
# Licenciatura em Engenharia Informática e Multimédia

## Desenvolvimento de Aplicações Móveis

Docente: Ana Correia

## Projeto Final

*2020/2021*

Aluno:

46338, José António Siopa

3 de julho de 2021

# Conteúdo

# Lista de Figuras

# 1. Introdução

No âmbito da unidade curricular de Desenvolvimento de Aplicações Móveis, foi solicitada a implementação e avaliação de um jogo em *Android* para dispositivos móveis, desde a fase de *design* até à fase de lançamento no mercado. O processo de desenvolvimento seguiu a metodologia centrada na experiência do utilizador, consistindo em quatro fases: planeamento, pré-produção, produção e pós-produção. Após finalizada, a aplicação foi avaliada em usabilidade, funcionalidade e modelo de negócios.

# 2. Planeamento

## 2.1 Conceito geral do jogo

O jogo a ser realizado tem como título "Cute Little Devil" e consiste num jogo plataforma de combate com armas corpo a corpo e com poderes de longo alcance. O jogador terá que enfrentar vários inimigos para conseguir completar o nível em que se encontra e desbloquear o próximo. Ao eliminar um inimigo, o jogador será recompensado com moedas e será possível usá-las na loja do jogo para comprar novas armas ou poderes. A quantidade de moedas obtidas será guardada na *Firebase* pelo que será necessário realizar *login* para conseguir comprar itens na loja.

## 2.2 Experiência pretendida

O objetivo será implementar no jogo um combate fluído e rápido, assim como a criação de um ambiente cativante, de modo que a experiência do utilizador seja satisfatória, divertida e desafiadora. Fazendo com que a aplicação esteja bem organizada e apelativa, irá motivar o utilizador a continuar a jogar o jogo várias vezes.

## 2.3 Características principais

Como características principais do jogo, este é do género plataforma 2D pixelizado, contendo combate corpo a corpo e de longo alcance. Possui itens coletáveis para, por exemplo, regenerar a vida, aumentar a força e moedas, para comprar armamentos melhores ou outras melhorias.

## 2.4 Regras do jogo

Relativamente às regras do jogo, não existe muita complexidade. O objetivo deste é completar cada nível chegando à parte final do mapa, onde haverá um baú que permitirá desbloquear o próximo nível. Para abrir este baú, é necessária uma chave que pode estar visível, escondida ou na posse de algum inimigo.

Existirão itens que permitem regenerar vida ou melhorar atributos temporariamente. A eliminação de um inimigo garante ao jogador moedas que pode utilizar na loja.

## 2.5 Público alvo

O público alvo será todos os utilizadores que procuram um passatempo divertido em forma de aplicação para dispositivos móveis.

## 2.6 Monetização

Na loja do jogo, existirá um poder e um novo personagem que apenas poderá ser obtido através de uma transação monetária.

## 2.7 Wireframes do jogo

Foram criados os *wireframes* dos ecrãs principais do jogo para planear a sua estrutura (Figura 2-1). Dentro de cada balão está qual o ecrã resultante do clique do botão a ele associado.

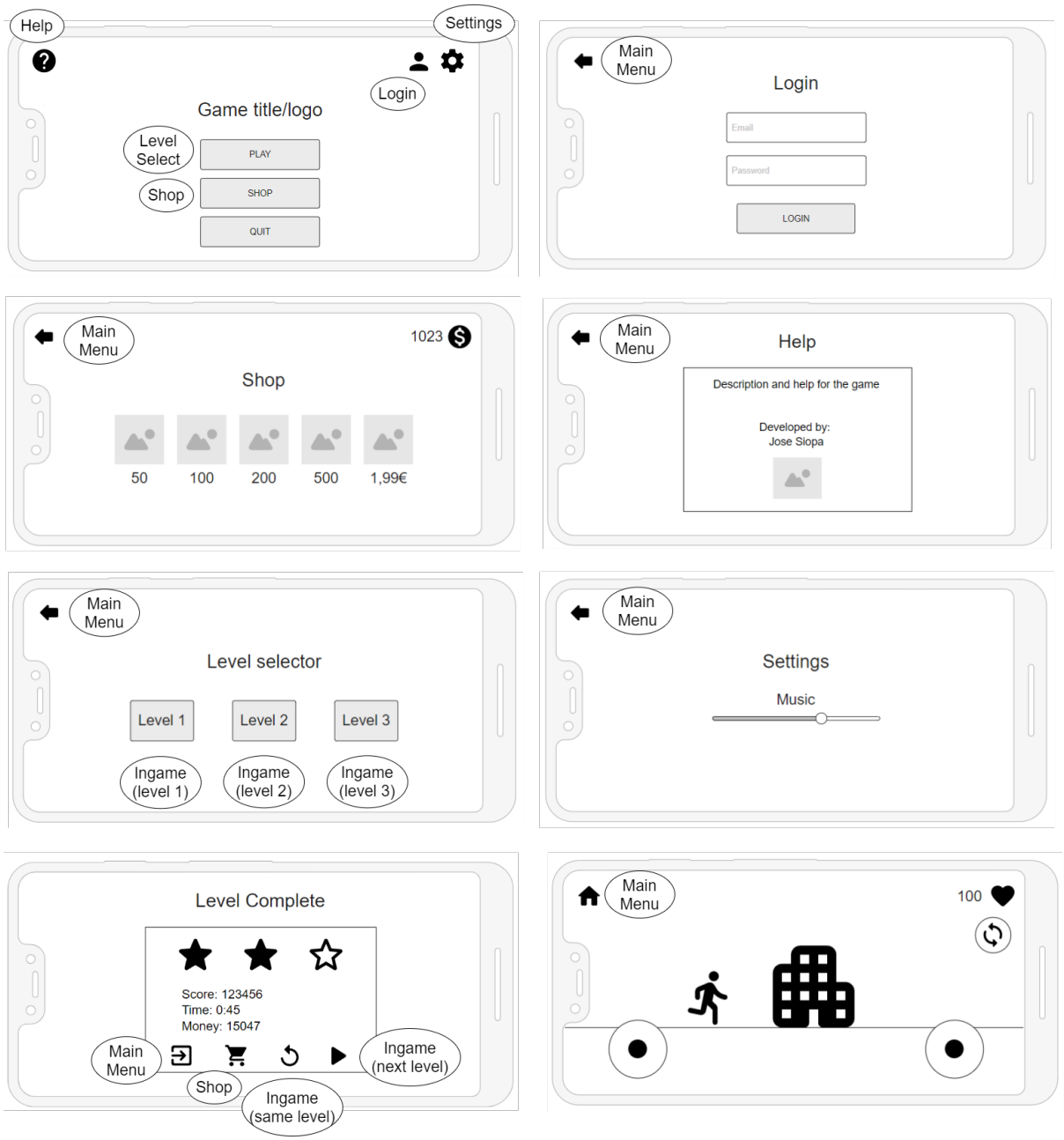



Figura 2-1: Wireframes do jogo

## 2.8 Criação da base de dados

Como já foi referido nos pontos anteriores, cada utilizador vai poder gastar moedas que coleta durante os níveis na loja do jogo. Assim, decidiu-se que estas moedas seriam guardadas na base de dados da *Firebase* para evitar quaisquer tipos de adulterações ao jogo que beneficiem o jogador sem este precisar de coletar as moedas legitimamente.

Criaram-se duas tabelas na base de dados da *Firebase*: tabela "Utilizador", que irá conter

um ID único correspondente a um *email* por pessoa e a tabela "Carteira", que contem uma referência para o ID do utilizador e o seu número de moedas. Na figura 2-2 está presente o modelo entidade-associação sem atributos.


Figura 2-2: Modelo Entidade-Associação sem atributos

# 3. Pré-produção

## 3.1 Avaliação da fase de planeamento

Para melhorar o conceito, os objetivos e os detalhes do jogo, criou-se um questionário para avaliar o protótipo e receber opiniões de vários utilizadores, de modo a construir uma aplicação que melhor os satisfaça. O questionário encontra-se no seguinte ***link***. Abaixo segue-se a lista de perguntas realizadas aos utilizadores:

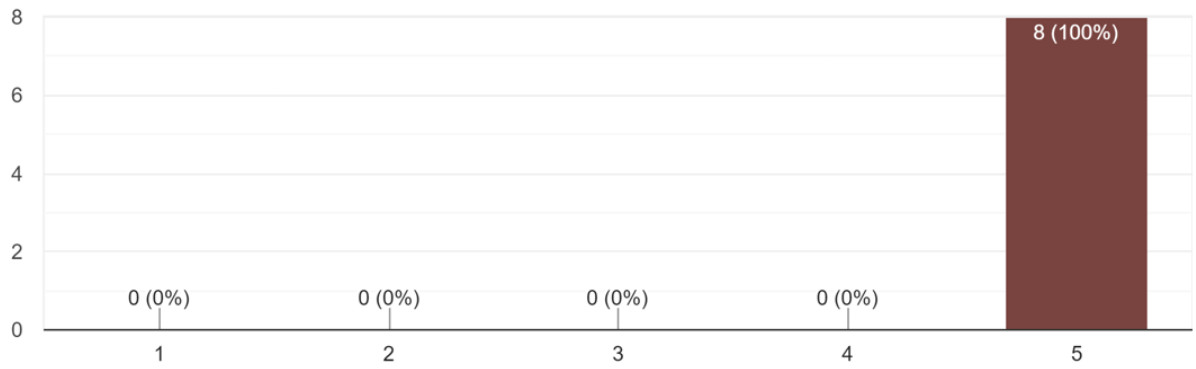
1. Qual o propósito do jogo?
2. Foi fácil perceber o conceito do jogo?
3. Gostou do jogo?
4. O que o jogador necessita fazer para ter sucesso?
5. Quando é que o jogo acaba?
6. Achou o jogo desafiante?
7. Jogaria este jogo?
8. Acredita que existe algum elemento inadequado?
9. O que gostaria de ver alterado no jogo?
10. Quanto estaria disposto a pagar para ter acesso a recursos/itens pagos?
11. Que recursos/itens gostaria que fossem adicionados e que pagaria de boa vontade?
12. Tem algum comentário adicional a fazer?

Obtiveram-se 8 respostas de diferentes utilizadores, sendo que os comentários ao jogo são bastante positivos. Segue-se a resposta dos utilizadores a cada pergunta.

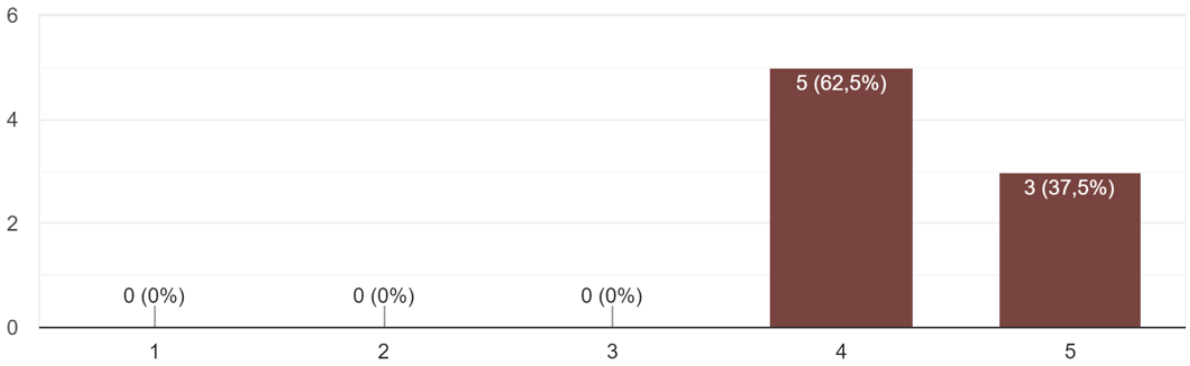
**Respostas à pergunta 1:**

- Enfrentar vários inimigos para conseguir completar o nível em que se encontra e desbloquear o próximo.
- Jogo plataforma de combate com armas corpo a corpo.
- Entreter e/ou passar o tempo.
- Bein a cute little devil and kill stuff, get stronger, platformer style.
- Matar bichos feios, avançar de nível.
- Passar vários níveis à medida que vou lutando com vários monstros.
- Matar coisas para ganhar dinheiro para comprar coisas para matar coisas melhor.
- Completar os níveis e coletar itens.

**Respostas à pergunta 2 (1 - Muito difícil, 5 - Muito fácil):**




**Respostas à pergunta 3 (1 - Detestei, 5 - Adorei):**




**Respostas à pergunta 4:**

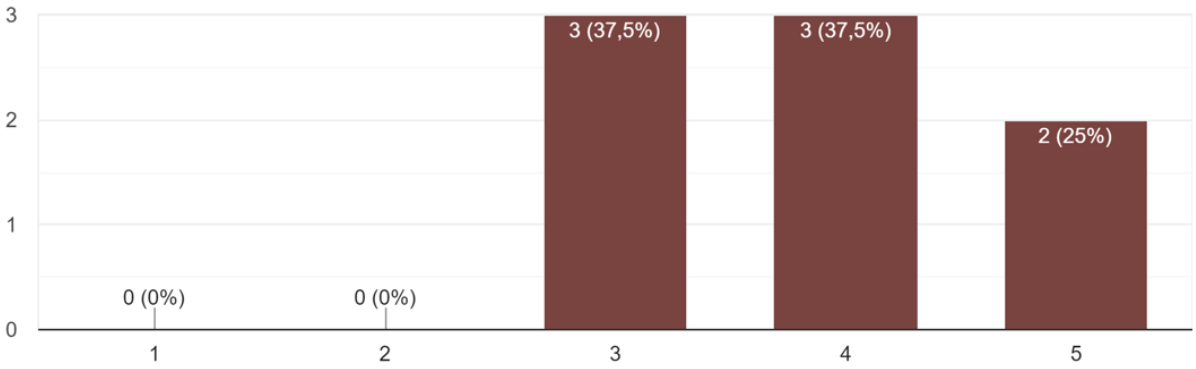
- Achar uma chave que pode estar escondida ou na posse de um inimigo, que permite abrir um baú que dará acesso ao nível seguinte.
- Derrotar inimigos para completar os níveis.
- Apanhar uma chave, que pode estar no mapa ou com um inimigo, para abrir um baú.
- Matar inimigos, comprar mais poderes, etc.
- Matar tudo.
- Perspicácia.
- Matar coisas e comprar coisas.
- Derrotar os inimigos para passar os níveis.

**Respostas à pergunta 5:**

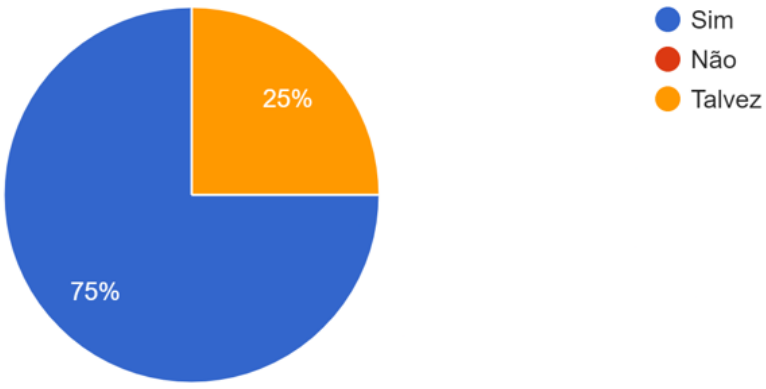
- Quando o jogador completar todos os níveis.
- O nível acaba quando o jogador abre um baú e avança para o próximo nível.

- Quando o jogador abre um baú.
- Tem vários níveis.
- Quando se matar tudo.
- Boa pergunta, provavelmente quando me fartar dele.
- Quando passares os níveis todos.
- Quando terminar todos os níveis.

**Respostas à pergunta 6 (1 - Nada desafiante, 5 - Muito desafiante):**




**Respostas à pergunta 7:**



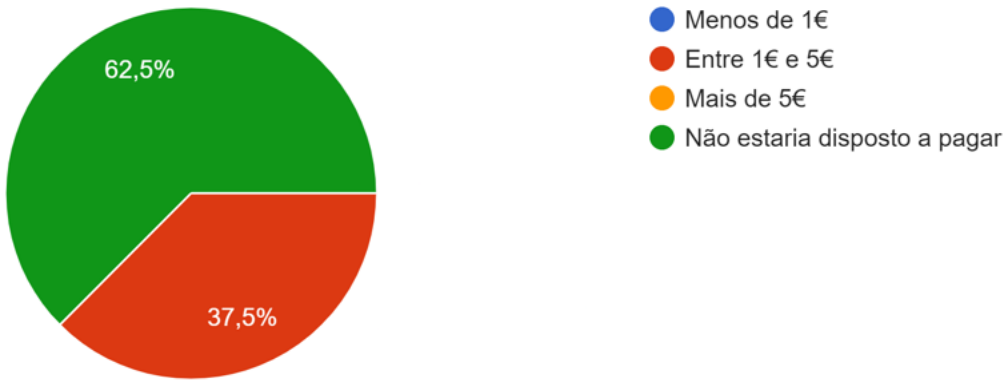

**Respostas à pergunta 8:**

- Micro transações.

**Respostas à pergunta 9:**

- Different characters.

**Respostas à pergunta 10:**




**Respostas à pergunta 11:**

- Possibilidade de jogar em modo multiplayer, a competir pelo melhor score ou tempo em cada nível.
- Skins.
- Diferentes temas em alguns níveis seria interessante.

**Respostas à pergunta 12:**

- Não obrigado.
- Porreiro pah, porreiro!

## 3.2 Melhorias sugeridas

Algumas das melhorias sugeridas pelos utilizadores seriam uma boa adição para o jogo, no entanto, a implementação destas iria ultrapassar o tempo limite definido para a produção do jogo. Assim, as únicas adições a serem feitas são a implementação de diferentes temas, um por cada nível, e alguns modelos diferentes para compra na loja.

## 3.3 Mockups do jogo

Após definidas as melhorias, criaram-se os *mockups* finais do jogo.


Figura 3-1: Mockups do jogo

## 3.4 Finalização da criação da base de dados

Após terminados os *mockups* e estando bem definido o conceito do jogo, reparou-se que é necessária a criação de outra tabela na base de dados que irá armazenar quais os itens que cada utilizador já adquiriu, evitando assim este ter que os comprar novamente sempre que abra o jogo.

Além disso, foram também adicionados os atributos ao modelo entidade-associação (Figura 3-2) e feitas algumas correções relativamente ao tipo das tabelas e das associações.

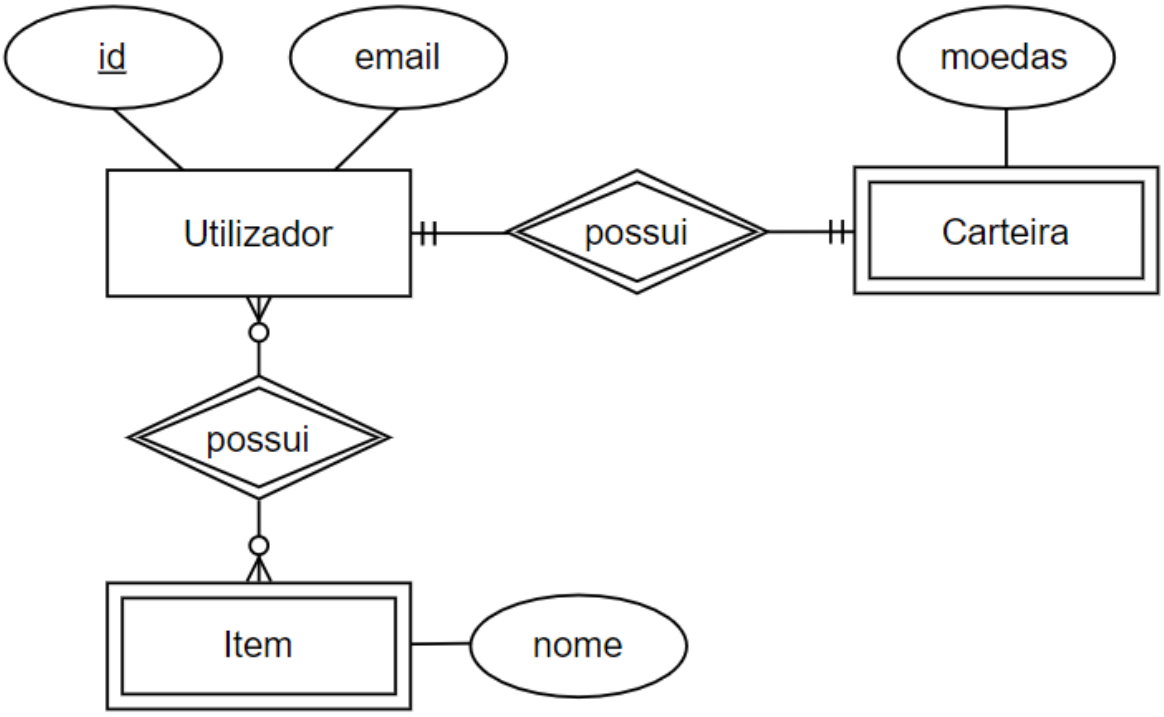



Figura 3-2: Modelo Entidade-Associação completo

# 4. Produção

## 4.1 Estruturação das classes iniciais

Para melhor organizar a estrutura da aplicação, começou-se pela criação e ligação das diferentes classes para cada ecrã. Com recurso à biblioteca *Scene2D* do *LibGDX*, facilmente se montou um esboço dos elementos da interface gráfica, utilizando tabelas para poder alinhar os diferentes botões nos sítios corretos.

Estando os elementos da interface estruturados, passou-se ao ecrã principal, o do jogo.

Assim, definiu-se duas classes abstratas principais, `Entity` e `GameMap`. A primeira classe vai englobar todos os elementos que vão interagir com o mapa ou com outras entidades. Relativamente à segunda classe, foi necessário escolher uma estratégia para facilmente criar o mapa dos níveis do jogo.

Após analisar diferentes técnicas, optou-se por utilizar o programa *Tiled* (Figura 4-1) que permite a criação de mapas utilizando "tiles", todos com tamanhos iguais. O ficheiro resultante desse mapa pode ser lido com um editor de texto, onde se verifica que este possuí uma estrutura XML, contendo referências para as imagens usadas e o mapa, que é uma grelha de "id's" para cada "tile". Com esta informação e a partir da abstrata `GameMap` é possível, com grande facilidade, a criação de qualquer nível do jogo.


Figura 4-1: Programa Tiled

## 4.2 Criação do mapa

Como se pode verificar pela figura anterior, cada "tile" está contido uma célula, sendo que diferentes "tiles" tem "id's" diferentes. Assim, criou-se a classe `enum TileType`, que permite identificar que valores correspondem a cada "tile" e para se poder saber se este tem colisão ou

se é um item coletável.

Na classe `GameMap`, criaram-se vários métodos relativos ao mapa. Sendo que se é mais fácil trabalhar com coordenadas em formato de grelha, criou-se o método `getLocationBy-GridCoordinates` que irá converter a posição da coluna e da linha dados como argumento para coordenadas globais. Isto irá facilitar a colocação de entidades manualmente.

Relativamente às colisões com o mapa, o método `doesRectCollideWithMap` irá avaliar todos os "tiles" à volta de cada entidade a verificar se estes permitem colisão.

Nesta classe serão também atualizadas e renderizadas todas as entidades criadas no mapa e, no caso da entidade jogador, verificar se este interagiu com o objetivo.

Cada nível diferente irá implementar a classe abstrata do mapa, onde no construtor serão colocadas todas as entidades nas posições pretendidas.

## 4.3 Criação das entidades

Antes da criação da classe do jogador, decidiu-se primeiramente criar a classe abstrata `Entity` que irá englobar todas as entidades possíveis existentes durante o jogo, tais como inimigos e arremessáveis. Estes são identificados através do enumerado `EntityType`, onde se pode definir o tamanho e o peso destes.

Cada entidade irá conter uma variável de posição no mundo, uma de vida, uma de verificação se a entidade está em contacto com o chão, uma de velocidade em Y, entre outras essenciais para o funcionamento correto das animações.

Assim, criaram-se os métodos que permitem renderizar as animações corretamente no ecrã, o método `moveX` que permite movimentar a entidade para a esquerda e para a direita, o método `receiveDamage` que desconta da vida atual o dano recebido e, no método `update`, sendo múltiplas vezes chamado durante o decorrer do jogo, aplica-se a força da gravidade à entidade, caso esta não esteja a colidir com o chão.

### 4.3.1 Entidade Jogador

Com esta classe está associada a classe `Controller` que contém todos os botões da interface que permitem a movimentação do personagem.

O jogador é composto por três métodos principais constantemente a serem percorridos: `handleMovement`, `handleThrowables` e `handleCollectibles`.

O primeiro, verifica que botões o jogador pressiona e aplica as respetivas ações a cada botão: seta para cima incrementa a velocidade em Y o que resulta num salto, setas para a esquerda e direita movimentam o personagem na respetiva direção, ataque verifica por alguma entidade no alcance e aplica-lhe dano se existir, arremesso instancia um objeto da classe `Throwable` na posição do jogador e movimenta-se na direção que olhava quando pressionou este botão. Para não haver sobreposição de animações, estas duas últimas ações impedem que o jogador se movimente.

O segundo método permite que os objetos atirados pelo jogador sejam movimentados em cada "frame" e irá verificar se estes se pararam de movimentar e, caso parem, são removidos do jogo.

O terceiro método irá analisar cada "tile" que estiver na posição do jogador e, caso seja do tipo coletável, irá realizar uma ação conforme o seu tipo. Logo após, é removido do mapa para evitar ser coletado novamente. O jogador pode coletar moedas para gastar na loja, comida para recuperar vida ou energia e a chave, sendo necessária para interagir com o baú e avançar de nível.

O jogador, ao realizar a ação de arremesso, é-lhe descontado um certo valor de energia, não podendo realizar novamente esta ação se esse valor estiver a zero. Ao receber um ataque de um inimigo, é descontada vida ao jogador, recomeçando o nível caso esta chegue a zero. Estes valores atualizam em tempo real as barras na interface gráfica.

### 4.3.2 Entidade Inimigo

Os inimigos funcionam com base numa máquina de estados: patrulha, ataque, ferido, morto e inativo.

Cada inimigo é criado com o tipo pretendido e com uma lista circular de pontos de patrulha. Durante o decorrer do jogo, cada inimigo irá movimentar-se na direção do próximo ponto e, quando chegar a este, irá esperar alguns segundos, enquanto define o próximo ponto de patrulha. Durante este estado, se o jogador entrar no alcance do inimigo, transita para o estado de ataque e aplica dano ao jogador até estar fora de alcance ou morto.

Um inimigo ao levar dano, transita para este estado, ficando alguns segundos parado. Caso a vida seja menor que zero, o inimigo transita para o estado inativo, que é um estado que permite este inimigo não ser mais afetado por ações externas.

Com esta classe abstrata `Enemy` é possível criar diferentes tipos de inimigos, havendo por agora só inimigos de ataque corpo a corpo.

### 4.3.3 Entidade Arremessável

Esta entidade não tira proveito dos métodos que aplicam dano, no entanto, necessita dos outros todos.

Dependendo do tipo arremessável introduzido no construtor, está atribuído diferentes valores de dano para cada um. Um arremessável irá mover-se constantemente numa direção até colidir com alguma entidade causando-lhe dano, através do método `checkForEnemies`, ou, caso quem instancie esta classe for inimigo, através do método `checkForPlayer`. Se o arremessável colidir com o mapa, também será removido.

## 4.4 Criação da loja

A loja contém uma lista de itens pré-definidos sendo eles: personagens, que apenas irá alterar o aspeto visual do jogador, mais dano e mais energia, que irão aumentar o dano de ataque e diminuir o custo de arremesso e arremessáveis diferentes, cada um causando mais dano que o outro.

O jogador pode comprar os itens que quiser com as moedas que coletou durante o decorrer dos níveis do jogo e pode equipá-los e desequipá-los quando quiser.


Figura 4-2: Ecrã da loja

## 4.5 Integração da base de dados

Como a biblioteca do *LibGDX* utiliza a pasta *core* para a criação de jogos, todas as classes aqui criadas não têm acesso às bibliotecas do *Android*. Por isso, foi necessário arranjar uma alternativa, sendo ela utilizar uma interface para poder comunicar entre as duas bibliotecas.

Assim, a classe principal do jogo `MyGdxGame` recebe como argumento um objeto que implemente a interface `Firebase`. A classe `AndroidFirebase` vai ser a responsável por implementar esses métodos, onde já há acesso à biblioteca da *Firebase*. Esta classe irá ser passada como argumento na criação do jogo.

Criaram-se métodos para se poder realizar o registo de um novo utilizador, autenticação e obter o número de moedas que cada um tem e que itens adquiriu.

No momento de registo, é adicionada à *Realtime Database* uma entrada única para cada utilizador, contendo o email, o número de moedas e os itens adquiridos.

Como os métodos que comunicam com a *Firebase* são assíncronos, o valor das moedas e os itens adquiridos de um utilizador é guardado nesta classe no momento em que a autenticação é realizada com sucesso.

É possível realizar o registo e a autenticação no ecrã de *login* (Figura 4-3), sendo que este segundo é necessário para se poder manter as moedas e itens adquiridos. Sempre que o utilizador completa um nível ou compra um item, é guardado na base de dados o novo valor de moedas e, no caso do segundo, é adicionada outra linha no respetivo utilizador com o nome do item.

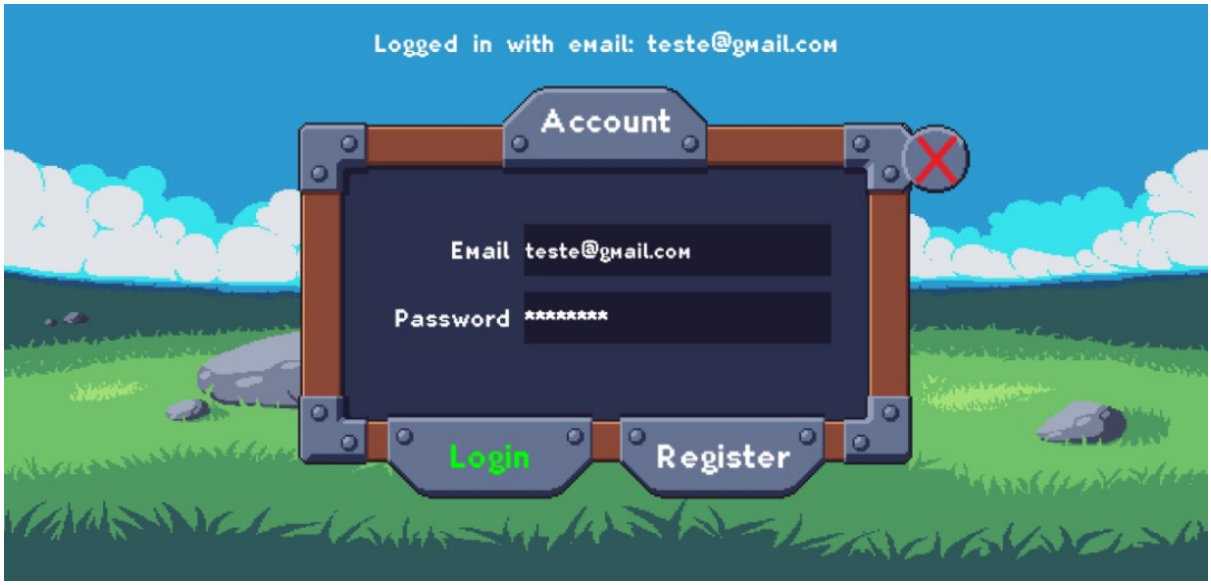



Figura 4-3: Ecrã de registo e autenticação

# 5. Pós-produção

## 5.1 Implementação de áudio

Estando concluída a produção e implementação do jogo, seguiu-se na implementação de áudio para trazer mais imersividade ao jogo. Criou-se a classe `AudioManager` onde contém todos os sons e músicas a ser utilizados durante o jogo.

Diferentes músicas são reproduzidas conforme o nível ou enquanto se navega pelos menus. A maioria das ações e interações do jogador com o mundo produzem um som.

## 5.2 Detalhes finais

Para finalizar o jogo, foi adicionado um ecrã de ajuda com o objetivo e informação sobre o jogo (Figura 5-1).

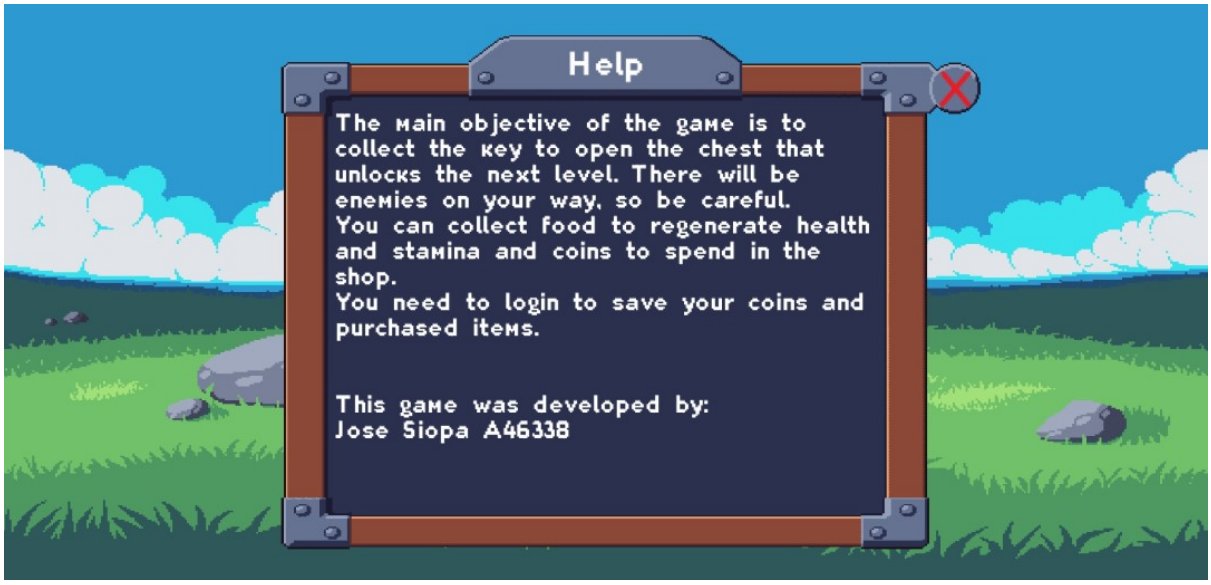



Figura 5-1: Ecrã de ajuda

Também foi criado um ecrã das definições, onde é possível alterar o volume dos efeitos sonoros ou da música (Figura 5-2).

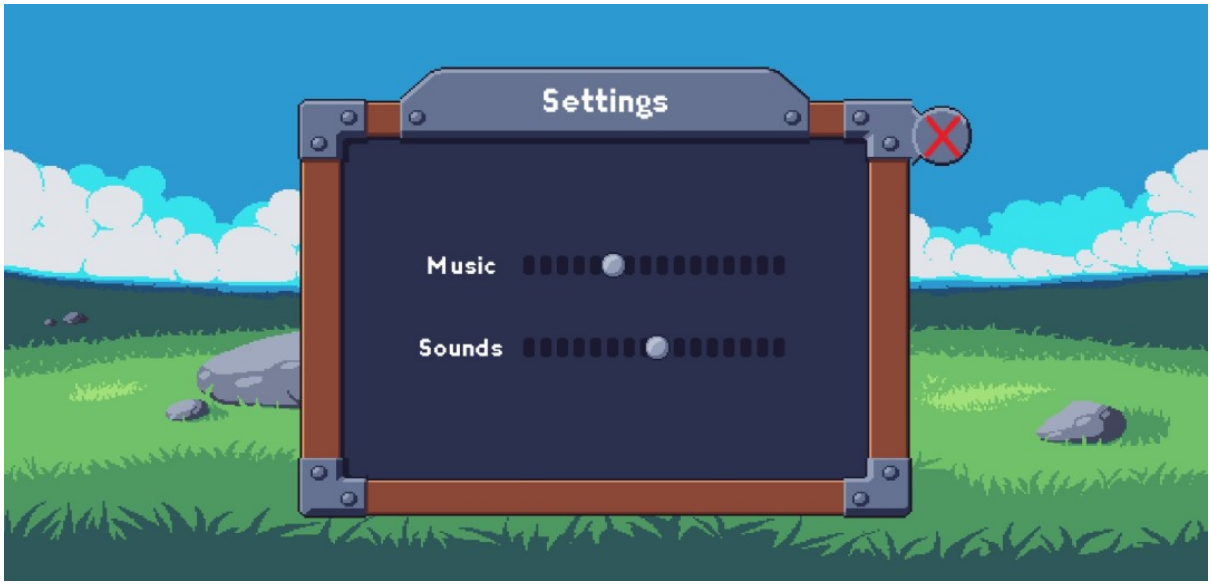



Figura 5-2: Ecrã das definições

Figura 5-3: Ecrã num nível

# 6. Conclusões

Estando a implementação do jogo feita e pronto para lançamento no mercado, concluiu-se que, apesar de a biblioteca do *LibGdx* ser complexa, alcançou-se uma aplicação robusta e divertida para os utilizadores, apesar de ter um conceito simples.

Os objetivos foram maioritariamente cumpridos com sucesso, deixando espaço para melhorias a ser feitas futuramente.

O *apk* do jogo pode ser transferido no seguinte ***link*** ou em https://bit.ly/cutelittledevil