Licenciatura em Engenharia Informática e Multimédia
2º Semestre Letivo 2019/2020

Desenvolvimento de Aplicações Móveis

Projeto Final – Jogo Android
'Stein It Up!'

Docente:
António Teófilo

Autor:
42356 Érica Pereira, 61D

Lisboa, 25 de junho de 2020

Índice

Índice de Figuras

1. Introdução

Este relatório descreve a definição de componentes básicas de um jogo Android para o projeto final, no âmbito da disciplina de Desenvolvimento de Aplicações Móveis (DAM).

Nesta fase, determinou-se o tema do jogo, as regras, o business model, alguns layouts sugestivos e o documento do conceito de jogo (Game Concept Document – GCD).

Realizaram-se e relataram-se testes à Firebase, nomeadamente o *login* de um cliente com autenticação de email/Password e guardar e obter texto e imagens.

2. Jogo *Stein It Up!*

2.1. Conceito do Jogo

Stein It Up! é um jogo que consiste de vários minijogos. Tem como objetivo o treinamento cerebral, estimulando o cérebro com desafios de matemática, lógica, memória, etc.

O jogo pode ser categorizado pelos géneros de "Logic Games", "Casual Games" e "Educational Games".

2.2. Regras do Jogo

Cada minijogo tem pontuação, número de vidas e, dependendo do tipo de jogo, tempo. Há medida que o utilizador for obtendo uma melhor pontuação, vai passando de nível, tendo desafios cada vez mais difíceis.

A pontuação final do minijogo vai depender do número de vidas e da rapidez das respostas para obter pontos bónus.

O minijogo acaba quando o tempo ou as vidas chegam ao fim, ou quando o jogador acaba todos os desafios.

Considera-se que o jogador ganhou o minijogo quando completa todos os desafios ainda com vidas restantes e, dependendo do minijogo, antes do tempo acabar.

Dependendo do minijogo, o utilizador terá de clicar, arrastar ou escrever a sua resposta.

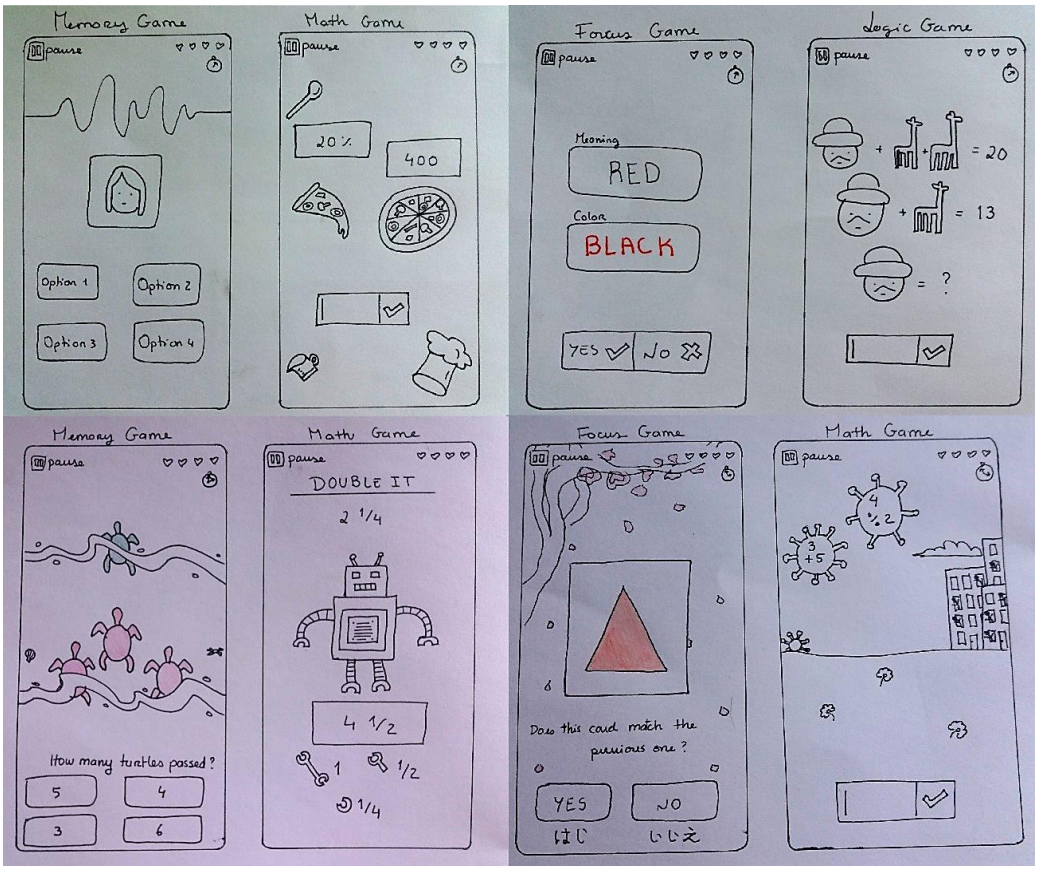


Figura 1 - Layouts sugestivos de minijogos – a) Jogo de memória; b) Jogo de matemática; c) Jogo de concentração; d) Jogo de lógica; e) Jogo de memória; f) Jogo de matemática; g) Jogo de concentração; h) Jogo de matemática.

2.3. Layout do Jogo

O ecrã inicial (figura 2) ilustra o nome do jogo, assim como o seu logotipo e um botão para o *login* do utilizador.

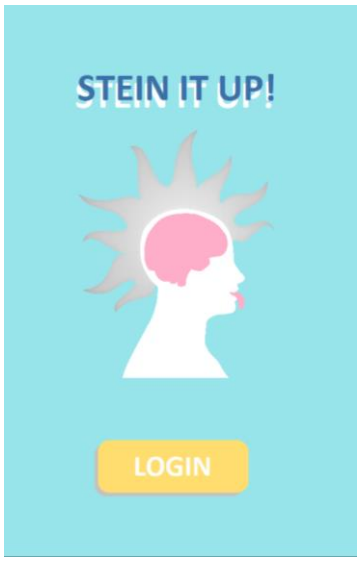


Figura 2 - Ecrã inicial do jogo.

O ecrã principal (figura 3.a) mostra três jogos aleatórios que se encontram disponíveis a cada dia e um botão para aceder a mais jogos. Este botão está só disponível para utilizadores *premium*.

O ecrã inicial (figura 3.b) tem o acesso ao menu dropdown com as opções:

- Para o utilizador se tornar *premium*, ou seja, contribuir mensalmente com uma dada quantia;
- Para o utilizador ver as suas estatísticas relativas ao seu desempenho global do jogo;
- Para editar o seu perfil;
- Para ver a informação sobre o jogo ("about");
- Para procurar ajuda com dúvidas em relação ao jogo ("help").

O ecrã inicial de um minijogo (figura 3.c) indica o nome e as regras do minijogo, assim com o botão para começar a jogar.

O ecrã final de um minijogo (figura 3.d) indica a pontuação final e um gráfico de todas as pontuações obtidas pelo utilizador. Caso o utilizador seja *premium*, o ecrã mostra também as estatísticas adicionais obtidas no minijogo.

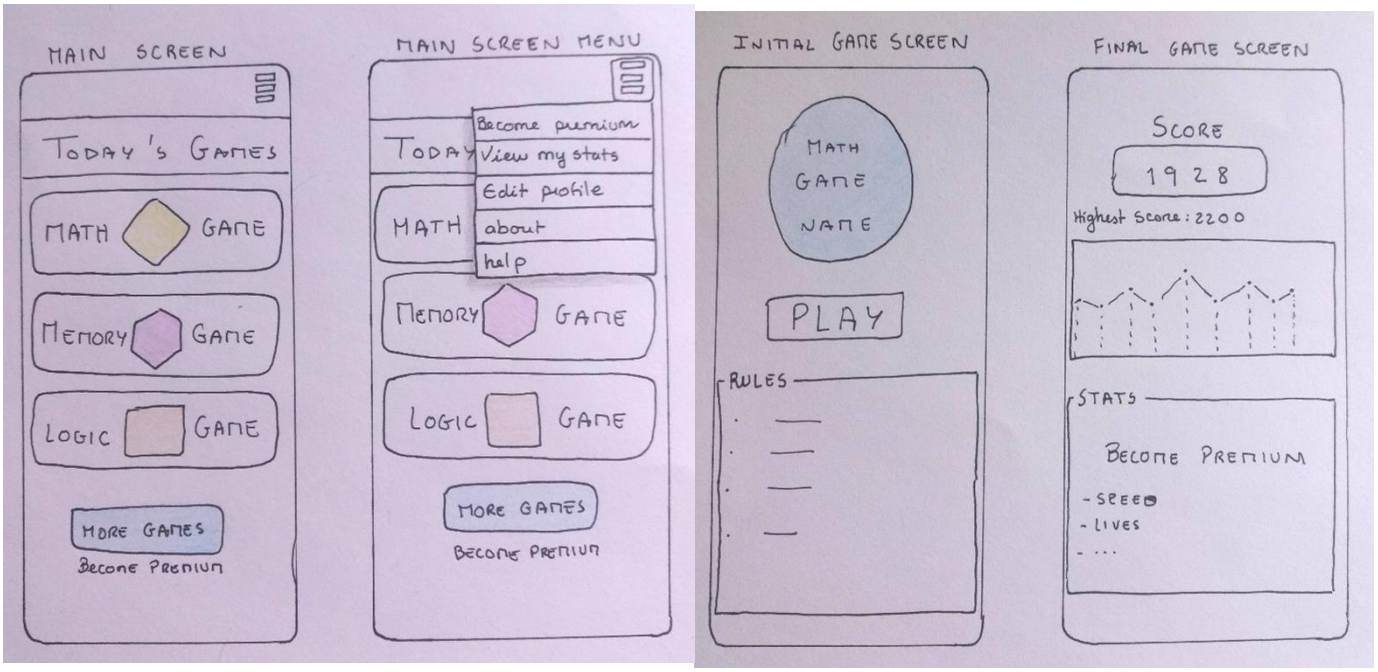


Figura 3 - a) Ecrã principal; b) Ecrã principal com dropdown menu; c) Ecrã inicial de um minijogo; d) Ecrã final de um minijogo.

2.4. Business Model

— Acesso a mais estatísticas dos jogos (speed, balanço de vidas, …) e dicas de melhoramento;
— Acesso diário a qualquer jogo ao invés de ter somente 3 jogos aleatórios disponíveis por dia.

2.5. Fun Activities

O jogo proporciona "Fun Activities" mentais e sociais:

- Mentais – Resolução de problemas e desafios para obter uma boa pontuação e passar de nível;
- Sociais – Compartilhar as estatísticas – pontuação (rapidez de respostas e balanço de vidas se o utilizador for *premium*) – com outros jogadores.

2.6. Modelo Entidade-Associação

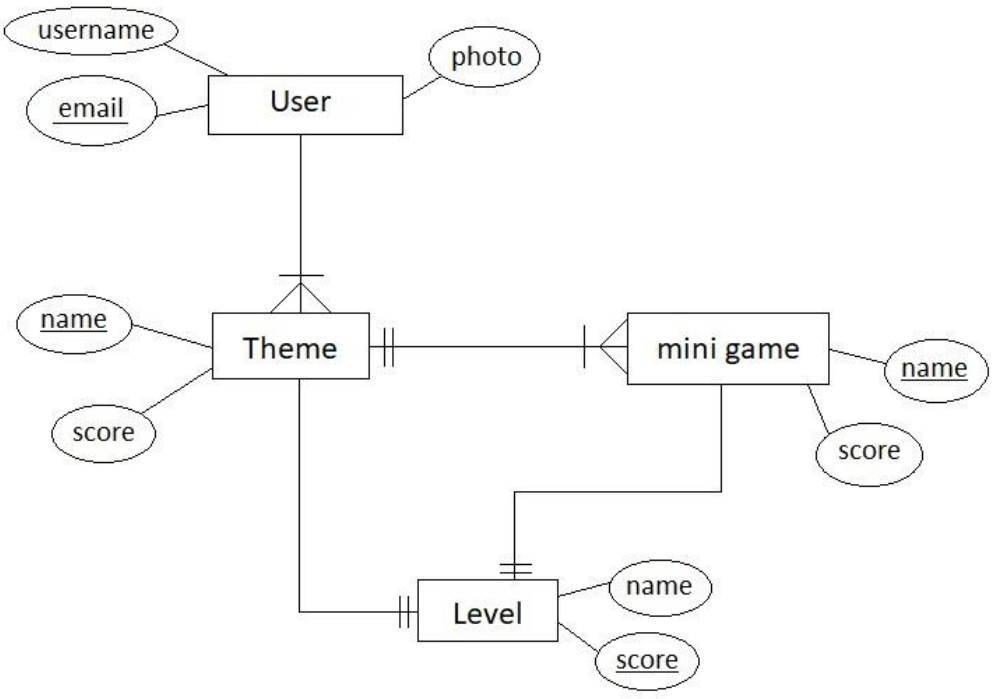


Figura 4 - Modelo Entidade-Associação.

3. Projeto *Firebase*

O projeto *Firebase* foi realizado para autenticar um utilizador, com email e password, e para salvar e obter imagens da *Firebase Cloud Storage* e texto da *Firebase Database*.

Para realizar a autenticação do utilizador com email/Password, seguiram-se as seguintes etapas:

- Criar um projeto *Firebase* – "User Authentication";
- Criar um projeto Android Cliente – "Firebase Android Client" –, no programa *Android Studio*;
- Conectar o projeto Cliente ao projeto *Firebase*;
- Desenvolver, do lado do cliente, uma *activity* (figura 6.a) com duas *EditText*'s – email e password –, assim como um botão com um Evento *OnClickListener*. Este, por sua vez, vai chamar o método *SignInWithEmailAndPassword*() do *Firebase* e autenticar o utilizador;
- Por fim, do lado do *Firebase*, adicionar manualmente um utilizador e lançar o programa.

```
mAuth.signInWithEmailAndPassword(emailText, pwdText).addOnCompleteListener(
        activity: MainActivity.this,
        new OnCompleteListener<AuthResult>() {
            @Override
            public void onComplete(@NonNull Task<AuthResult> task) {
                if (!task.isSuccessful()) {
                    Toast.makeText( context: MainActivity.this,  text: "Unsuccessful, " +
                            "please try again", Toast.LENGTH_SHORT).show();
                }
                else {
                    startActivity(new Intent( packageContext: MainActivity.this, LoadActivity.class));
                }
            }
        }
);
```

Figura 5 - Código para autenticação com email e password.

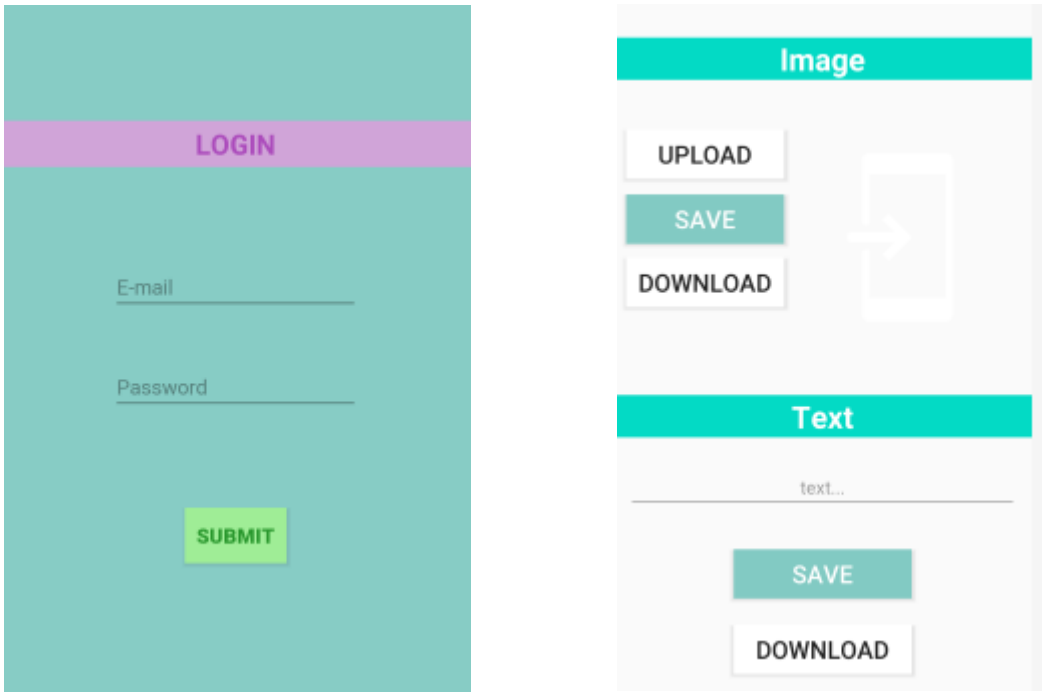


Figura 6 - a) Ecrã de login; b) Ecrã para salvar e obter imagens e texto.

Para a aplicação guardar e obter imagens, foram realizados os seguintes passos:

- Criar, no mesmo projeto cliente, uma outra *activity* (figura 6.b) com botões para salvar, obter ou fazer *upload* de uma imagem, assim como uma *EditText* para carregar texto;

- Adicionar um *OnClickListener* ao botão UPLOAD da imagem – cria um *Intent* para chamar um *startActivityForResult(),* no intuito de permitir adicionar uma imagem do dispositivo;

- Adicionar um *OnClickListener* ao botão SAVE da imagem – utiliza uma referência da *Storage* para salvar a imagem com o método *StorageReference .putFile(Uri imgUri)*;

- Adicionar manualmente uma imagem no *Firebase Storage* (para o download na aplicação cliente);

- Adicionar um *OnClickListener* ao botão DOWNLOAD da imagem - utiliza uma referência da *Storage* para ir buscar uma determinada imagem, escolhida no código, com o método *StorageReference .getFile(File file)*.

```
    public void setImage() {
        Intent intent = new Intent();
        intent.setType("image/*");
        intent.setAction(Intent.ACTION_GET_CONTENT);
        startActivityForResult(Intent.createChooser(intent, title: "Select Picture"), IMG_REQUEST_ID);
    }
@Override
protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == IMG_REQUEST_ID && resultCode == RESULT_OK && data!= null && data.getData() != null) {
        imgUri = data.getData();

        Bitmap bitmapImg = uriToBitmap(imgUri);
        image.setImageBitmap(bitmapImg);
        saveImg.setEnabled(true);
    }
public void saveImage() {
    if (imgUri != null) {
        setProgressDialog();

        StorageReference ref = storageRef.child("picture/"+ UUID.randomUUID().toString());
        try {
            ref.putFile(imgUri).addOnSuccessListener(new OnSuccessListener<UploadTask.TaskSnapshot>() {
                @Override
                public void onSuccess(UploadTask.TaskSnapshot taskSnapshot) {
                    dialog.dismiss();
                    Toast.makeText( context: LoadActivity.this, text: "Saved Successfully", Toast.LENGTH_SHORT).show();
                }
    public void getImage() {
        try {
            StorageReference storageReference = storageRef.child("picture/mew.jpg");
            final File file = File.createTempFile( prefix: "mew", suffix: "jpg");
            storageReference.getFile(file).addOnSuccessListener((OnSuccessListener) (taskSnapshot) -> {
                    Toast.makeText( context: LoadActivity.this, text: "Picture Retrieved", Toast.LENGTH_SHORT).show();
                    Bitmap bitmap = BitmapFactory.decodeFile(file.getAbsolutePath());
                    image.setImageBitmap(bitmap);
            }).addOnFailureListener((e) -> {
                    Toast.makeText( context: LoadActivity.this, text: "Error Occurred", Toast.LENGTH_SHORT).show();
            });
```

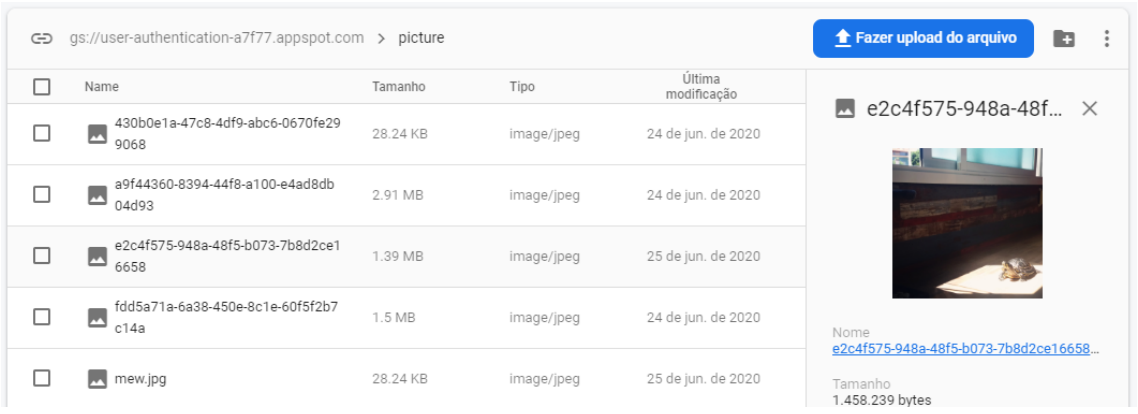


Figura 7 – Código de upload, save e download e a Firebase Storage com imagens criadas (mew.jpg) e guardadas.

Para a aplicação guardar e obter texto, foram realizados os seguintes passos:

- Adicionar manualmente uma *collection* no *Firebase Database,* para salvar e ir buscar texto de uma dada coleção;

- Adicionar um *OnClickListener* ao botão SAVE do texto – utiliza uma referência da *FirebaseStore*, que faz o acesso à *database* do *Firebase,* para salvar o text com os métodos *FirebaseFireStore .collection("text") .add(Map<String, Object> texto);*

- Adicionar manualmente um documento à *collection* do *Firebase Database,* para ir posteriormente buscar texto com o botão DOWNLOAD;

- Adicionar um *OnClickListener* ao botão DOWNLOAD da imagem - utiliza uma referência da *FirebaseStore* para ir buscar um determinado texto, escolhido no código (utilizado um ID especificado para o documento criado no *database*), com os métodos *FirebaseFireStore .collection("text") .get().*

```
public void saveText() {

    Map<String, Object> send = new HashMap<>();
    send.put("helloAppText", text.getText().toString());

    firestore.collection( collectionPath: "text").add(send).addOnSuccessListener((OnSuccessListener) (documentReference) → {
            Toast.makeText( context: LoadActivity.this,  text: "Text Inserted successfully", Toast.LENGTH_SHORT).show();
    }).addOnFailureListener((e) → {
            Toast.makeText( context: LoadActivity.this,  text: "Error Occurred", Toast.LENGTH_SHORT).show();
    });
```

```
public void getText() {
    firestore.collection( collectionPath: "text").get().addOnCompleteListener((task) → {
            if (task.isSuccessful()) {
                boolean found = false;

                for (QueryDocumentSnapshot document : Objects.requireNonNull(task.getResult())) {

                    if (document.getId().equals("text1")) {
                        String txt = document.getData().get("helloGamerText").toString();
                        text.setText(txt);
                        Toast.makeText( context: LoadActivity.this,  text: "Retrieved Successfully", Toast.LENGTH_SHORT).show();
                        found = true;
                    }
                }
                if (!found) {
                    Toast.makeText( context: LoadActivity.this,  text: "Text not found", Toast.LENGTH_SHORT).show();
```

Figura 8 - Código de save e download de texto.

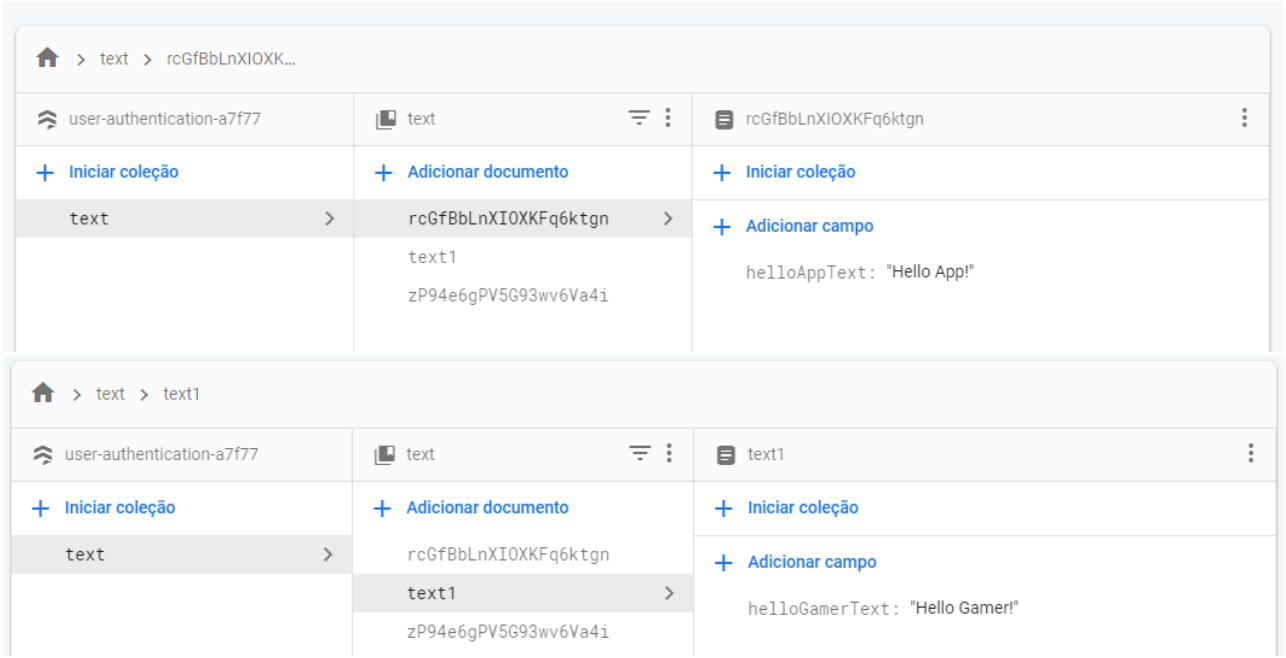


Figura 9 - Firebase Database com texto criado (text1) e texto guardado.

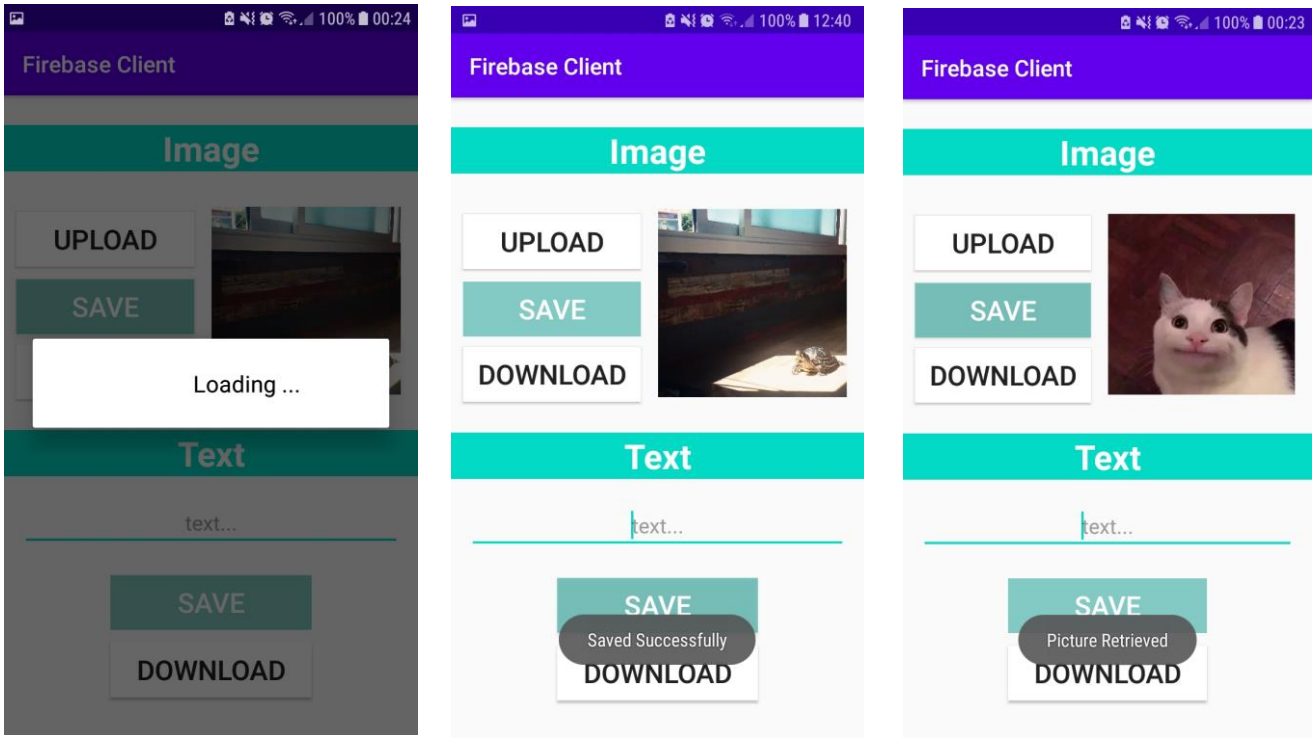


Figura 10- a) A guardar uma imagem selecionada; b) Imagem selecionada guardada com sucesso; c) Imagem resgatada da Firebase Storage.

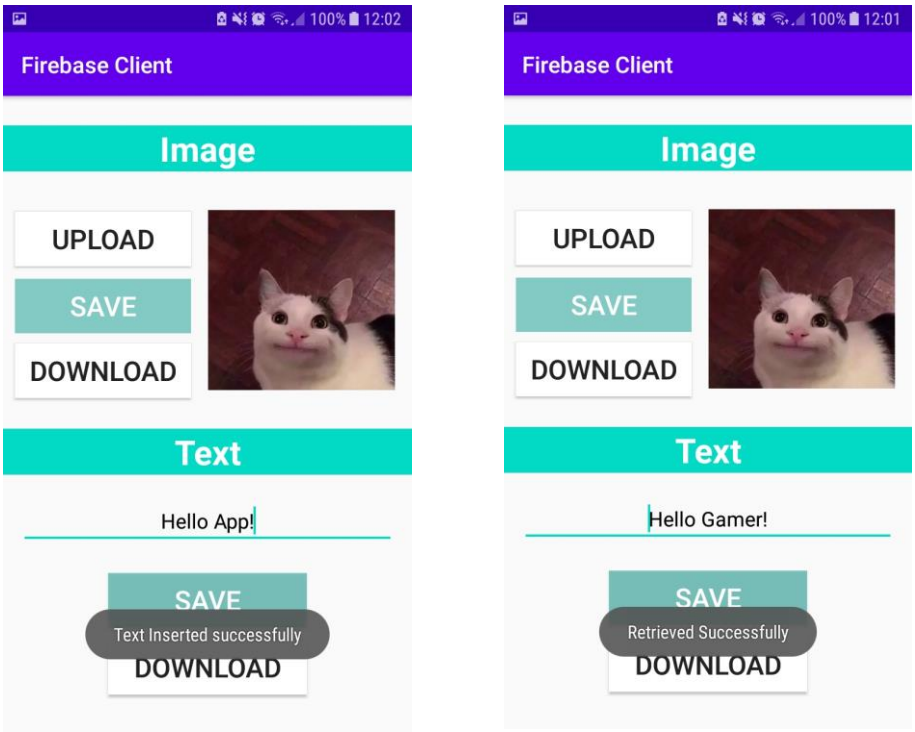
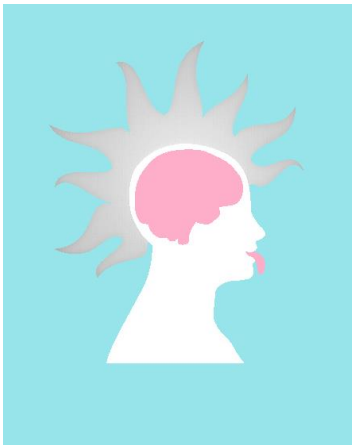


Figura 11 - a) Texto guardado com sucesso na Firebase Database; b) Texto resgatado da Firebase Database.

4. Game Concept Document (GCD)

STEIN IT UP!



Stein It Up! é um jogo que envolve "usar a cabeça", treinando o cérebro com jogos de lógica, matemática, memória e concentração, de uma maneira mais divertida e dinâmica.

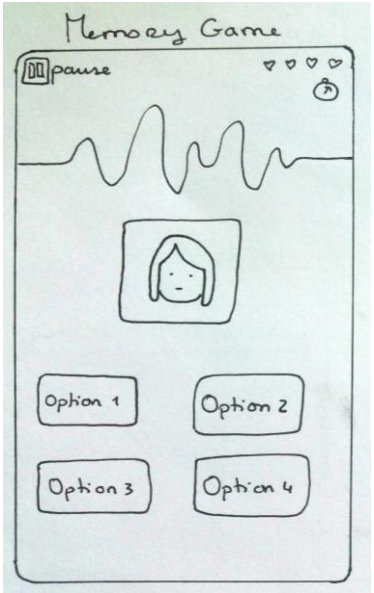
Características do Jogo ----------

❖ *Features*

Stein It Up! é um jogo **educativo**, de **lógica** e de **self-improvement**, com vários minijogos disponíveis, jogado por **um único jogador**.

❖ *StoryBoard*

O jogador começa no nível iniciante. A progressão do jogo reside na acumulação das pontuações de cada minijogo, então, para subir de nível, o jogador precisa de acumular um determinado número de pontos.



Nível	Pontuação
Iniciante	0
Intermédio	10000
Avançado	30000
Especialista	70000
Master	150000

Figura 12 - Exemplo de um layout de um jogo e uma tabela com os níveis e pontuação respetiva.

O jogador, ao conseguir obter uma elevada acumulação de pontos (jogando o minijogo várias vezes), consegue subir de nível, o que faz com que aumente cada mais a dificuldade dos desafios do minijogo.

O jogador é classificado pelos níveis em cada minijogo e em cada género do jogo (figura 11).

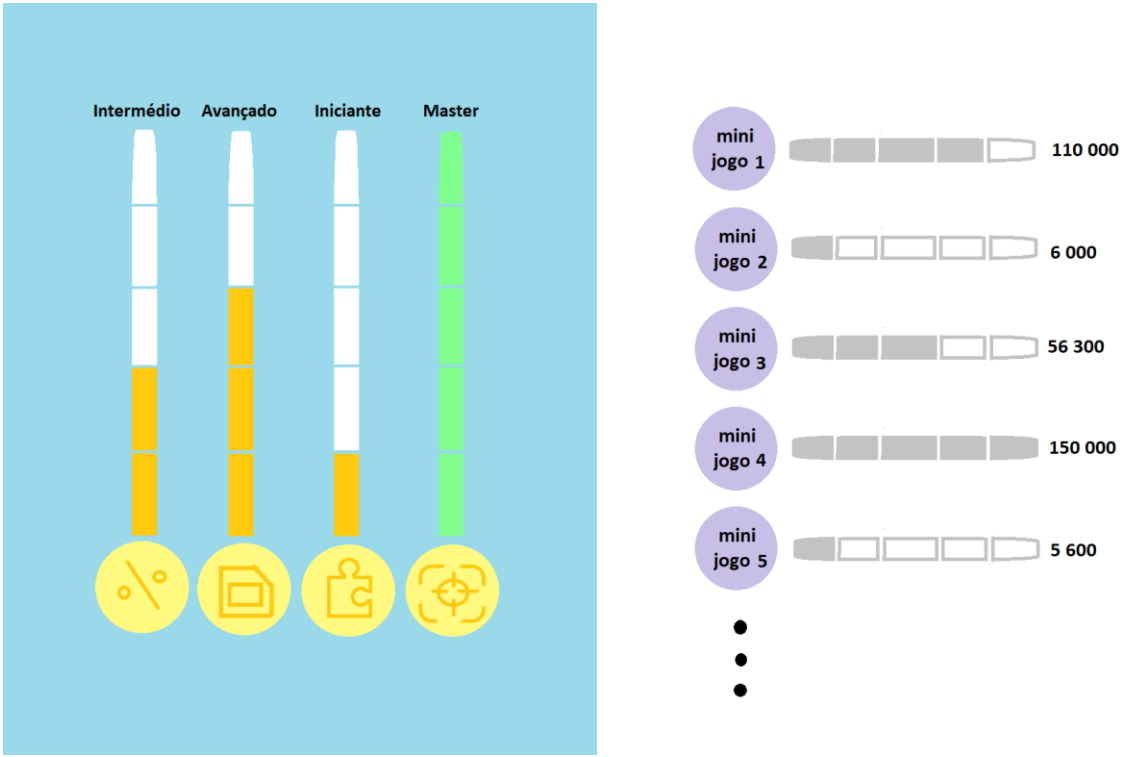


Figura 13 - Exemplificação da classificação do jogador nos tipos de minijogos e em cada minijogo.

❖ *Genres*

O jogo pode ser categorizado pelos géneros de "Logic Games", "Casual Games" e "Educational Games":

— É um jogo do género "Logic Game" por requerer ao jogador a resolução de puzzles de lógica;

— É um jogo do género "Casual Game" por ser um jogo que permite diárias rajadas de jogo potencialmente curtas e ser um passatempo curto e relaxante, um descanso entre outras ocupações;

— É um jogo do género "Educational Game" por poder servir de veículo de aprendizagem, uma vez que serve de estímulo cerebral, com desafios matemáticos, lógicos, de memória e de concentração.

❖ *Audience*

Este jogo tem como alvo principal o público adulto. Este jogo destina-se também a qualquer utilizador com interesse e gosto por desafios mentais.

Por ser um jogo mais casual, ou seja, 10 a 20 minutos de treino cerebral, possivelmente 1 ou 2 vezes ao dia, é provável ser mais popular entre os dados demográficos que têm menos tempo livre.

❖ *Player Experience*

O jogo permite ao utilizador aceder a um determinado número de minijogos na sua página inicial.

O utilizador tem acesso a quatro tipos de minijogos: matemática, lógica, memória e de concentração.

O jogo permite ao utilizador ver as suas estatísticas relativas a cada tipo de minijogo, tais como pontuação, rapidez de resposta, número de vidas e o nível em que se encontra (iniciante, mediano, avançado, especialista e master).