens . . . . .	45
41	Ecrã de registo de <i>highscore</i> . . . . .	47
42	Correspondência entre pinos do Arduino e portas do ATMEGA328	48
43	Fluxograma dos vários ecrãs . . . . .	52
44	Design 3D a partir de um esboço 2D . . . . .	53
45	Exemplo de uma montagem com duas partes . . . . .	54
46	Exemplo de um encaixe <i>Snap Hook</i> . . . . .	55
47	Caixa da consola . . . . .	55
48	Botões encaixados na parte de cima da caixa . . . . .	56
49	Produto final . . . . .	57



## Listagens

1	Função <code>setup</code> para inicializar o programa . . . . .	38
2	Função <code>drawBitmap</code> utilizando variáveis de posição . . . . .	39
3	Classe <code>Hero</code> . . . . .	41
4	Função executada após passagem de nível . . . . .	43
5	Atualização do ataque dependendo dos itens equipados . . . . .	44
6	Função de geração de itens . . . . .	45
7	Verificação de <i>highscore</i> conquistado . . . . .	46
8	Alteração da ordem de <i>highscores</i> . . . . .	46
9	Escrita de uma nova pontuação . . . . .	47
10	Definição dos botões nas portas do chip ATMEGA328 . . . . .	48
11	Definição do pino e portas do botão A . . . . .	49
12	Exemplo do uso da função <code>justPressed</code> . . . . .	49
13	Definição de ecrãs e a variável que guarda o atual . . . . .	51
14	<code>case</code> do ecrã <i>over</i> . . . . .	52



# 1 Introdução

Nos dias de hoje, videojogos são um forte componente social e educativo presente na vida das pessoas de todas as faixas etárias. O primeiro tipo de videojogos, introduzidos à população nos anos 50, não se tornariam convencionalmente aceites até aos anos 80, onde a maior parte dos jogos desenvolvidos seriam conversões de jogos previamente existentes em *arcades*, para consolas de carácter familiar. Com isto, os jogos deixam de ser constrangidos a espaços públicos ou a poder monetário, e começam a adotar um grande papel no divertimento da comunidade.

Uma das mais notórias e fascinantes histórias da evolução dos videojogos conta que Gunpei Yokoi, antigo *designer* de videojogos da Nintendo, numa certa viagem de comboio viu um homem de negócios a brincar com os botões de uma calculadora. Desta interação surgiram os primeiros sistemas *Game & Watch*, um relógio portátil que funciona como uma miniatura de consola de videojogo, com o fim de 'matar tempo'.



**Figura 1:** Sistema Game & Watch

Esta invenção proporcionou o inicio do reconhecimento e desenvolvimento da jogabilidade portátil. Em 1989 é lançada a consola *Gameboy*, e vem acompanhada de um título que é conhecido por todos nós: o *Tetris*, que foi um *hit* instantâneo. Em 2005, o *Gameboy* e a sua versão a cores lançada em 1998, o *Gameboy Color*, juntos, teriam vendido mais de 118 milhões de unidades.

A Nintendo continua a dominar este setor até aos dias de hoje. Com o lançamento dos *upgrades* à série *Gameboy* com os sistemas *Advance* e *Advance SP*, a revolução dos sistemas DS com ecrã de toque e capacidades de processamento em crescimento exponencial, e, finalmente, a *Switch*, que implementa características de sistemas familiares e portáteis, a Nintendo marca a sua presença e revoluciona a percepção da população para sempre.

## 1.1 Enquadramento

Apesar da Nintendo dominar o mercado de consolas portáteis à mais de 30 anos, não significa que não tenha tido competição. Ao longo dos anos houve uma multitude de empresas deste setor que decidiram implementar e desenvolver a sua visão de um sistema sem fios, e muitos tiveram sucesso.

Em 2004, a Sony, empresa que, até ao momento, teria como foco o setor de *home gaming*, ramifica o seu negócio e entra no mundo de jogabilidade portátil com o lançamento da primeira revisão do sistema *Playstation Portable*.



**Figura 2:** Sistema Playstation Portable 1000

O sistema PSP vai ficar gravado na história da jogabilidade portátil devido a uma multitude de fatores. Com o lançamento da consola, junta-se uma grande comunidade de *hacking*, que juntos tentariam descobrir potenciais *exploits* que poderiam correr código não assinado. Em 2005, o código interno do sistema é partilhado na Internet. Nos anos seguintes seriam expostos uma grande quantidade de *exploits*, maioritariamente em jogos específicos, como o demo do título *Patapon 2*, que permitia que o utilizador inicializasse jogos ou aplicações não nativas ao sistema, como **emuladores** de outras consolas, portáteis ou de carácter familiar.

Os utilizadores, com esta vulnerabilidade, podiam correr jogos dos sistemas *Gameboy*, *NES* e *Super NES*, *Nintendo 64*, bem como jogos únicos e divertidos desenvolvidos por membros da comunidade tudo num único sistema, que era bastante acessível para as especificações que apresentava.

Esta evolução do poder dos consumidores instituiu a mania do *retro gaming*, prevalente até aos dias de hoje, onde a separação entre corporações e utilizadores com interesses para além da *rollercoaster ride* que as corporações oferecem aos jogadores é evidente.

## 1.2 Mercado

Atualmente, é facilmente observável a mudança de mentalidade sobre o tema. Os criadores adotam projetos de menor escala e de natureza *open-source*, com base em emulação ou totalmente independentes. Isto deve-se a duas razões:

1. Consequências sofridas provenientes da ilegalidade da manipulação de sistemas protegidos contra exploração indevida;
2. Deterioração dos sistemas utilizados devido ao passar dos anos, condições impostas pelo dono e má qualidade de produto;

A comunidade de *retro gaming* agora encontra-se mais forte mas de popularidade modesta. Com a nossa pesquisa admitimos que existem três ramificações de interesse por parte dos membros:

1. Utilizadores que emulam jogos *retro* no computador;
2. Utilizadores que emulam jogos *retro* em consolas portáteis desenvolvidas para tal;
3. Utilizadores que jogam jogos independentes (*homebrew*), com características *retro*, em consolas portáteis desenvolvidas para tal;

Os exemplos de soluções de *software* mais relevantes para utilizadores que apenas pretendem reviver ou criar novas experiências com jogos de uma ou mais consolas específicas que achamos mais relevantes são: RetroArch (sistemas Sega e Nintendo *NES*, *SNES*, *64* e *DS*), Dolphin (sistemas Nintendo *Wii* e *Gamecube*) e Visual Boy Advance (sistemas Nintendo *Gameboy*, todas as versões).



**Figura 3:** Emulador Visual Boy Advance

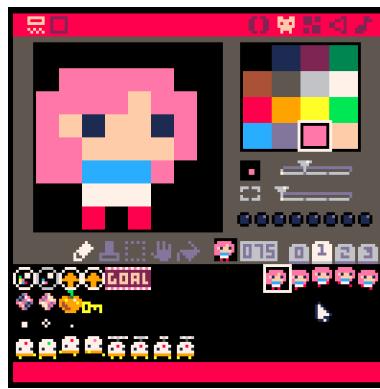
Por outro lado, os utilizadores que procuram emular os sistemas que desejam numa consola portátil não teram sorte a encontrar um projeto sólido e reputável, visto que é ilegal produzir tais sistemas no mundo ocidental. Apesar disto, existem produtos poderosos e bem sucedidos em mercados asiáticos como o Aliexpress. A série de dispositivos *RG35X* da loja Anbernic mostrou grande sucesso entre utilizadores pela sua alta capacidade de processamento, qualidade de produção e dedicação da equipa.



**Figura 4:** Anbernic RG350M

Finalmente, existem várias comunidades dedicadas ao desenvolvimento de aplicações em sistemas operacionais dedicados, tendo como foco o desenvolvimento independente de jogos com características semelhantes a jogos da era *retro*, alcançados através de limitações impostas pelos projetistas. Ambos os membros do grupo concordam que os sistemas *PICO-8* e *Arduboy* são os mais populares do mercado, tendo em conta os atributos definidos.

A "consola de fantasia" *PICO-8* permite que o utilizador jogue, crie e partilhe jogos de ocupação de memória, gama de cores, resolução e poder de processamento reduzido. A principal plataforma focada pela equipa do projeto é o computador, mas existem soluções de outras empresas, como o *GameShell* da ClockworkPi, que emulam projetos desta natureza.



**Figura 5:** Ecrã de criação de sprites do sistema PICO-8

Corre aplicações com resolução de 128x128 pixels, uma paleta de 16 cores predefinidas e áudio a 4 canais. Pelo aspeto visual, podemos notar semelhanças entre este sistema e o sistema *SNES* da Nintendo, um sistema bastante poderoso nos seus tempos de glória, no que cabia à representação de arte pixelada a cores.

Os jogadores que decidam utilizar o seu computador para reviver experiências de jogos passados, ou para criar novas experiências caso sejam entusiastas do género, acabam por ser prejudicados quando comparadas às experiências obtidas com outros possíveis meios. As capacidades e limitações da versão física da consola que o utilizador pretende emular ajudam a tornar a experiência mais autêntica.

Por outro lado, o projeto *Arduboy* tem como foco singular a jogabilidade em *hardware* dedicado e partilha ferramentas e produtos que permitam que os utilizadores obtenham uma experiência semelhante à proporcionada pela primeira iteração da consola *Gameboy*.



**Figura 6:** Consola oficial Arduboy

O sistema *Arduboy* é baseado na arquitetura *Arduino*, apresenta um ecrã monocromático de 128x64 pixels, um único canal de áudio e botões de *input* numa placa de circuitos um pouco maior que o tamanho de um cartão. Incorpora um processador ATmega32u4 de 8MHz, *flash* de 32KB, *RAM* de 2.5KB e *EEPROM* de 1KB. Devido à natureza *open-source* do projeto, é possível adotar as bibliotecas aplicadas num microcontrolador *Arduino* previamente adquirido, sendo que muitos membros da comunidade decidem correr as aplicações desenvolvidas em sistemas caseiros devido ao custo do sistema.



## 2 Proposta

O documento de proposta de projeto apresentado pelo Professor Manfred Niehus resume que a atividade deve consistir no desenvolvimento e implementação de uma consola de jogo portátil, focando-se na importância da disponibilização e utilização de ferramentas de carácter *open-source*. Menciona o desenho de uma consola acessível com base em microcontroladores da família *Arduino* OU *Raspberry Pi*. Apresenta a possibilidade de uso de uma placa eletrónica dedicada como meio de agrupamento de componentes e de realizar a projeção de uma capa que proteja o produto de condições exteriores.

### 2.1 Motivação e Fundamentos

Com a introdução do tema e considerando as datas dos eventos descritos, ambos os membros do grupo admitem ter crescido em paralelo com o progresso que as consolas portáteis tomaram. Os sistemas apresentados, principalmente a gama de sistemas *Gameboy*, ocuparam grande parte do tempo dedicado a lazer da nossa infância e início da adolescência, mas, com o passar dos anos, o interesse foi gradualmente convertido para outros temas. As ocupações extracurriculares dos membros do grupo tornam-se de carácter criativo, focados na projeção, criação e desenvolvimento.

Tendo em conta a instrução universitária que ambos os membros estão a completar, principalmente focada no planeamento e desenvolvimento de aplicações e resolução de soluções de *software*, achamos relevante que haja transparência na interligação de conhecimentos entre este plano e o *hardware* a ser desenvolvido.

É também importante mencionar que uma opinião partilhada e discutida entre o grupo nomeia as consolas nostálgicas como as mais divertidas. Será interessante desenvolver esta temática, verificar as condições que eram impostas e importar os benefícios para o nosso projeto, de modo a transmitir uma experiência semelhante à que nos é estimada. Atualmente, a capacidade de processamento presente nos sistemas encontra-se num plano completamente diferente, quando, por exemplo, comparado com os primeiros sistemas *Gameboy*. Considerámos as limitações desta gama de sistemas em particular, e dos seus jogos nativos:

- Imposição do sistema numa consola nativa, física e portátil;
- Baixo poder de processamento (processador personalizado de 4.19MHz parecido ao Intel 8080);
- Ecrã de baixa resolução (160x144 pixels);
- Representação monocromática com 4 tons de cinzento (2 bits);

Considerando a pesquisa realizada e apresentada previamente que relata as alternativas à compra de sistemas como o mencionado, os membros do grupo concordam que a arquitetura *Arduboy* proporciona a experiência mais aproximada da desejada. Contudo, na opinião da equipa, a consola nativa do sistema aponta para duas características negativas: é cara, chegando a custar 70€ em certas lojas portuguesas, e, apesar de ser extremamente benéfica para indivíduos que queiram desenvolver capacidades da componente de *software*, não se pode dizer o mesmo quando se menciona a componente de *hardware*.

## 2.2 Requisitos

O projeto a ser desenvolvido terá como objetivo satisfazer as qualidades e limitações de uma consola *retro* como o sistema *Gameboy* original, adotando componentes e ideologias recentes que permitam desassociar os problemas dos sistemas originais da representação gráfica e jogabilidade estabelecida na era, colocando a ênfase na libertação da criatividade por parte dos utilizadores sem paradigmas e complicações programáticas. Os principais focos da equipa na projeção e produção da solução final serão:

1. Manter as dimensões da consola reduzidas;
2. Reduzir o custo de produção da consola e componentes necessários;
3. Colocar ênfase na aprendizagem de *hardware* e *software* compatível com a máquina;
4. Utilizar a arquitetura *Arduboy*;
5. Visualizar de dados e gráficos através de um ecrã monocromático;
6. Disponibilizar pouca complexidade de controlo e sónora;
7. *Hardware* e *software open-source*;
8. Reutilização de conhecimento previamente adquirido;

A consola virá acompanhada de um jogo nativo ao sistema, desenvolvido pela equipa, que vai permitir que o jogador tenha uma primeira experiência *out of the box*, que irá servir como meio de aprendizagem para utilizadores pouco experientes. O código do jogo também deve ser *open-source*, permitindo que o utilizador atualize ou adicione informação ao jogo, interprete funcionalidades programadas ou utilize como base do seu próprio jogo. É também proposta a inclusão de uma caixa envolvente que agrupa a placa de circuitos e componentes eletrónicos internos e externos num objeto de carácter simples e de "bordas ásperas", como grande parte da modelação de encapsulamentos de consolas da era *retro*.

## 3 Implementação

A descrição da implementação do produto previamente categorizado encontra-se fragmentada em três subcategorias: as componentes *hardware*, *software* e de *encapsulamento*. Em cada capítulo será descrito o processo tomado, apresentação dos seus constituintes, funcionalidades programadas, bibliotecas e conhecimentos exteriores utilizados.

### 3.1 Hardware

Sustenta-se que o objetivo da constituição da componente *hardware* do projeto é a criação de uma placa de circuitos dedicada que agrupe componentes e módulos necessários para que o utilizador realize *inputs* que interajam com um sistema de jogo, sendo possível verificar os dados de *output* num ambiente gráfico.

Para desenvolver esta *printed circuit board* (PCB) será utilizado o programa *EasyEDA*, uma ferramenta de projeção e automação de desenho de circuitos. Apesar de existir uma ampla gama de material possivelmente utilizável para resolução do problema descrito, foi selecionado o *EasyEDA* devido às seguintes características que disponibiliza para os utilizadores:

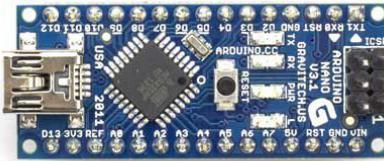
- Acesso a biblioteca de esquemáticas e desenhos de componentes e módulos disponibilizados pela empresa ou membros da comunidade;
- Conversão automática de componentes e ligações estabelecidas numa esquemática para um desenho da placa de circuitos;
- *Auto-routing*, permite que o utilizador estabeleça um servidor que interliga conexões estabelecidas de uma forma eficiente, automática;
- Geração e *upload* de ficheiros de produção automática, permitindo que o utilizador encomende o produto realizado sem esforços;

Com a definição da plataforma utilizada na completação dos capítulos de desenvolvimento da componente *hardware* da consola a ser produzida, inicializa-se a descrição de componentes utilizados e a fabricação da esquemática correspondente.

### 3.1.1 Microcontrolador

Com a utilização da biblioteca *Arduboy* será obrigatória a utilização de um microcontrolador da família *Arduino*. A arquitetura adotada pela empresa criadora do projeto disponibiliza um processador *ATmega32u4*, partilhado pelas versões *Arduino Leonardo* e *Micro*. Contudo, os membros consideram que o *Arduino Nano* seria o mais vantajoso para a constituição da nossa consola.

Apesar desta versão apresentar menor tamanho de *SRAM* (1KB *versus* 2.5KB) e menor tamanho de *EEPROM* (0.51KB *versus* 1KB), disponibiliza também mais pinos, consome menos energia, é mais barato e popular entre a comunidade de utilizadores, e, em virtude de que constituir o processador *ATmega328p*, apresenta um *FTDI chip* que trata da transferência de dados entre dispositivos conectados via USB, que permite a utilização do microcontrolador sem instalação de requisitos extra.



**Figura 7:** Microcontrolador Arduino Nano

Este microcontrolador apresenta também um consumo de energia bastante baixo de 19mA, disponibilizando uma corrente DC de 40mA por pino de *input* ou *output*. Tendo em conta os gastos identificados, esta arquitetura é perfeita para adotar uma bateria de pequeno porte para portabilização do sistema a ser criado.

Sendo que o *Arduino* irá surgir como controlador principal de todos os componentes dispostos, é necessário notar as seguintes notações aplicadas à arquitetura adotada:

- Tensão operacional: 5V;
- Tensão de entrada: 7-12V;
- Portas digitais: 14;
- Portas analógicas: 8;
- Conexão *I2C* por *SDA* e *SCL* nas portas analógicas 4 e 5, respetivamente;

## Componentes de Entrada

Devido à necessidade de interação entre o utilizador e a consola fabricada, são disponibilizados os seguintes componentes e as suas respetivas funcionalidades:

1. Três interruptores táteis principais, com o objetivo de realizar comandos de *input* no ambiente do jogo;
2. Um interruptor deslizante para alterar o estado de ligação da consola (desligada para ligada ou vice-versa);
3. Uma bateria de polímero de lítio que permita a portabilização da consola durante um determinado tempo;
4. Um módulo de carregamento linear completo de corrente e voltagem constante de uma bateria de polímero de lítio que, com um *step up booster* incorporado, permita aumentar a voltagem para entrada no microcontrolador.

### 3.1.2 Interruptores

Os interruptores táteis de montagem superficial utilizados são réplicas do modelo *PTS636* de 2.5mm de altura. Os *switches* utilizados são de baixo custo e apresentam o nome de modelo *3\*6\*2.5H*.



**Figura 8:** Interruptor tátil

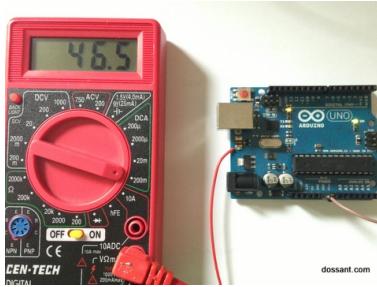
O interruptor deslizante de dois estados escolhido identifica-se como *MSK-12C01-07*, apresenta montagem superficial e dimensões 6.6\*1.98\*1.4mm.



**Figura 9:** Interruptor deslizante

### 3.1.3 Bateria

Para que a consola produzida seja de carácter portátil, esta tem de apresentar uma bateria que permita a inicialização e execução do microcontrolador, componentes e módulos associados. Para tal, é necessário conduzir uma leitura à corrente do aparelho em conjunto, sendo que esta pesquisa é conduzida com a utilização de um multímetro.



**Figura 10:** Esquema de leitura de corrente

Com a execução do esquema apresentado foi possível obter a leitura da corrente da consola e foi obtido o valor médio de **44mA**. Para escolha de uma bateria adequada foram também tomadas as seguintes considerações:

- Tem de apresentar dimensões reduzidas para combinar com o pequeno porte que a consola apresenta;
- Tem de permitir algumas horas de jogabilidade;
- Tem de ser de custo reduzido para a capacidade disponibilizada.

Visto que, em geral, a importação de baterias de países estrangeiros, especialmente de outros continentes, é extremamente difícil visto que a regulação destes itens dificulta a obtenção de baterias, incluindo de grandes companhias como a *Adafruit*, a bateria escolhida para o projeto terá de ser comprada numa loja portuguesa.



**Figura 11:** Bateria de polímero de lítio

Tendo em conta os pontos apresentados, é escolhido uma bateria de 3.7V de voltagem e de dimensões 6\*17\*30mm, disponível na loja *Mauser*, que replica o modelo 601730 e providencia uma capacidade de carga de **250mAh**. Com isto, é possível determinar que a consola vai ter um tempo de jogabilidade portátil de aproximadamente **5 horas e 40 minutos**. Após este tempo, será necessário que o utilizador recarregue a bateria.

### 3.1.4 Carregamento e *Step-up*

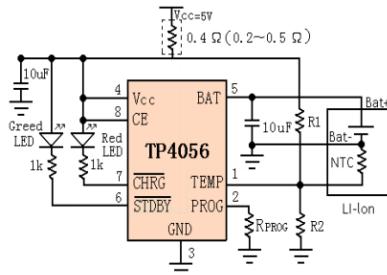
Não é possível que se realize a ligação entre a bateria e o *Arduino* diretamente, porque a bateria adotada apresenta uma voltagem de **3.7V** e, considerando que o *Arduino* necessita de alimentação a 7-12V, será necessário um módulo de *step-up* que permita aumentar a voltagem de saída da bateria.

Foi então utilizado um módulo bastante famoso para resolução de problemas deste carácter, que conjuga um **TP4056**, uma entrada micro USB e um *step-up booster*.



**Figura 12:** Módulo de carregamento e *step-up booster*

O chip **TP4056** atua como um carregador linear de corrente e voltagem constante para baterias de lítio de uma célula. É muito aplicado em projetos portáteis devido à sua configuração SOP e baixo número de componentes externos.

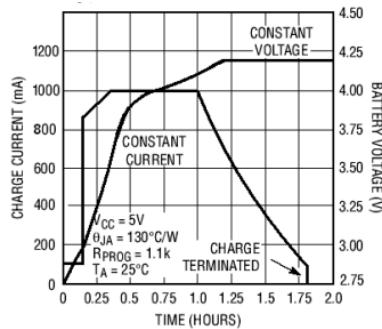


**Figura 13:** Esquemática do chip **TP4056**

As funcionalidades dos pinos do chip representado são as seguintes:

- TEMP (pino 1): Carregamento é suspenso caso a temperatura da bateria estiver demasiado alta ou baixa. Isto é verificado caso a voltagem deste pino estiver a menos de 45% ou a mais de 85% da voltagem de alimentação durante mais de 0.15 segundos;
- PROG (pino 2): A corrente de carga é definida e monitorizada através da ligação de uma resistência deste pino até ao GND. Na eventualidade de carregamento de uma bateria, a voltagem é regulada para 2V, caso contrário, a voltagem é regulada para 0.2V;
- GND (pino 3): Terminal terra;
- VCC (pino 4): Fonte de energia para o circuito interno. Quando o seu valor baixa para dentro de 30mV de diferença da voltagem do pino BAT, o *TP4056* entra em *low power sleep mode*;
- BAT (pino 5): Conectado ao terminal positivo da bateria;
- STDBY (pino 6): Quando o carregamento da bateria é considerado terminado, o pino STDBY é puxado para baixo por um *switch* interno;
- CHRG (pino 7): Quando a bateria está a ser carregada, o pino CHRG é puxado para baixo por um *switch* interno;
- CE (pino 8): Caso este pino seja puxado para cima, o *TP4056* volta a modo normal de operação.

A voltagem de carregamento é fixa a 4.2V e a corrente de carregamento pode ser configurada externamente através de uma única resistência. O chip termina automaticamente o ciclo de carregamento quando a corrente desce para **um décimo** do valor programado, após a tensão flutuante final ter sido alcançada.

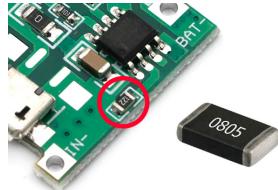


**Figura 14:** Ciclo de carregamento completo

Como foi mencionado anteriormente, a corrente de carga da bateria pode ser definida com uma resistência que conecta o pino PROG à terra. O valor da resistência a utilizar pode ser verificado através da seguinte equação:

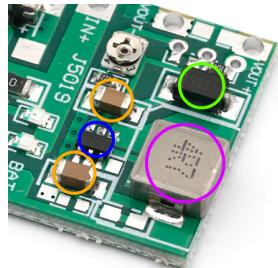
$$I_{BAT} = \frac{V_{PROG}}{R_{PBOG}} \times 1200 \quad ( V_{PROG} = 1 \text{ V}) \quad (1)$$

Considerando que  $I_{BAT} = 0.25Ah$ ,  $R_{PBOG} = 4,800\Omega \approx 5k\Omega$ , é possível admitir que a resistência de tamanho 0803 e de numeração 122 de  $1.2k\Omega$  previamente soldada ao módulo necessita de ser substituída por uma resistência de numeração 5001 de  $5k\Omega$  para obter uma corrente de carga adequada.



**Figura 15:** Resistência de programação

Com a secção de carregamento da bateria do módulo analisado corretamente programada, podemos divergir a atenção para o segmento com funcionalidade de conversor *boost*. Com uma tensão de entrada de 4.5-8V, este módulo consegue ajustar continuamente a tensão de saída para valores compreendidos entre 4.3-27V, visto que isto é possível devido à combinação de um diodo (●) e um transistor (●), ou seja, dois semicondutores, e um elemento de armazenamento de energia, neste caso um indutor (●). Com a intenção de reduzir o efeito de *ripple* da tensão, são adicionados capacitores (●) na entrada e saída do conversor.



**Figura 16:** Componentes do conversor *boost*

## Componentes de Saída

Com o objetivo de representar os resultados provenientes da computação do programa a ser executado e interações criadas por meio de um *input* como o pressionar de um botão, serão necessários os seguintes componentes ou módulos:

1. Um ecrã que permita a representação de gráficos num domínio visual;
2. Um altifalante que permita tanger efeitos sonoros ou pequenas melodias.

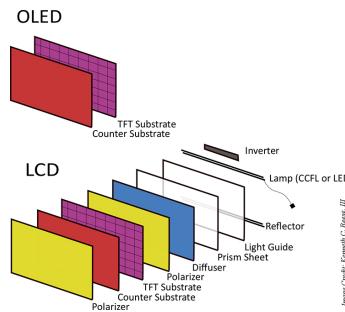
### 3.1.5 Ecrã

A projeção visual é realizada num ecrã monocromático *OLED* de 0.96 polegadas com 128x64 pixels de resolução, controlado pelo *driver SSD1306*. O *SSD1306* incorpora controlo de contraste, RAM de exibição e oscilador, o que reduz o número de componentes externos e consumo de energia.



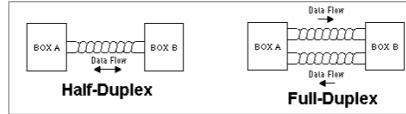
**Figura 17:** Ecrã OLED

A tecnologia OLED (diodo orgânico emissor de luz), considerada melhoria sobre a tecnologia LCD (ecrã de cristal líquido), conta com uma camada orgânica que emite luz quando estimulada por um campo eletromagnético. Este *upgrade* remete para a diminuição de consumo de energia, peso, espessura e aumento do contraste, brilho e ângulo de visão.



**Figura 18:** Comparação entre tecnologias LCD e OLED

Este módulo está disponível em duas versões: uma que utiliza o protocolo *SPI* em *full-duplex* e outra que usa o protocolo *I2C* em *half-duplex*. Ambos utilizam um meio de comunicação síncrona e bidirecional para o envio e receção de dados mas são de diferente execução, sendo que o protocolo *I2C* apenas contém uma linha destinada à transmissão e receção de dados.



**Figura 19:** Comparação entre comunicações *half-duplex* e *full-duplex*

Devido à existência de apenas uma linha de comunicação na comunicação *I2C*, o protocolo batalha esta eventualidade ao garantir a entrega de dados com o envio de um bit de *acknowledge* para cada byte de informação transferida.

Comparando as versões disponíveis do ecrã em questão e considerando que a diferença de *framerate* entre métodos de comunicação é negligível devido à natureza dos gráficos a representar, a versão *I2C* é mais pequena, mais económica, mais popular entre aficionados do *Arduino* e apenas precisa de dois pinos de comunicação e alimentação. Posto isto, a versão do ecrã OLED escolhida será a que utiliza comunicação *I2C*.

### 3.1.6 Altifalante

Considerando as razões da opção tomada no domínio da representação gráfica (baixo custo, tamanho reduzido, popularidade), o domínio sonoro é representado através de um *buzzer* de 5V, de 12cm de diâmetro, onde o seu único canal é controlado pelas entradas positiva e negativa.



**Figura 20:** Altifalante *buzzer*

Não é um verdadeiro altifalante piezoelétrico, mas comporta-se de forma semelhante. Em vez de um cristal piezoelétrico que vibra com uma corrente elétrica, este minúsculo alto-falante usa um eletroímã para acionar uma folha de metal fina. Para tal, é necessário utilizar alguma forma de corrente alternada, neste caso de um microcontrolador, para obter som.

### 3.2 Esquemática

Atendendo às considerações da configuração da consola apresentadas anteriormente, esta nova etapa remete para o estabelecimento de regras de conexões e ligações entre componentes, formando o esquema elétrico base do projeto a desenvolver e servindo como guia para a completação das tarefas seguintes, incluindo o *desenho e projeção da placa de circuitos dedicada*, que vai agrupar os componentes e módulos previamente referidos.

É possível admitir que o microcontrolador encontra-se adequadamente preparado para os componentes e módulos selecionados, visto que:

- Existem pinos suficientes para o projeto, sendo necessários 3 pinos digitais para interruptores táteis, 2 pinos digitais para o *buzzer*, 2 pinos analógicos para o ecrã e 4 para expansibilidade, se necessário;
- O ecrã e *buzzer* funcionam a 5V, possibilitando a alimentação através do respetivo pino do microcontrolador;
- A conexão I2C pode ser corretamente instalada quando conectada aos pinos adequados (**A4** = SDA, **A5** = SCL);
- A alimentação do dispositivo por meio de um módulo de carregamento e conversor *boost* é estabelecida através da conexão da porta de alimentação de saída do respetivo módulo e o pino **Vin** do microcontrolador.

Os desenhos elétricos (símbolos) dos componentes apresentados foram projetados pela equipa da ferramenta *EasyEDA*, pela empresa parceira de revenda de componentes eletrónicos *LCSC* ou até mesmo por utilizadores específicos que quiseram compartilhar o seu trabalho na distribuição de projetos *open-source*.

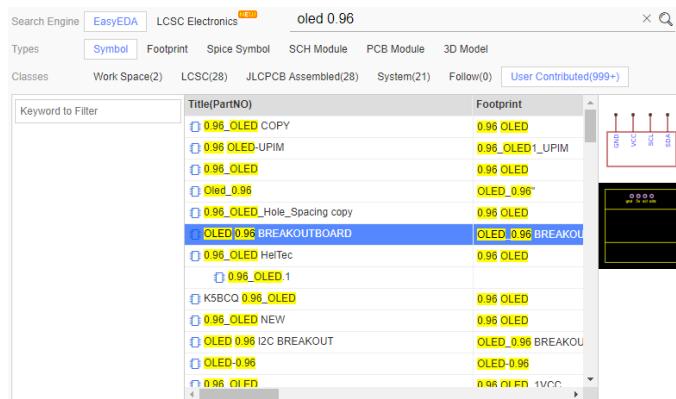
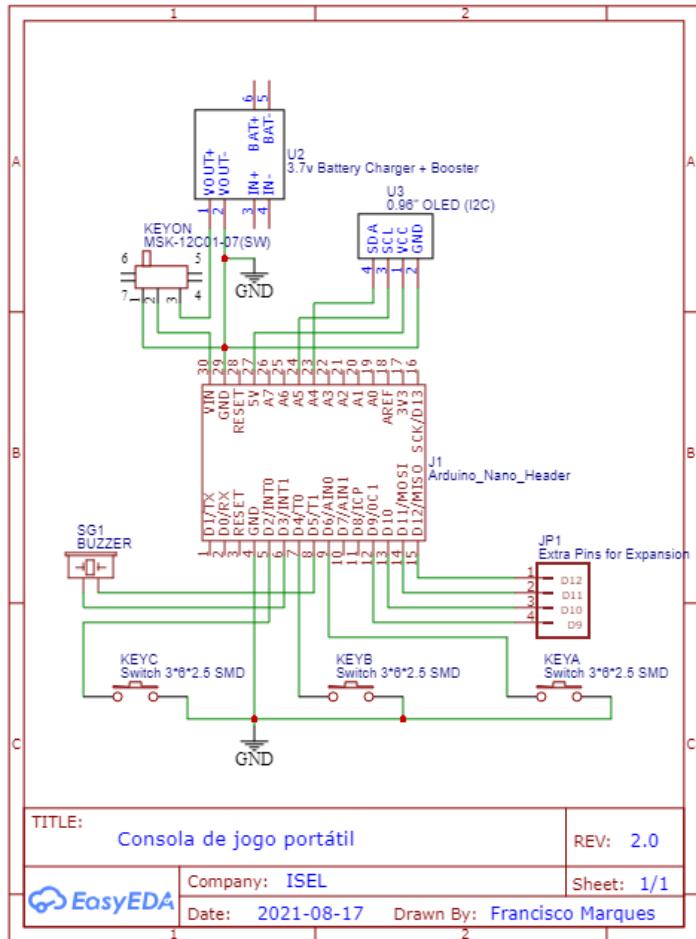


Figura 21: Ferramenta de pesquisa de componentes



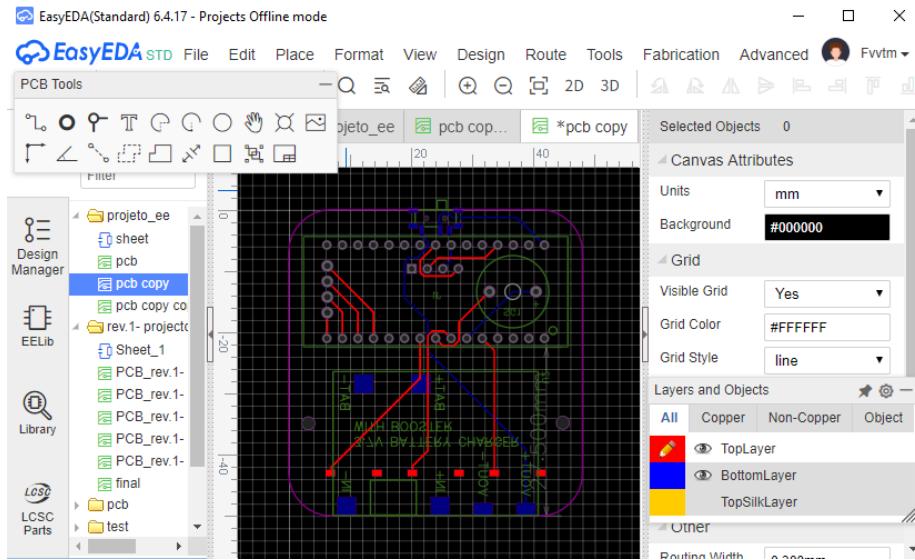
**Figura 22:** Esquemática elétrica

Os elementos representados na esquemática apresentada representam os seguintes módulos e componentes:

- J1: microcontrolador *Arduino Nano*;
- KEYA, KEYB, KEYC: interruptores táteis de *input*;
- U2: módulo de carregamento de baterias e *step-up booster*;
- U3: ecrã OLED de comunicação *I<sub>2</sub>C*;
- KEYON: interruptor deslizante para ligar ou desligar a alimentação do microcontrolador por parte do módulo U2;
- SG1: *buzzer* para representação sonora;
- JP1: propagação de pinos digitais para expansibilidade na placa de circuitos;

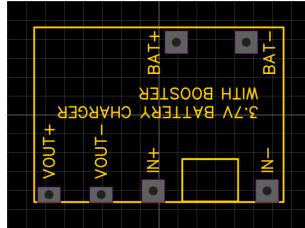
### 3.2.1 Placa de Circuitos

A esquemática elétrica previamente apresentada, que engloba as ligações estabelecidas entre componentes e módulos de *input* e *output* com o microcontrolador *Arduino*, permite que, através do software utilizado *EasyEDA*, seja gerado um ficheiro de desenho da placa de circuitos a ser projetado, de forma automática e atualizável.



**Figura 23:** Janela de desenho da placa de circuito

Este ficheiro gerado contém um limite de placa pré-definido e todos os *footprints* (e os seus respetivos *pads*, vias e conexões sublinhadas) dos componentes e módulos previamente adicionados ao esquema elétrico.

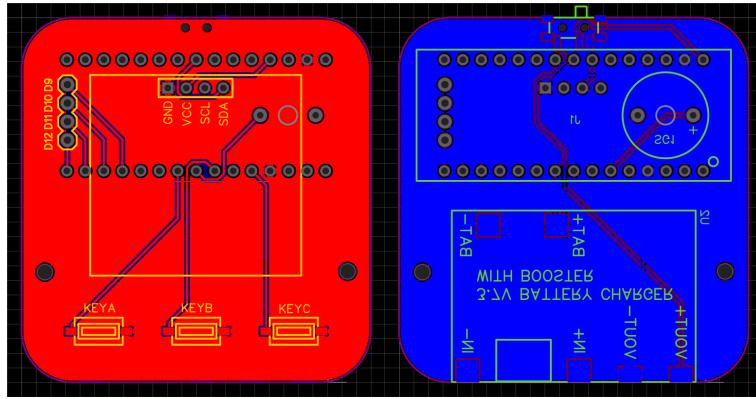


**Figura 24:** Exemplo de *footprint*

Inicialmente é definida uma área através da delimitação final da placa de circuitos a ser posteriormente impressa. Esta área é definida por medidas inseridas pelo utilizador e vai permitir criar uma superfície onde os componentes e módulos previamente retratados vão ser agrupados. Esta delimitação é realizada através das ferramentas de desenho do programa, na camada denominada de *Board Outline*.

Para projetar a iteração final da consola foram realizados os seguintes passos, organizados pela sua ordem de execução:

1. Organizar disposição de componentes e módulos na área delimitada;
2. Visualizar a disposição em 3D para verificar integralmente o espaçamento e posicionamento de componentes e módulos;
3. Utilizar a ferramenta de *auto-routing* para criar pistas que conectam componentes e módulos eficientemente;
4. Corrigir erros nas ligações, caso existam;
5. Remover conexões realizadas entre pinos GND;
6. Criar área de cobre que cubra a camada superior e inferior, conectando os pinos GND dos componentes e módulos;

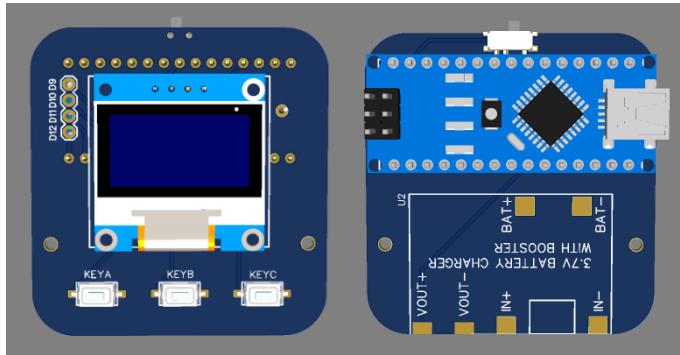


**Figura 25:** Desenho final da placa de circuitos com camadas de cobre

O plano organizado para construção da placa de circuitos consistiu na utilização de **duas camadas** de impressão para construir um produto super compacto. Para tal, a camada superior vai organizar componentes de *input* e *output* visual e de interação com o produto. Por outro lado, a camada inferior vai organizar o microcontrolador e partes internas do sistema. Visto que o *Arduino*, o ecrã e o altifalante apresentam disposição *through-hole*, na montagem do circuito da consola é necessária a remoção de excessos de pinos que existam após a fase de soldagem.

No final, a projeção do plano definido resultou numa placa de circuitos com as seguintes especificações:

- Apresenta dimensão de 50x48mm, com 1.6mm de espessura;
- As ligações e *routings* realizados entre componentes apresentam 0.3mm de espessura;
- Os *pads* da placa de circuito exibem os tamanhos padrão: largura de 1.524mm com um buraco de 0.914mm de diâmetro, à exceção dos *pads* presentes nas laterais da placa, que apresentam 2.2mm de largura com um buraco de 2mm de diâmetro;



**Figura 26:** Visualização 3D do desenho final

Com a conclusão do desenho e projeção da placa de circuito dedicada a ser utilizada na consola, podem agora ser gerados ficheiros **.gbr** (*gerber*), que serão fornecidos à empresa que irá realizar o processo de impressão, visto que estes ficheiros contêm toda a informação necessária para produção do projeto em questão: camadas de cobre, máscara de solda, legendas, perfurações, etc.

### 3.2.2 Produção

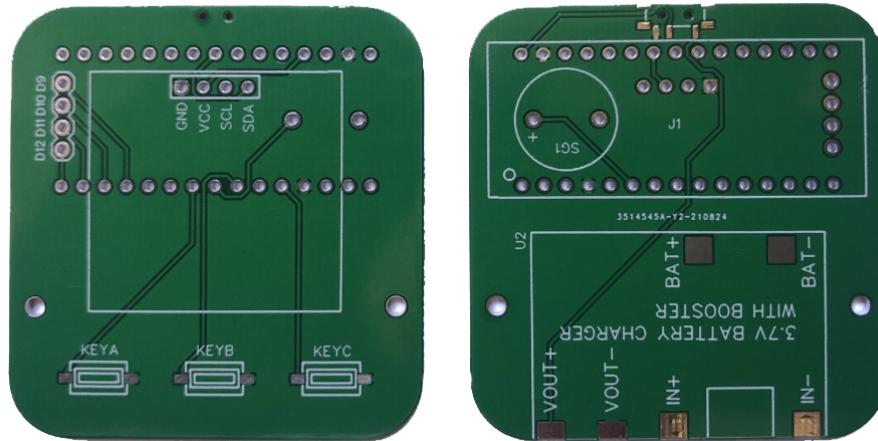
Os ficheiros *gerber* gerados foram encenidos à companhia parceira da empresa autora da ferramenta *EasyEDA*, que apresenta um serviço de impressão do produto projetado. Foram encomendadas 10 placas de circuitos como primeiro *batch*, com as seguintes especificações de produção:

1. *FR-4* como material base da placa de circuitos, sendo um material composto de tecido de fibra de vidro com um aglutinante de resina epóxi resistente a chamas, aprovado pela *National Electrical Manufacturers Association* e extremamente popular entre projetos semelhantes.
2. *HASL* (*Hot Air Solder Leveling*) como acabamento de superfície, onde a placa de circuitos normalmente é mergulhada num banho de solda derretida de modo a que todas as superfícies de cobre expostas sejam revestidas;



**Figura 27:** Passo do procedimento de acabamento HASL

A seguinte sequência de eventos representa a produção da placa de circuitos exemplar do produto final: perfuração, deposição de cobre, projeção das camadas externas, chapeamento, inspeção ótica automática, colocação de máscara de solda e *silkscreen*, HASL, testes elétricos e inspeção final. Com a receção do produto, prossegue-se à comparação com o modelo produzido, à verificação microscópica das ligações e, finalmente, à montagem dos componentes e módulos planeados.



**Figura 28:** Placa de circuitos final da consola

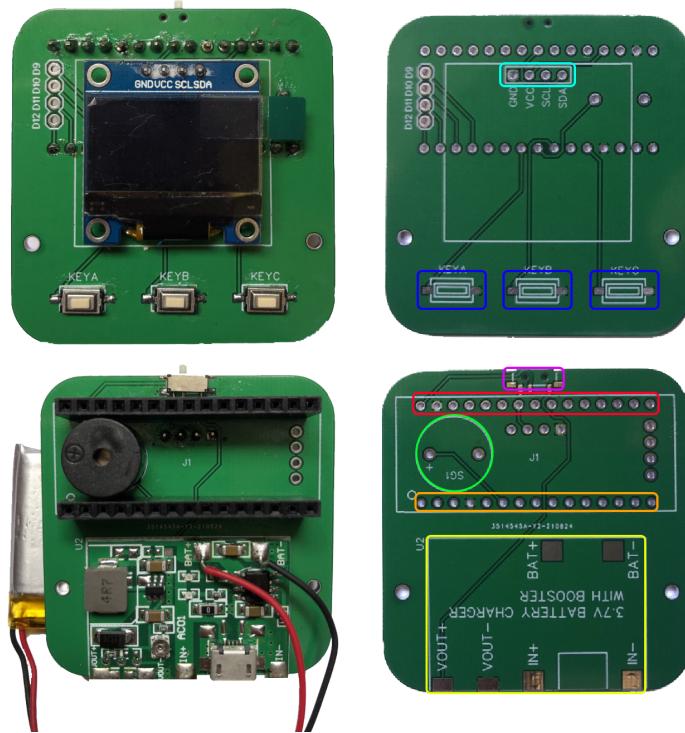
### 3.2.3 Montagem

Com o circuito elétrico desenvolvido, a montagem e soldagem dos componentes à base desenvolvida irá requerir instrumentos auxiliares que permitam a completação da tarefa sucedidamente, sendo que esta listagem agrupa as seguintes ferramentas e complementos:

- Ferro de soldagem;
- Soldadura;
- Fluxo de soldagem, para libertar a camada oxigenada de qualquer tipo de superfície, ampliando a precisão do trabalho;
- Máscara de solda, para afirmar conexões entre o circuito e o módulo de carregamento e *step-up booster*;
- Pistola de ar quente para ativar a máscara de solda;
- Alicate pequeno;

Visto que o circuito base do projeto aplica duas camadas onde ambas agrupam componentes *through-hole*, será necessária uma ordem de montagem e consequentemente, o corte de restos de pinos que ultrapassam para a camada contrária. A ordem de montagem dos componentes, de maior acessibilidade e que permite um resultado profissional, será a seguinte:

1. Módulo de carregamento de bateria e *step-up booster* (●) com resistência 5001 soldada, devido à utilização da pistola de ar quente que possivelmente pode infligir dano em ambas as superfícies superior e inferior;
2. *Header* inferior (●), com os pinos cortados após soldados;
3. *Buzzer* (●) com os pinos cortados após soldados;
4. Interruptor deslizante (●);
5. Ecrã (●). Verificar que não existem conexões de *headers* em contacto com a parte inferior do ecrã suspenso;
6. Interruptores táteis (●);
7. Bateria (●), onde o fio vermelho deve ser soldado ao campo **BAT+** e o preto no **BAT-**;



**Figura 29:** Passo do procedimento de acabamento HASL

### 3.3 Software

Sustenta-se que o objetivo da constituição desta componente de *software* consiste na criação de um jogo interativo que consista com os princípios do projeto previamente definidos, e que proporcione uma experiência entre o comprador do produto e o sistema desenvolvido, diretamente *out-of-the-box*.

É de notar a importância da existência de um programa funcional de teste, que o utilizador possa testar a consola e verificar erros existentes na montagem do produto. Serve de entretenimento para utilizadores focados particularmente na componente de *hardware* (aprendizagem de montagem de circuitos, técnicas de soldagem *through-hole* e *surface-mount*) e permite que os utilizadores mais imersos na área de *software* possam desenvolver conceitos programáticos através da verificação, melhoramento ou alterações divertidas da base de dados dos ficheiros de compilação do projeto, que estão disponibilizados abertamente ao público, visto o carácter *open-source* do projeto.

#### 3.3.1 Projeção Inicial

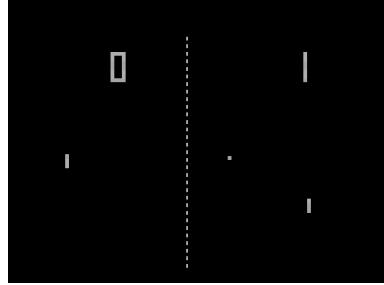
Considerando a natureza *retro* da consola e os pontos base de referência para projeção do produto previamente definidos, foi determinada a meta de criar uma experiência que permita que os utilizadores vivam uma experiência semelhante à proporcionada por consolas antigas, juntamente com características temáticas e de jogabilidade presente em jogos da atualidade.

**Dung!** é um jogo que incorpora mecânicas parentes a um jogo conhecido mundialmente como o primeiro jogo lucrativo da história, o *Pong*, desenvolvido por *Nolan Bushnell* e *Ted Dabney* que mais tarde iriam formar a gigante de entretenimento *Atari*. O jogo desportivo *Pong* consiste num jogo de dois jogadores que simula uma partida de ténis, onde cada utilizador controla o movimento vertical de uma barra (raquete) que permite refletir uma bola para o campo do jogador adversário. Caso a bola trespassse a barra do jogador adversário, este perde um ponto.



**Figura 30:** Consola *Arcade* de jogos

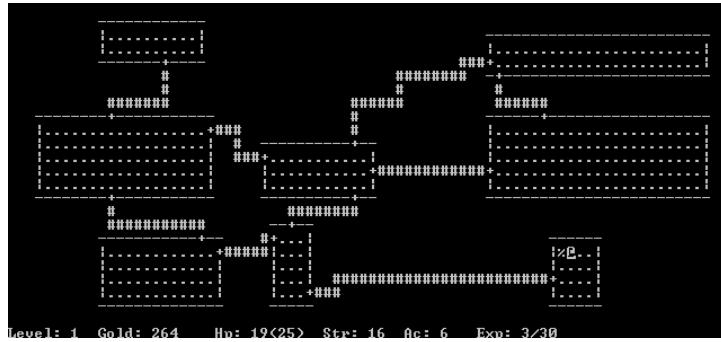
Com isto, é evidente que um jogador é bem-sucedido se a observação do ângulo ao qual a bola se está a direcionar for perspicaz, sendo recompensado por prenúncios corretos da posição vertical da barra. No caso do nosso jogo *Dung!* queremos incorporar uma jogabilidade semelhante, ignorando quaisquer tipos de movimentos e focando na percepção rítmica das colisões, reflexões e velocidades da bola.



**Figura 31:** Jogo *Pong*

Considerando as condições apresentadas, o jogo *Dung!* incorpora duas personagens, um herói controlado pelo jogador e um inimigo controlado pelo programa (computador). Um poder mágico (bola) é lançada contra o jogador pelo inimigo e este tem como objetivo de o refletir no momento em que se encontra perto de si, simulando algo como uma batida inicial de um jogo de *baseball*. Caso o jogador falhe a execução dessa ação, é deduzido um ponto. Caso seja sucedido, o inimigo perde um ponto de vida assim que o projétil entrar em contacto consigo.

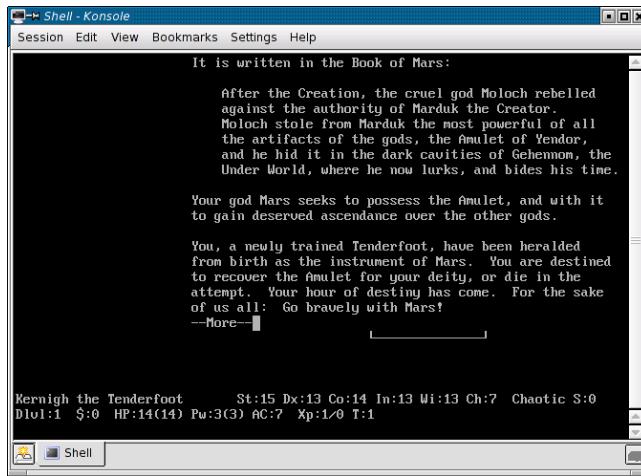
Com a mecânica principal do jogo estabelecida e considerando o tipo de experiência que queremos transmitir ao utilizador, será necessária a invenção e implementação de mecânicas atuais ou reconstruídas. Com isto, foi decidida a implementação de recursos presentes em categorias *niche* de jogos de carácter *indie*: **rogue-like** e **wave-survival**.



**Figura 32:** Exemplo dos primeiros jogos *rogue-like*

O género de jogos *rogue-like* surge de um dos jogos mais clássicos nativos a plataformas *Unix* e extremamente popular e inspirador para muitos jovens que cresceram nos anos 80, de título *Rogue*. As escolhas de *design* que foram tomadas e que estão diretamente relacionadas com recursos dispostos no jogo *Rogue* são as seguintes:

1. *Perma-death*: se o jogador perder todas as suas vidas, o jogo acaba e o utilizador precisa de reiniciar a sua partida, fazendo com que as decisões tomadas sejam valiosas e importantes.
2. Níveis infinitos: considerando condições impostas pelo programa, o jogo gera um novo nível sempre que o corrente é completado;
3. Heróis/classes diversas: na sequela do jogo *Rogue* de título *NetHack*, que aumenta o nível de complexidade do tema, apresenta a possibilidade do jogador escolher uma *role*, ou seja, um tipo de uma gama de classes que apresentam condições de jogo, estatísticas ou interações com o ambiente diferentes das restantes.



**Figura 33:** Placa de circuitos final da consola

O género de jogos *wave-survival* é principalmente conhecida pelos modos de jogo *Zombies* presentes na série de *shooters* de primeira pessoa de título *Call of Duty*, incorporada inicialmente no lançamento da versão de 2008 *World at War*. As escolhas de *design* que foram tomadas e que estão diretamente relacionadas com recursos dispostos na série de jogos *Call of Duty: Zombies* são as seguintes:

1. Rondas: os níveis do jogo são representados por rondas, onde são criados obstáculos (inimigos) que o jogador tem de derrotar. O nível de complexidade aumenta exponencialmente com a completação de cada ronda;
2. *Upgrades*: com o decorrer dos níveis o jogador pode colecionar tesouros encontrados que o podem ajudar ofensivamente ou defensivamente;

Com a junção das características mencionadas, o produto final resulta num jogo de ritmo acelerado onde existem vários heróis diferentes com características diferentes (número de vidas, ataque especial, etc.) que irão batalhar contra monstros, sendo que estas criaturas aparecem por rondas. Cada ronda tem um conjunto de criaturas normais e, no final, um monstro mais poderoso que, se os derrotarem, existe uma chance de ser recompensado com um item. Os items têm carácter defensivo ou ofensivo, podendo recolher itens como poções ou armas. As rondas de criaturas comuns e *bosses* são infinitas e o jogador perde, e reinicia a partida se assim o desejar, caso as vidas da sua personagem esgotem!



**Figura 34:** Ecrã inicial do jogo

### 3.3.2 Arquitetura

Tendo optado por usar o microcontrolador *Arduino Nano*, naturalmente o desenvolvimento de qualquer aplicação seria feito no Arduino IDE, que usa a linguagem de programação C++ com adição de métodos e funções especiais. As aplicações escritas interagem com o microcontrolador, dependendo de um conjunto de instruções apresentadas, que, por vezes, sem a utilização de bibliotecas externas para realizar interações com componentes, tornam-se bastante extensas. Com isto, é tomada a decisão de incorporar a biblioteca *Arduboy*, que disponibiliza recursos necessários para desenvolvimento de todos os aspetos de um jogo *retro* com as metas previamente definidas.

#### Inicialização

Com a adoção da biblioteca *Arduboy* e instanciação da variável correspondente é possível dar inicio à criação do *sketch*. Um projeto *Arduino* inicializa-se no método **setup** que, com a utilização da biblioteca importada, vai consistir no seguinte conjunto de instruções:

```

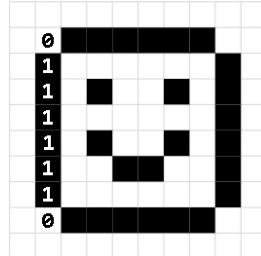
1 void setup(){
2     arduboy.beginDoFirst();
3     arduboy.initRandomSeed();
4     sound.tones(score);
5 }
```

**Listagem 1:** Função **setup** para inicializar o programa

O método `beginDoFirst` inicializa a instância da variável da biblioteca *Arduboy*, permitindo a utilização dos métodos disponibilizados. Normalmente o começo discrito é realizado através da função `begin`, sendo que a única diferença é que o posterior apresenta um ecrã de propaganda ao produto *Arduboy*. Posteriormente, é definida uma `seed` com o método `initRandomSeed`, sendo que esta é responsável pela randomização dos números gerados pelo programa, futuramente utilizados para criação de eventos de chance ou definição de ambientes variados. Finalmente, é utilizado o objeto `sound` que utiliza os recursos da biblioteca *ArduboyTones* para tocar o tema do jogo.

## Gráficos

Com a incorporação dos métodos de representação gráfica da biblioteca *Arduboy*, a disponibilização dos *sprites* do jogo, representantes das componentes do ambiente, torna-se bastante fácil. Os gráficos são instânciados através de *bitmaps*, conjuntos de bits espaçados em *bytes*. Os *bytes* deste conjunto representam uma coluna vertical onde o bit menos significativo corresponde ao bit do topo da coluna. Os bits colocados a 1 são representados com uma cor definida pelo programador (preto ou branco) e os restantes são ignorados, tendo em conta as dimensões introduzidas na função de desenho `drawBitmap`. A ordem dos bits a serem ignorados por parte desta função é decrescente.



**Figura 35:** Exemplo de um *sprite* desenhado usando a função `drawBitmap`

É importante notar a existência de um ficheiro complementar denominado de *sprites.c* que é importado para o projeto e contém todos os *bitmaps* necessários para representação. Caso o objeto a ser representado não seja estático, este contém variáveis de posição que podem ser alteradas durante os ciclos de jogo.

```
1 arduboy.drawBitmap(spellX, FLOOR_LEVEL - 9, spell, 10, 3);
```

**Listagem 2:** Função `drawBitmap` utilizando variáveis de posição

Neste caso o programa irá representar no ecrã da consola um poder mágico. O bitmap de tamanho 10 por 3 píxeis que o representa é definido na variável `spell` e este gráfico vai ser representado na posição (`spellX`, `FLOOR_LEVEL - 9`), sendo que o eixo horizontal pode ser deslocado e o eixo vertical é constante.

Os *sprites* dos heróis, inimigos e itens são importados (e alterados caso necessário) de um *tileset* denominado de *Fantasy 1-bit* proveniente do repositório do utilizador *tazmoe* existente na plataforma *Itch*, onde são disponibilizados conteúdos destinados a jogos de borla ou sobre um serviço de monetização, protegidos sobre uma lei de direitos de autoria que permite a utilização, produção e venda de aplicações que disponibilizem os recursos obtidos.



**Figura 36:** Tileset com os *sprites* utilizados

O *tileset* apresentado encontra-se numa grelha de blocos de 10 por 10 píxeis, excluindo os monstros maiores que ocupam 4 blocos, que em conjunto representam um *sprite* de 20 por 20 píxeis. Gráficos com âmbito informativo ou de seleção de opções ou atividades são desenhados pela equipa. Para estabelecimento do ambiente gráfico do jogo, os blocos de gráficos necessários foram convertidos para *bitmaps* e posteriormente importados para a aplicação, representados no ficheiro **sprites.c**.

### Personagens

No *Dung!* existem 3 opções de heróis, onde é necessária a escolha de uma destas personagens, influenciando o resto da partida. Os heróis têm estamina, que é gasta quando atacam e recuperada com o decorrer do tempo, e têm pontos de ataque especial, que são ganhos ao infligir dano em criaturas e gastos em grupos de 6 para executar ataques poderosos.



**Figura 37:** Ecrã de escolha de herói

Apesar disto, os heróis disponibilizados também apresentam diferenças nas 3 estatísticas e habilidades de jogo: **número de vidas, poder de ataque e ataque especial**.

	Vidas	Poder	Especial
Guerreiro	6	2	Barreira refletora
Arqueiro	4	3	Flecha perfurante
Mago	2	4	Sanguessuga

**Tabela 1:** Estatísticas dos heróis

Como demonstrado na tabela anterior, os heróis apresentam diferenças entre si. As vidas do jogador representam o número de vezes que um projétil pode colidir com o seu herói e o poder de ataque representa o dano que o herói vai inflingir nos monstros que combater, ou seja, a quantidade de pontos de vida que vai retirar ao contador do inimigo. Por outro lado, as mecânicas dos ataques especiais diferem bastante de personagem para personagem e permitem atribuir um papel específico ao herói:

1. Barreira refletora: o guerreiro cria um escudo à sua volta que o protege e reflete projéteis que colidam com ele;
2. Flecha perfurante: o arqueiro dispara uma flecha que, ao colidir com projéteis, arrasta-os no seu caminho até ao inimigo. Caso não entre em contacto com nenhum projétil no seu caminho, este ataque especial faz mais dano.
3. Sanguessuga: O mago lança um feitiço que, ao colidir com o inimigo, aplica dano e recupera um ponto de vida. Caso o feitiço entre em contacto com um projétil o mago fica exausto, perdendo toda a sua estamina e impossibilitando a recuperação desta durante 2 segundos;

Os heróis são definidos no programa através de uma classe `Hero`, que será extensida, dando origem às classes específicas das personagens `Swordsman`, `Archer` e `Mage`, sendo que estas classes sobreponem a informação de estatísticas, habilidades e gráficos específicos a cada herói.

```

1 class Hero{
2     public:
3         byte maxHealth, health, basicAttack, attack, arrowX,
4             spellX, specialPoints;
5         uint8_t* weapon1; uint8_t* weapon2; uint8_t* weapon3;
6             uint8_t* weapon4; uint8_t* weapon5;
7         uint8_t* item1; uint8_t* item2;
8         boolean usedSpecial = false, item1Def = false, item2Def
9             = false;
10    virtual void draw();
11    virtual void useSpecial();
12    virtual heroType getType();
13};

```

**Listagem 3:** Classe `Hero`

As características apresentadas atribuem papéis diferentes a todas as personagens, sendo cada uma relevante para um certo tipo de jogabilidade, partindo da dominância e aprendizagem do jogo por parte do utilizador. O *design* das personagens resultou do seguinte planeamento:

- Guerreiro: personagem inicial, para jogadores principiantes. Tem bastante margem de erro com 6 vidas e ataque especial defensivo mas acaba por ficar aquém devido ao reduzido poder de ataque;
- Arqueiro: personagem de dificuldade intermédia e balançada, o poder de ataque já é superior ao do guerreiro, ao custo de 2 vidas. Ataque especial bastante poderoso mas requer concentração e bom *timing*;
- Mago: personagem *glass-cannon* (imenso poder ao custo de estatísticas defensivas drasticamente reduzidas). Ataque especial permite que um utilizador experiente consiga revirar a partida, recuperando uma vida perdida de cada vez, ou falhar imensamente caso execute o feitiço incorretamente;

### Progressão e Inimigos

O jogo *Dung!* é composto por rondas, sendo que estas agrupam pequenos níveis onde o utilizador tem de ultrapassar e derrotar criaturas comuns e, quando estas estiverem consideradas vencidas, surge um monstro mais poderoso, de estatísticas superiores e exponencialmente crescentes ao longo do decorrer da partida. Se este for derrotado com sucesso, a ronda dá-se como terminada e o herói do jogador é transportado para a seguinte.



**Figura 38:** Ecrã de jogo

Com o desenrolar dos desafios apresentados ao utilizador vão exister duas variáveis que se vão tornar superiores, sendo estas as seguintes:

1. Número de criaturas comuns até ao *boss*: `número de ronda corrente + 2`, para que haja uma progressão linear (ex. ronda 3 tem 5 criaturas e 1 boss);
2. Pontos de vida das criaturas comuns: `2 * número de ronda corrente + nível na ronda`, para que haja uma progressão entre criaturas dentro de uma única ronda (ex. ronda 3 e já derrotou 3 criaturas, significando que está no 4º nível dentro da 3ª ronda, então a 4ª criatura vai ter 10 de vida);
3. Pontos de vida de *bosses*: `2 * número de ronda corrente + nível na ronda + 6`, realizando a progressão de uma criatura normal, adicionando 6 pontos de vida para se destacar das restantes;

É de notar que existem vários tipos monstros, que são os seguintes: **pântano**, **morto-vivo**, **ancião**, **fogo**, **floresta** e **normal**. O tipo de monstro é definido no início de cada ronda e não influencia diretamente estatísticas dos monstros, apenas a sua representação gráfica.

Com isto, é necessário também implementar um sistema de aumento de velocidade do projétil corrente no ambiente que complemente a evolução estatística das criaturas com que o jogador se depara. Este aumento vai obrigar o utilizador a focar a sua atenção total no movimento dos projéteis, que inicialmente apresentam velocidade de 1 pixel por frame. Por cada ronda, a variável do programa correspondente à velocidade é incrementada por 0.2 e, em prática, os valores da posição do projétil são calculados através desta variável arredondada para baixo.

Valor calculado	0	1.2	2.4	3.6	4.8	6
Píxel	0	1	2	3	4	6

**Tabela 2:** Posição do projétil ao longo de 5 frames com velocidade de 1.2 píxeis

Na tabela apresentada é notado um exemplo onde o utilizador se encontra na 2<sup>a</sup> ronda do jogo e o projétil a ser disparado tem velocidade de 1.2 píxeis. No caso do projétil se mover do pixel 0 ao pixel 6, este nunca passa pelo pixel 5.

```

1 void nextLevel(bool bossLvl){
2     sc = game;
3     level++;
4     getRandomEnemy(bossDefeated, bossLvl);
5     if(bossDefeated){
6         level = 1;
7         gameRound++;
8         if(projectileSpeed < 3)projectileSpeed += 0.2;
9         sc = roundOver;
10        bossDefeated = false;
11        itemFound = getRandomItem();
12    }
13    projectileX = enemyLeft;
14    goToPlayer = true;
15    currentStamina = REQUIRED_STAMINA;
16    hero->arrowX = 0;
17    hero->usedSpecial = false;
18    maxEnemyHealth = 2*gameRound + level;
19    if(bossLvl)
20        maxEnemyHealth += 6;
21    enemyHealth = maxEnemyHealth;
22    frame = 0;
23 }
```

**Listagem 4:** Função executada após passagem de nível

## Itens

Ao derrotar um *boss* de uma certa ronda, o jogador pode ser recompensado com itens que o podem ajudar na sua aventura, mas apenas pode carregar 2 de uma única vez. Estes itens são separados em três categorias: **armas**, **gemas** e **poção**. As armas encontradas aumentam o poder de ataque do jogador contra todas as criaturas por  $1 \text{ valor} * \text{nível da arma}$ . Visto que existem 5 *tiers* de armas possivelmente encontradas no jogo a arma mais poderosa aumenta o poder de ataque da personagem por 5 valores.



**Figura 39:** Ecrã onde se equipam os itens

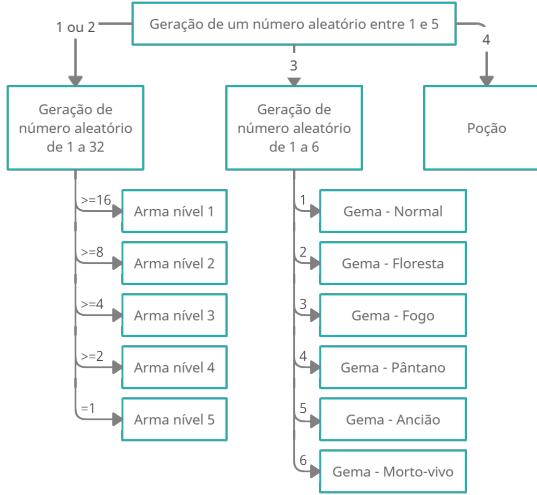
Existem também várias gemas, uma por cada cada tipo de monstro e todas de carácter ofensivo. Uma gema equipada permite que o herói do jogador duplique os seus pontos de ataque, quando em combate com um monstro do mesmo tipo. Este tipo de item é incentivado caso o jogador se esteja a sentir sortudo, pois a chance de o tipo de gema equipado corresponder com o tipo de monstro da ronda corrente é  $1/6$ . Caso esta eventualidade aconteça, o jogador consegue aplicar o dobro do dano e acabar a ronda rapidamente.

```
1 void updateAttack(){
2     hero->attack = hero->basicAttack;
3     if(hero->item1Def && getGemType(hero->item1) == enemyType)
4         hero->attack *= 2;
5     if(hero->item2Def && getGemType(hero->item2) == enemyType)
6         hero->attack *= 2;
7     if(hero->item1Def){
8         addWeaponTierStats(hero->item1);
9     }
10    if(hero->item2Def){
11        addWeaponTierStats(hero->item2);
12    }
13 }
```

**Listagem 5:** Atualização do ataque dependendo dos itens equipados

No caso de o jogador ser recompensado com uma poção, este pode ingeri-la para ser recompensado com 1 ponto de vida, sendo o único item de carácter defensivo.

Todos os itens mencionados são atribuídos ao derrotar um *boss*. O item escolhido é gerado através de um algoritmo de seleção que permite o programa escolher uma categoria de item e, de seguida, o item específico.



**Figura 40:** Algoritmo para a escolha de itens

Com a figura anterior é possível verificar que inicialmente é gerado um número de 1 a 5 para decidir a categoria de um item a ser atribuído ao jogador. Considerando que todos os números menos o número 5 estão atribuídos a uma determinada categoria, se este for produzido o jogador não recebe recompensa. É também possível admitir que a distribuição de probabilidades das armas inicialize-se no nível 1 com 50% de chance e decresce com o aumento de cada nível. As gemas, por outro lado, têm probabilidades constantes.

```

1 bool getRandomItem(){
2     randomSeed(arduboy.generateRandomSeed());
3     long itemType = random(1, 6);
4     uint8_t* item;
5     switch(itemType){
6         case 1:
7             case 2: return getRandomWeapon();
8             case 3: return getRandomGem();
9             case 4:
10                rewardName = F("HEALTH POTION");
11                rewardedItem = healthPotion;
12                return true;
13            case 5: return false;
14        }
15    }
  
```

**Listagem 6:** Função de geração de itens

## Pontuação

No Dung!, a consola guarda na sua EEPROM as três melhores pontuações. Cada pontuação corresponde ao número de rondas derrotadas e são definidas quando o jogador esgota todas as vidas da sua personagem.

```
1 if(hero->health == 0){  
2     highscoreAddr = checkForHighscore();  
3     if(highscoreAddr > 0) sc = highscore;  
4     else sc = over;  
5 }
```

Listagem 7: Verificação de *highscore* conquistado

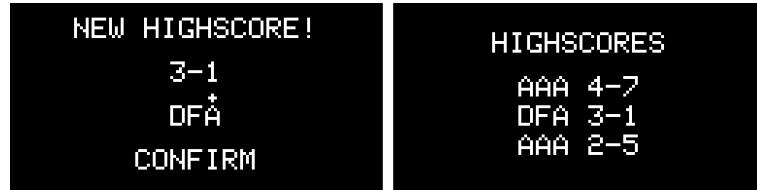
Na listagem anterior é apresentada a condição de verificação de nova pontuação recorde. Quando a partida é terminada, é verificado se a pontuação conquistada é melhor que uma das três registadas. Em caso positivo, a nova pontuação é inserida na posição correspondente.

```
1 byte checkForHighscore(){  
2     byte addr = EEPROM_STORAGE_SPACE_START;  
3     for(byte i = addr; i <= addr + 10; i += 5){  
4         byte eepromRound = EEPROM.read(i);  
5         byte eepromLevel = EEPROM.read(i+1);  
6         if(eepromRound == 255 || gameRound > eepromRound ||  
7             (gameRound == eepromRound && level > eepromLevel)){  
8             if(i == addr){  
9                 EEPROM.write(addr + 10, EEPROM.read(addr + 5));  
10                EEPROM.write(addr + 11, EEPROM.read(addr + 6));  
11                EEPROM.write(addr + 12, EEPROM.read(addr + 7));  
12                EEPROM.write(addr + 13, EEPROM.read(addr + 8));  
13                EEPROM.write(addr + 14, EEPROM.read(addr + 9));  
14                EEPROM.write(addr + 5, EEPROM.read(addr));  
15                EEPROM.write(addr + 6, EEPROM.read(addr + 1));  
16                EEPROM.write(addr + 7, EEPROM.read(addr + 2));  
17                EEPROM.write(addr + 8, EEPROM.read(addr + 3));  
18                EEPROM.write(addr + 9, EEPROM.read(addr + 4));  
19             }else if(i == addr + 5){  
20                 EEPROM.write(addr + 10, EEPROM.read(addr + 5));  
21                 EEPROM.write(addr + 11, EEPROM.read(addr + 6));  
22                 EEPROM.write(addr + 12, EEPROM.read(addr + 7));  
23                 EEPROM.write(addr + 13, EEPROM.read(addr + 8));  
24                 EEPROM.write(addr + 14, EEPROM.read(addr + 9));  
25             }  
26         }  
27     }  
28 }
```

Listagem 8: Alteração da ordem de *highscores*

A listagem anterior apresenta a função de alteração de *highscores* no *EEPROM* do microcontrolador através da biblioteca do *Arduino*. O *EEPROM* é uma memória que pode ser reescrita elétricamente.

Com a utilização da biblioteca mencionada são acedidos a vários endereços localizados a partir do *EEPROM\_STORAGE\_SPACE\_START*, um local recomendado pela comunidade *Arduboy* para armazenamento de dados, visto que o sistema operativo ocupa bastante da memória disponibilizada. São utilizados 15 *bytes* começando pelo local especificado e são utilizados 5 *bytes* por entrada de pontuação onde o 1º *byte* corresponde à ronda alcançada, o 2º *byte* corresponde ao nível respetivo, e os últimos 3 *bytes* correspondem ao nome do jogador representado por três letras.



**Figura 41:** Ecrã de registo de *highscore*

A função *checkForHighscore* compara e verifica três situações que podem ter resultados diferentes dependendo dos valores retornados após a leitura sequencial de todas as posições de registo nos endereços dedicados:

1. Se os endereços estão vazios (*eepromRound* == 255);
2. Se a ronda corrente é maior que a ronda analisada (*gameRound* > *eepromRound*);
3. Se a ronda corrente for igual à ronda analisada mas o nível corrente for maior que o nível correspondente;

Em qualquer uma das situações verificadas é retornado o índice onde a condição se verifica e a tabela é restrukturada caso a posição analisada encontre-se em primeiro ou segundo lugar (*shift* das posições restantes). Caso nenhuma destas eventualidades se suceder é retornado o valor 0, significando que a pontuação obtida é menor que todas as posições atualmente guardadas no *EEPROM*. O valor verificado permite a escritura dos dados necessários para criar a entrada na tabela.

```

1 EEPROM.write(highscoreAddr, gameRound);
2 EEPROM.write(highscoreAddr+1, level);
3 EEPROM.write(highscoreAddr+2, alphabet[letter1]);
4 EEPROM.write(highscoreAddr+3, alphabet[letter2]);
5 EEPROM.write(highscoreAddr+4, alphabet[letter3]);

```

**Listagem 9:** Escrita de uma nova pontuação

## Controles

A placa de circuito desenvolvida e apresentada em capítulos anteriores disponibiliza três botões: A, B e C, que, no ambiente de jogo, realizam *inputs* necessários para interações entre utilizador e o sistema. O *Arduboy*, por outro lado, está configurado para suportar até 6 *inputs*. Os botões são configurados diretamente nos ficheiros *Arduboy2Core.cpp* e *Arduboy2Core.h* da biblioteca adotada.

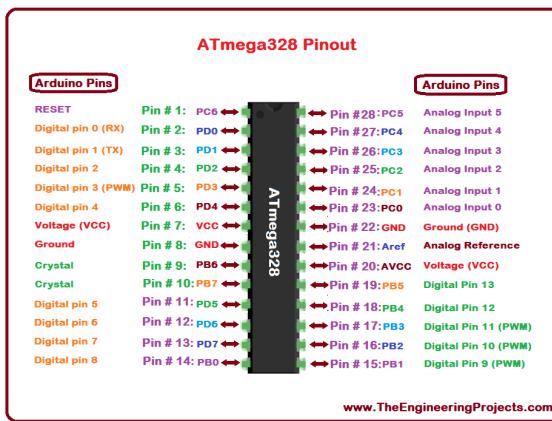
```

1 #ifdef SLIMBOY
2   PORTC |= _BV(LEFT_BUTTON_BIT) | _BV(UP_BUTTON_BIT);
3   DDRC &= ~(_BV(LEFT_BUTTON_BIT) | _BV(UP_BUTTON_BIT));
4   PORTD |= _BV(RIGHT_BUTTON_BIT) |
5           _BV(DOWN_BUTTON_BIT) | _BV(A_BUTTON_BIT) |
6           _BV(B_BUTTON_BIT);
7   DDRD &= ~(_BV(RIGHT_BUTTON_BIT) |
8             _BV(DOWN_BUTTON_BIT) | _BV(A_BUTTON_BIT)) |
9             _BV(B_BUTTON_BIT);

```

**Listagem 10:** Definição dos botões nas portas do chip ATMEGA328

Devido à esquemática elétrica desenhada e interligações entre componentes estabelecidos, é claro admitir que os botões da consola apresentam-se todos conectados a pinos digitais. É necessário definir essa eventualidade, visto que a biblioteca original inclui o botão B como um botão analógico.



**Figura 42:** Correspondência entre pinos do Arduino e portas do ATMEGA328

Considerando a figura anteriormente apresentada, que remete para o *chip* do microcontrolador adotado, são apresentados todos os pinos digitais como membros da notação *PORTD*, e todos os pinos analógicos como *PORTC*. A inclusão do *B\_BUTTON\_BIT* na variável *PORTD*, demonstrado na listagem exibida, vai mapear corretamente o *input* proveniente do pino definido no ficheiro *Arduboy2Core.h*.

```

1 #define PIN_A_BUTTON 6
2 #define A_BUTTON_PORT PORTD
3 #define A_BUTTON_PORTIN PIND
4 #define A_BUTTON_DDR DDRD
5 #define A_BUTTON_BIT PORTD6

```

**Listagem 11:** Definição do pino e portas do botão A

Com isto, é possível definir as funcionalidades que os *inputs* dos três botões configurados, que, dependendo da instância do ambiente onde estão colocados correntemente, vão realizar funções desiguais:

	Menus	Jogo
A	Esquerda	Refletir
B	Selecionar/Confirmar	Ataque especial
C	Direita/Voltar	Usar poção

**Tabela 3:** Funções dos botões

Como todos os *inputs* realizados pelo utilizador, o lançamento de um ataque especial é processado na função `loop` do *Arduino*, onde a função `arduboy.pollButtons` é chamada a cada *frame* e permite ler e guardar o estado dos botões. As funções `justPressed` e `justReleased` verificam se o estado dos botões foi alterado entre chamadas consecutivas, com o objetivo de descobrir se os botões acabaram de ser pressionados ou libertos.

```

1 if(arduboy.justPressed(B_BUTTON)){
2     if(currentStamina == REQUIRED_STAMINA)
3         hero->useSpecial();
4         if(playAudio) sound.tone(4000,50,2000,50,1000,50);
5 }

```

**Listagem 12:** Exemplo do uso da função `justPressed`

## Música e Efeitos Sonoros

A consola desenvolvida está equipada com um altifalante que pode ser operado com o auxílio da biblioteca *Arduboy*, que está preparada para manipular os valores de oscilação do circuito elétrico do *buzzer*, tendo em conta pares de frequência e tempo, correspondendo, respetivamente, ao tom representado sonoramente e a duração deste.

Na 4ª linha da LISTAGEM 12 é possível verificar a presença de uma função `tone`. Esta função, tendo em conta os parâmetros agrupados, vai tocar pares de frequência e tempo, neste caso 3 frequências diferentes (4000, 2000, 1000), com um intervalo de 50 milisegundos entre frequências. Os efeitos sonoros presentes no jogo são os seguintes:

Ação	Frequência (Hz)	Duração (ms)
Mudança de seleção em menus	4000	100
Confirmar herói/Reiniciar jogo	4000	100
	4500	100
	5000	100
	500	100
Projétil instânciado	800	200
Inimigo atingido	600	200
Jogador atingido	100	100
Poção usada	1000	200
Especial usado	4000	50
	2000	50
	1000	50
Jogador perde	300	200
	200	200
	100	200
Ronda terminada	3000	200
	3500	200
	4000	200

**Tabela 4:** Pares frequência-duração dos tons

Existe também uma música que toca quando o utilizador inicializa o programa. Esta música foi gravada através do *software* de produção e gravação de áudio *Fruity Loops Studio*. Com um instrumento base é tocada a melodia com as notas específicas requisitadas, num espaço de batimentos por minuto pré-definido. Estas notas são convertidas e gravadas num ficheiro *midi*, que, com o auxílio da ferramenta de conversão *midi2tones*, disponibilizada por *MLXXXP*, um utilizador dedicado da comunidade *Arduboy*, são convertidas para o formato apropriado.

### Ciclos

Com a apresentação das várias possíveis aventuras e eventualidades que o jogador pode participar é fácil admitir a existência de uma multitudine de ecrãs correspondentes à interação com uma ou mais atividades. Por exemplo, recorrendo a um dos últimos pontos previamente apresentados, as pontuações, são apresentados dois cenários:

- O utilizador quer verificar as pontuações atualmente presentes na tabela de classificações;
- O utilizador regista a pontuação conquistada na tabela de classificações caso tenha perdido a partida mas o resultado obtido é melhor do que pelo menos uma das entradas, escrevendo e associando um *alias* de três letras;

Não só ambas as possibilidades apresentam funcionalidades e respostas diferentes, como necessitam de verificação de componentes por parte do sistema, para que se possa fazer *trigger* de um evento e apresentar o ecrã estipulado. Para tal, é implementado um `enum` que define todos os ecrãs apresentáveis.

```

1 enum screen {start, choose, game, over, won, roundOver,
   equipItem, highscore};
2 screen sc = start;

```

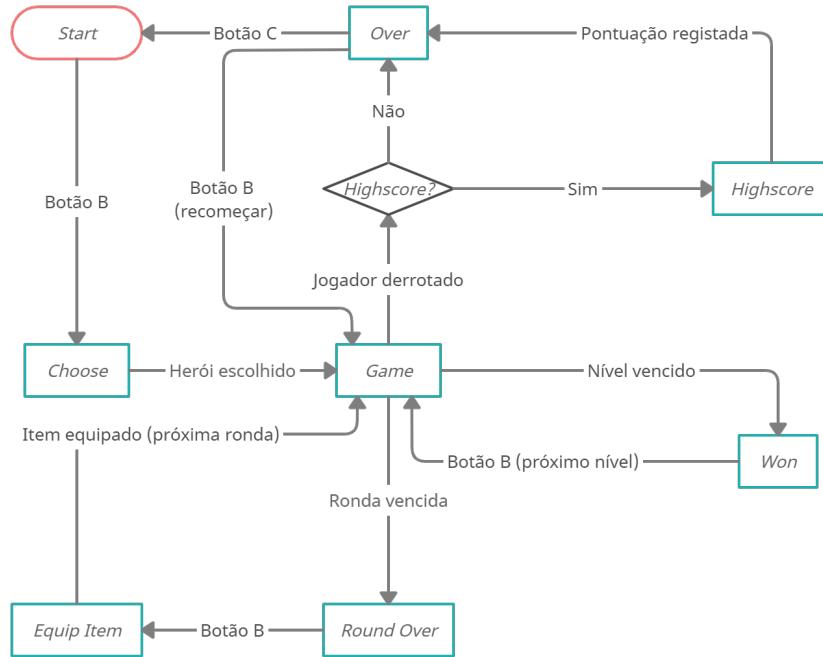
**Listagem 13:** Definição de ecrãs e a variável que guarda o atual

Cada entrada presente no enumerador `screen` da listagem anterior corresponde a um possível cenário onde o jogador é transportado caso cumpra as condições necessárias. As descrições das finalidades dos ecrãs presentes no projeto são as seguintes:

Nome	Descrição
Start	Logotipo do jogo, opções de som e acesso à tabela de pontuações
Choose	Opções de heróis, escolha de classe
Game	Decorrer de eventos do jogo
Over	Ecrã final do jogo, pode reiniciar ou voltar ao ecrã inicial
Won	Inimigo derrotado com sucesso
Round Over	Monstro final derrotado
Equip Item	Item obtido e opções de equipamento
Highscore	Possibilita o registo da pontuação obtida

**Tabela 5:** Ecrãs e as suas funções

O fluxo entre ecrãs definidos por entradas presentes no enumerador apresentado na listagem anterior é definido pelo seguinte fluxograma:



**Figura 43:** Fluxograma dos vários ecrãs

As decisões e condições necessárias para passagem entre blocos são verificadas no ciclo *loop* de função nativa do *Arduino*. No início de cada ciclo é realizado um *switch* que determina o ecrã corrente, onde cada caso define as funcionalidades, aplicações e circunstâncias de alteração caso os requisitos sejam cumpridos.

```

1 case over:
2   printCentered(10, F("GAME OVER"));
3   printCentered(30, F("PRESS B TO RESTART"));
4   printCentered(40, F("PRESS C TO GO BACK"));
5   if(arduboy.justPressed(DOWN_BUTTON)){
6     reset();
7     sc = game;
8   }else if(arduboy.justPressed(B_BUTTON)){
9     sc = start;
10 }
11 break;
  
```

**Listagem 14:** *case* do ecrã *over*

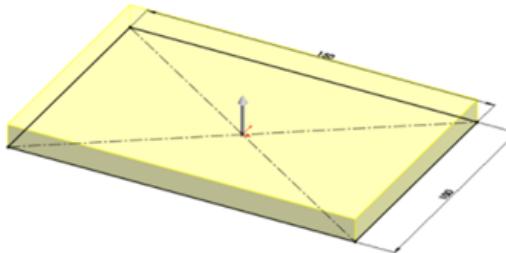
## 3.4 Encapsulamento

Sustenta-se que o objetivo da constituição da componente de encapsulamento consiste na criação de uma caixa que agrupe a placa de circuitos desenvolvida, juntamente com todos os componentes e módulos conectados, protegendo-os de condições exteriores e permitindo que o utilizador experiencie o jogo desenvolvido e outros profissionalmente.

### 3.4.1 *SolidWorks*

Sendo o nosso objetivo a construção de uma caixa que encapsule a placa eletrónica, é necessário uma ferramenta de *software* que permita a modelação de objetos 3D complexos, sem uma curva de aprendizagem que dificulte a criação de metas definidas. Com isto, foi adotado para o projeto a ferramenta *SolidWorks*, um *software* de CAD (*Computer-Aided Design*) que se foca na modelação 3D dando ênfase na fidelidade física dos objetos desenvolvidos.

Este programa utiliza modelação paramétrica, significando que utiliza parâmetros como a dimensão para definir o modelo. Com este tipo de modelação os parâmetros podem ser alterados posteriormente e o modelo será atualizado para refletir as mudanças efetuadas. Para facilitar o processo existem também restrições, relações entre as entidades que compõem o modelo. Por exemplo, ao desenhar um ecrã, os lados podem ser definidos como tendo o mesmo tamanho. Este tipo de modelação foi uma mais-valia para o nosso projeto, pois visto que o desenvolvimento da caixa foi realizado em paralelo com o *design* final da placa eletrónica, durante o processo criativo as alterações ao modelo seriam inevitáveis.

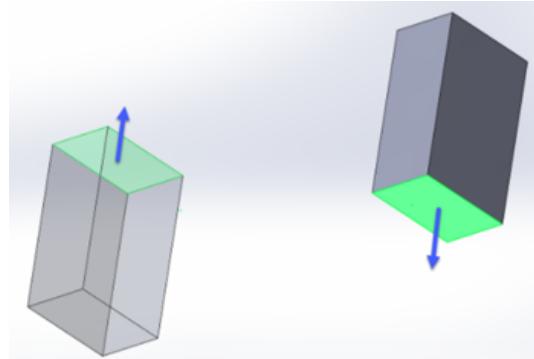


**Figura 44:** Design 3D a partir de um esboço 2D

Com a inicialização da construção de uma peça no SolidWorks, tipicamente o passo inicial será um esboço 2D que defina a forma básica e o tamanho do modelo. Posteriormente serão utilizadas variadas ferramentas para modelar o esboço num espaço 3D até este apresentar a forma desejada.

Considerando que a caixa a desenvolver é composta por diferentes peças, será essencial que o programa escolhido permita a montagem de um modelo composto por várias partes. O *SolidWorks* integra esta funcionalidade com base numa conduta de fácil aprendizagem, integrando um modo de desenvolvimento dedicado a este tema.

Cada parte projetada pode ser contida num ficheiro separado, o que facilita a organização do projeto. Para a montagem das peças é apenas necessário selecionar o modo que o permite (*Assembly*) e escolher os ficheiros que contém as partes pretendidas. O processo de montagem também é simplificado, sendo que o utilizador apenas necessita de selecionar as faces onde as partes a juntar se conectam.



**Figura 45:** Exemplo de uma montagem com duas partes

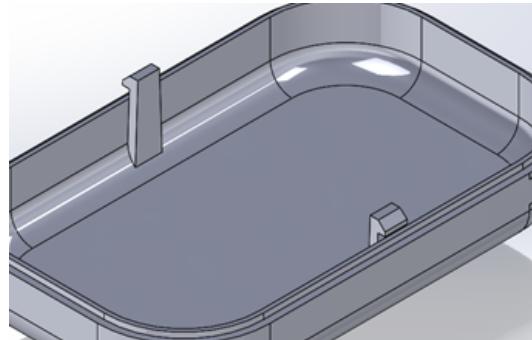
### 3.4.2 Caixa

#### Tentativa

O processo de criação da caixa da consola é inicializado definindo que a mesma seria composta por duas partes, que encaixariam uma por cima da outra, e a placa de circuitos inserida entre ambas. Seria também necessário a aplicação de cortes nas superfícies laterais, de modo a permitir a inserção de componentes externos como o cabo de carregamento da consola, assim como facilitar o acesso aos componentes internos como o ecrã. Numa primeira iteração o encaixe mencionado foi definido com uma funcionalidade denominada de *Snap Hook*, onde a parte superior disponibilizaria um gancho que encaixasse numa secção da encapsulação inferior.

Devido à pesquisa realizada previamente à passagem de uma fase de projeção para o sector de produção foi decidida a remoção deste recurso devido aos seguintes factores considerados:

1. Dado as dimensões da consola, os ganchos utilizados para o encaixe seriam bastante pequenos;
2. O material usado para a impressão, PLA (Poliácido láctico), não exibe elasticidade suficiente para a ação de desencaixe;
3. A impressão 3D é executada por camadas;

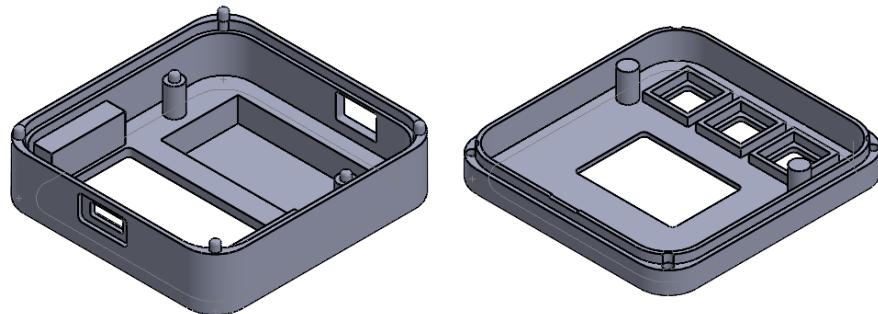


**Figura 46:** Exemplo de um encaixe *Snap Hook*

Considerando os factores mencionados, é convincente que a produção de tal modelo iria resultar num produto frágil, e, consequentemente, falhado. A imprecisão de impressão de tamanhos bastante reduzidos por meio de uma máquina que não apresenta qualidade industrial e a fragilidade do material utilizado removem a possibilidade da utilização da mecânica mencionada.

#### Finalização

Nesta iteração final foi adotado um *design* que mantesse as partes superior e inferior conectadas. Foram realizadas extrusões nos cantos da parte inferior da caixa que encaixam em buracos laminados na parte superior. Para reforçar a junção de ambas as peças foi utilizada a ferramenta *Lip/Groove* que cria uma cavidade na borda de uma das peças e uma extrusão na outra, o que facilita a união destas.

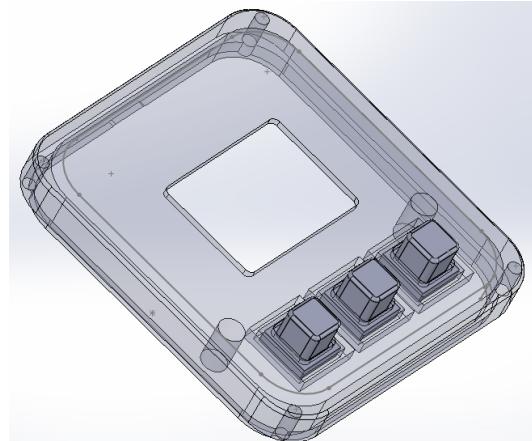


**Figura 47:** Caixa da consola

Na figura anterior também se pode observar que, de modo a segurar a placa eletrónica à distância correta para que os componentes fiquem colocados na posição

projetada, foram realizadas duas extrusões que se inserem nos dois buracos presentes na placa de circuitos adotada. As perfurações exigidas no *design* do circuito apresentam o propósito mencionado. Existem igualmente duas elevações, colocadas próximas do encaixe do microcontrolador. As extrusões da parte cima, em conjunto com o mencionado anteriormente, completamentam o encaixe e clausura da placa dedicada no meio da caixa.

Finalmente, será necessária a projeção dos botões, que têm de ser desenvolvidos separadamente. Esta separação permite a criação de botões que permitam criar elevação entre o interruptor tátil a pressionar e a caixa, visto que a altura que o encapsulamento enforça à consola é demasiado elevada. Com isto, os botões desenvolvidos são produzidos externamente e permitem o encaixe dentro do encapsulamento superior através de um anel, que permite a estabilização deste quando é pressionado até à sua libertação. O utilizador pode empurrar ligeiramente os botões projetados para que estes pressionem os interruptores soldados na placa de circuitos, de modo a realizar os *inputs* necessários.



**Figura 48:** Botões encaixados na parte cima da caixa

## 4 Resultado



**Figura 49:** Produto final

## 5 Conclusões

Concluíndo o projeto, com as componentes de *hardware*, *software* e encapsulamento projetadas e produzidas, dá-se como terminado todos os objetivos demonstrados no documento. Com isto, não ignoramos que tenham existido factores que foram decisivos para a finalização da concepção mencionada. Existiram também contrapartidas em ações tomadas vista a natureza do projeto, e problemas que, como um grupo, tivemos de tomar uma posição anteriormente nunca aplicada. Redirecionamos esta secção do projeto para descrever os pontos que a equipa determina como importantes a notar.

### 5.1 Equipa

Finalizando o projeto, sentimos-nos satisfeitos com o produto final criado, sendo que a nossa visão foi bem-sucedida. A experiência e conhecimentos estudados no decorrer do nosso curso foi principalmente focada no desenvolvimento de *software*, com grande foco no *design* de multimédia, especialmente em ambientes 3D.

Com isto, admitimos a reduzida preparação para a realização do projeto estabelecido, considerando a abundância de conhecimentos e vertentes necessárias a serem estudadas e aplicadas para a execução das metas definidas. Sendo que a equipa, durante a atividade, divagou em ambientes variados na vertente de engenharia e considerando o reduzido número de leções focadas no planeamento e fases de execução de projeto, contemplamos o produto final como um sucesso.

A informação a reter, aprender, relembrar de cadeiras anteriormente completadas e estudos necessários para o êxito na projeção realizada foi extensa. Houve complicações durante a projeção devido a fundamentos desconhecidos. As ferramentas de *software* que acompanhavam a matéria necessária para o *design* e aplicação das

noções mencionados eram de alto nível, comparando com ambientes de programação recentes, onde o nível de especificação e curva de aprendizagem são bastante reduzidas.

No inicio do projeto, complicações derivadas da indecisão e encontro de obstáculos, com reduzida visão para a sua resolução, levaram à um começo abruto. Ambos admitos ter *drive* para fazer, mas visto o reduzido nível de planeamento no ínicio do projeto, foram necessário serem tomadas precauções que modificaram o rumo do projeto - para melhor.

## 5.2 Projeto

Um dos grandes obstáculos na produção da componente de *hardware* foi o vírus *SARS-CoV-2* que, por consequência dos danos causados na sociedade, removeu bastantes privilégios que o grupo poderia ter usufruído durante a execução do projeto. A excisão da facilidade de contacto entre alunos e o docente responsável e engenheiros envolvidos, ou da encomenda de itens necessários para testes e planeamentos afetou bastante o desenvolvimento do projeto. Ensaios e tentativas inovadoras e arriscadas poderiam ter sido tomadas mas foram evitadas devido a problemas como, por exemplo, a não receção ou receção extremamente demorada de um item vital para o projeto.

O custo de peças, componentes, módulos e mesmo da impressão 3D foi bastante elevado, sendo que ambos os membros se juntaram para completar objetivos necessários. Isto poderia ter sido evitado caso a ajuda necessária fosse procurada.

### Circuito

A integração de luzes que indicassem visualmente o estado da bateria (carregada totalmente, parcialmente carregada, pouco carregada) e o estado do carregamento (bateria a carregar, bateria completamente carregada) que fossem visualizadas no exterior da consola seriam bastante úteis, visto que não existe a disponibilização de tal recurso no exterior da consola. Apesar disto, devido às capacidades do *chip* do módulo de carregamento, o sobre-carregamento da bateria está fora de questão.

### Caixa

Consideramos que o encapsulamento da consola seria capaz de apresentar uma melhor experiência para o utilizador, vistas as condições impostas mencionadas no capítulo adequado. Não considerando os botões, as entradas de *input* por parte do utilizador estão bastante subdesenvolvidas, vista a dificuldade de utilização das entradas dedicadas.

### **5.3 Final**

Avaliando o produto final, achamos que a ideia projetada encontra-se concluída e concebida. A equipa foi autónoma e decidina no desenvolvimento do projetos. Foram apresentados obstáculos e problemas durante a execução do projeto que a equipa necessitou de recorrer às bases da arquitetura do projeto e reformular. Concluíndo o projeto, admitisse que as decisões tomadas foram integradas de acordo o convencionado.

A integração de ambientes de caractér de *design*, ambos artístico com modelação 3D em ambientes gráficos e a projeção de circuitos e criação e estabelecimento de regras de conexão entre componentes e módulos, e aprendizagem e preceção de todos os ecossistemas explorados, é determinada bastante conveniente e acertada para o futuro de ambos os concorrentes. A interligação entre conhecimentos de componentes *hardware*, *software* e *design* artístico e de equipamento são temas bastante promissores.



## 6 Bibliografia

- [1] Arduboy2: An alternative library for the Arduboy miniature game system,  
<https://github.com/MLXXXp/midi2tones>
- [2] TP4056 datasheet,  
<https://dlnmh9ip6v2uc.cloudfront.net/datasheets/Prototyping/TP4056.pdf>
- [3] LP601730 datasheet,  
<https://www.tme.eu/Document/aa593083f76c72af8796398caaac30a8/ce10016.pdf>
- [5] SSD1306 datasheet,  
<https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf>
- [4] MIDI2TONES: Convert a MIDI file into a simple bytestream of notes,  
<https://github.com/MLXXXp/midi2tones>
- [5] Fantasy One Bit,  
<https://tazmoe.itch.io/fantasy-one-bit>
- [6] Ardbitmap: A library to compress and draw bitmaps on the Arduboy,  
<https://github.com/igvina/ArdBitmap>