Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e Multimédia (LEIM)

Modelação e Programação (MP) – 20/21

Projecto Final – Parte B

Trabalho realizado por:	número
Miguel Ginga	46292

Lisboa, 11 de Junho de 2021

Índice de matérias

1. Introdução 5
2. Apresentação da Interface 6
3.Diagrama UML de Classes 9
4.Classe UI 12
5. Selection Panel 14
6. PokemonPoolPanel e PokemonLabel 17
7. TeamPanel 20
8. EventPanel 21
9. BattlePanel 23
11. ImagePanel 26
11. Conclusões 27

Índice de Imagens

Figura 1 – Mockup do Painel de selecção 6
Figura 2 – Mockup do Painel de Batalha 7
Figura 3 – Mockup do Painel de Eventos 7
Figura 4 – Mockup do Painel de Opções 7
Figura 5 – Diagram UML das classes associadas a JSwing 9
Figura 6 – Diagrama UML da Classe UI 12
Figura 7 – Exemplo de CardLayout 12
Figura 8 – Corpo do método PlaySound 13
Figura 9 – Diagrama UML da Classe SelectionPanel 14
Figura 10 – Exemplo de BorderLayout 14
Figura 11 – Exemplo da utilização do layout GridBagLayout 15
Figura 12 – Exemplo de adição de componentes ao Borderlayout 16
Figura 13 – Painel de Selecção 16
Figura 14 – Diagrama da Classe PokemonLabel 17
Figura 15 – Exemplo de PokémonLabel 17
Figura 16 – Exemplo de PokémonLabel seleccionada 18
Figura 17 – Diagrama da classe PokemonPoolPanel 18
Figura 18– Diagrama da Classe TeamPanel 20
Figura 19– Exemplo de TeamPanel e TeamPanel com uma label disabled 20
Figura 20– Diagrama da classe EventPanel 21
Figura 21– Exemplo da aplicação da linguagem HTML e representação de battleText 21
Figura 21– Exemplo do Painel com as imagens dos Pokémon activos 22
Figura 22– Comandos para alterar o visual da JProgressBar 22
Figura 23– Diagrama da classe BattlePanel 23
Figura 24– Painel optionsPanel 24
Figura 25– Painel movesPanel 24
Figura 26– Painel changePokemon 25
Figura 27– Painel endPanel 25
Figura 26– Diagrama da classe ImagePanel 26
Figura 27– Exemplo da criação de um ImagePanel 26

1. Introdução

No âmbito da unidade curricular Modelação e Programação, foi nos proposto como trabalho final o desenvolvimento de uma aplicação com interface gráfica. Neste trabalho devemos aplicar e consolidar os conhecimentos desenvolvidos ao longo de todo o semestre. O projecto está dividido em duas partes:

- Parte A - Desenvolvimento da aplicação em Java em modo de texto/consola.
- Parte B – Desenvolvimento da interface gráfica para aplicação desenvolvida na parte A.

Este relatório é exclusivo à Parte B do projecto, pelo que irá abordar todo o desenvolvimento da interface gráfica para a aplicação desenvolvida na Parte A, desde os diagramas UML ao desenvolvimento das classes. Irá ser descrita a razão que nos levou a utilizar este modelo de interface, o que foi necessário para alcançar a interface final e as funcionalidades presentes na mesma.

2. Apresentação da Interface

Tal como descrito na Parte A do relatório este projecto destina-se a um simulador de batalhas Pokémon. O primeiro passo para dar inicio à batalha será apresentar uma lista ao utilizador de Pokémon por onde possa escolher a sua equipa.

Figura 1 – Mockup do Painel de selecção

Teremos um painel de selecção com uma listagem de todos os Pokémon disponíveis onde cada uma das entradas terá:

- Uma imagem (*Sprite*) que representa o Pokémon numa vista minimizada
- O nome do Pokémon e os seus tipos (*PokemonType*)
- Uma cor de fundo associada ao tipo principal

Só é possível ao utilizador avançar para o próximo menu após seleccionar exactamente 3 Pokémon. A equipa do computador será escolhida aleatoriamente após a equipa do utilizador estar definida.

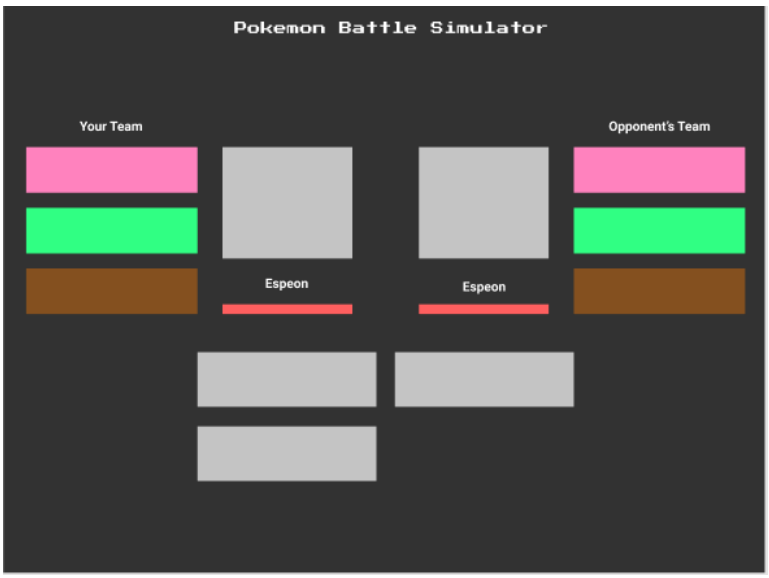


Figura 2 – Mockup do Painel de Batalha

O próximo painel, será o painel de Batalha, que vai ter todas as opções necessárias para executar o fluxo de combate. Teremos neste painel 2 painéis responsáveis por mostrar ao utilizador a sua equipa e do seu adversário que irão ser alterados para demonstrar a situação actual de cada equipa, ou seja, que Pokémon's já não podem lutar e quais podem.

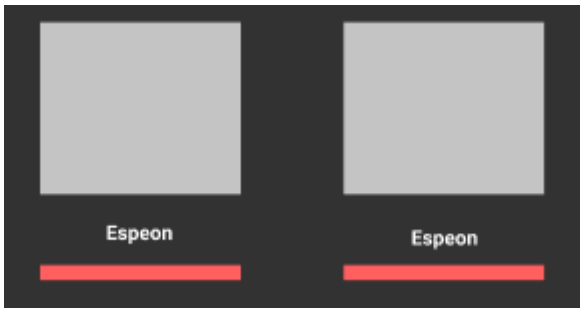


Figura 3 – Mockup do Painel de Eventos

O painel de eventos irá conter duas imagens mais detalhadas de cada Pokémon activo, tal como os seus nomes e uma barra com os pontos de vida. Caso o Pokémon seja afectado por algum tipo de status (*StatusType*), isso será reflectido juntamente com o seu nome.

Figura 4 – Mockup do Painel de Opções

O painel de opções contém todos os botões que serão necessários para combater. Teremos um botão de batalha onde iremos escolher o ataque a utilizar, um botão que nos disponibiliza os nossos Pokémon disponíveis para trocar para Pokémon activo e ainda um botão

para começar um novo jogo que nos irá levar de volta ao painel de selecção. Embora este Painel seja de certa forma dinâmica por ser sempre alterado que um botão seja pressionado, o conteúdo será semelhante sempre no formato de botões com acções associadas.

3.Diagrama UML de Classes

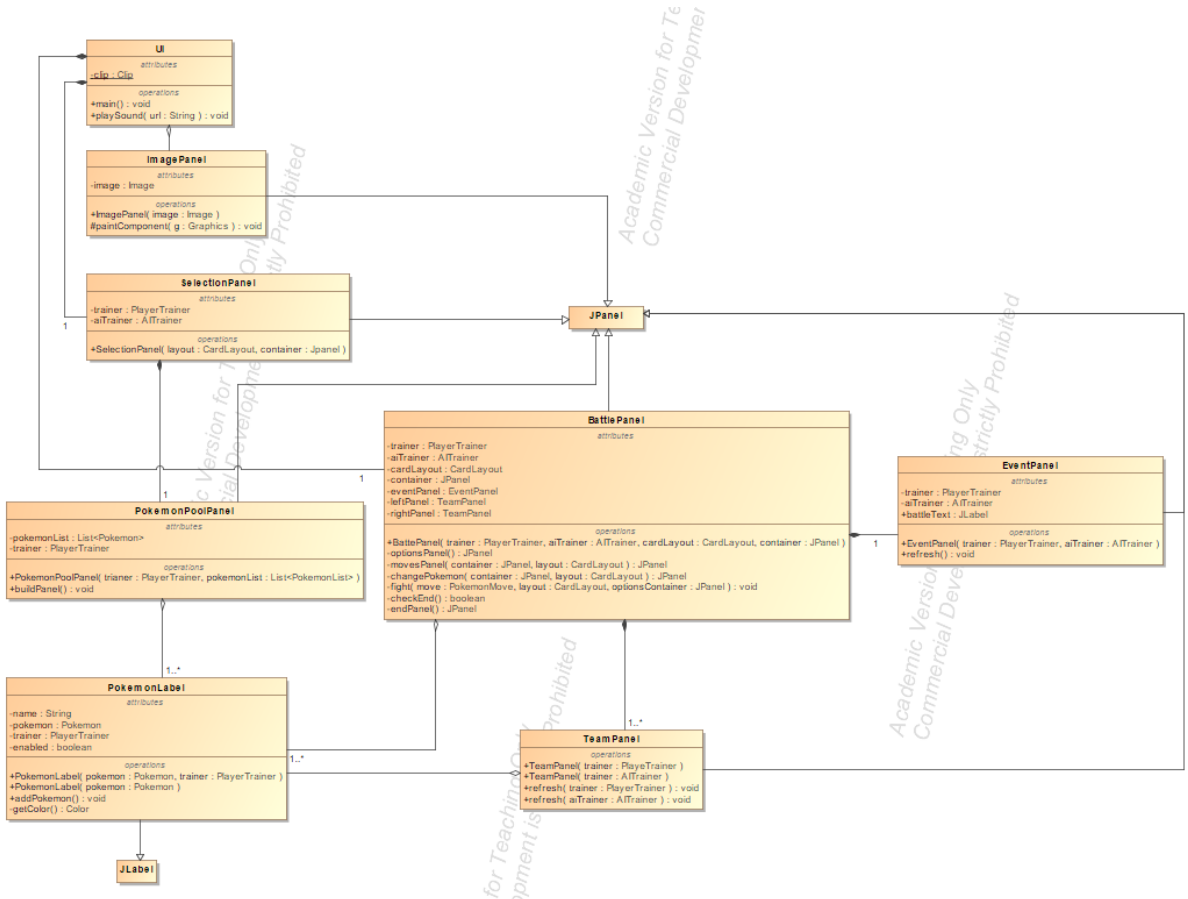


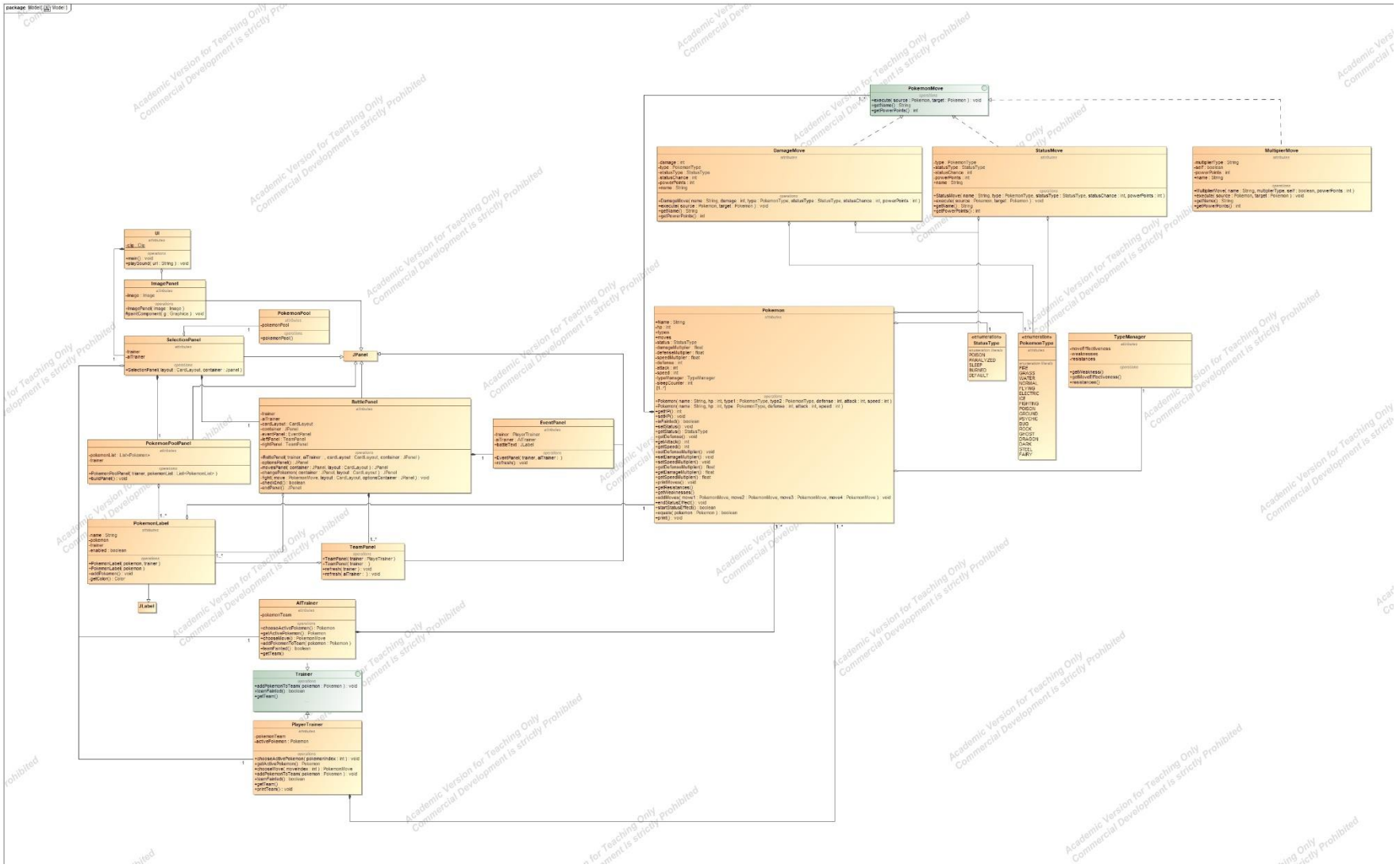
Figura 5 – Diagram UML das classes associadas a JSwing

Na figura 5 podemos observar o diagrama das novas classes desenvolvidas para a realização da Interface Gráfica da aplicação. Estas classes são as classes associadas exclusivamente a elementos da biblioteca JSwing.

Podemos observar temos várias classes que derivam de JLabel, sendo elas BattlePanel, SelectionPanel, PokemonPoolPanel, EventPanel e TeamPanel.

Na classe UI temos directamente associados o SelectionPanel e o BattlePanel. O SelectionPanel está por sua vez associado ao PokemonPoolPanel e este tem uma associação fraca com PokemonLabel.

A classe BattlePanel tem directamente associado as classes TeamPanel, por sua vez com associação fraca a PokemonLabel , e a classe EventPanel.


PokemonMove
DamageMove
StatusMove
MultiperMove
Pokemon
TypeManager
UI
ImagePanel
PokemonPool
SelectionPanel
JPanel
BattlePanel
EventPanel
PokemonPoolPanel
PokemonLabel
TeamPanel
JLabel
AiTrainer
Trainer
PlayerTrainer

4.Classe UI

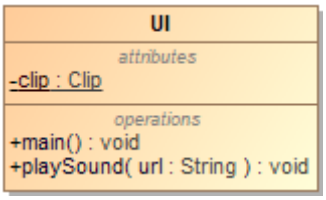


Figura 6 – Diagrama UML da Classe UI

A classe UI é a classe onde tudo será iniciado. Nesta classe iremos declarar o nosso *JFrame* que irá ser o contentor para todos os painéis desenvolvidos. Como pretendemos trocar painéis ao longo do decorrer da aplicação necessitamos de adicionar um painel com um layout que permita estas operações. Aqui fazemos o nosso primeiro uso do layout *CardLayout*, que nos permite adicionar componentes ao nosso painel contentor associados a uma *String* que será usada como chave. Recorrendo ao método *show* de *CardLayout*, definimos qual o contentor que estamos a usar e qual o painel dentro deste que queremos mostrar através da chave introduzida.

```
container.setLayout(cardLayout);
container.add(selectionPanel, "1");
cardLayout.show(container, "1");
```

Figura 7 – Exemplo de CardLayout

Aqui definimos o layout de container como um *CardLayout*, adicionamos o painel *selectionPanel* com a chave "1" e definimos este painel como o que será apresentado através de *show*.

Para além de definirmos as dimensões da nossa *JFrame*, iremos ainda declarar dados adicionais, como a criação de dois painéis- SelectionPanel e *ImagePanel* (abordados mais à frente no relatório) – e a declaração de métodos auxiliares que nos permitem visualizar o *frame* e terminar a aplicação quando fechado o mesmo.

Um dos objectivos da aplicação era a introdução de música no mesmo, este efeito foi obtido através de um método obtido após alguma pesquisa. Este método é o playSound e faz uso das bibliotecas nativas do java sem requerer o uso de uma biblioteca adicional. Nesta biblioteca utilizamos as seguintes classes:

- AudioSystem – uma classe que serve como ponto de acesso para os misturadores de som do nosso dispositivo tal como acesso a ficheiros de áudio.

- AudionInputStream – uma classe que contém streams de áudio com determinado formato e dimensão.
- Clip – uma interface que permite o controlo dos streams de áudio.

```
clip  = AudioSystem.getClip();
AudioInputStream inputStream = AudioSystem.getAudioInputStream(new File(url).getAbsoluteFile());
clip.open(inputStream);
clip.start();
```

Figura 8 – Corpo do método PlaySound

Começamos por obter um clip que suporte ficheiros de aúdio através dos misturadores do nosso dispositivo. Após isto vamos obter o nosso *stream* de áudio a ser tocado por este *clip*. Com o ficheiro de áudio obtido podemos iniciar a sua reprodução com o método *start* de *Clip*.

5. Selection Panel

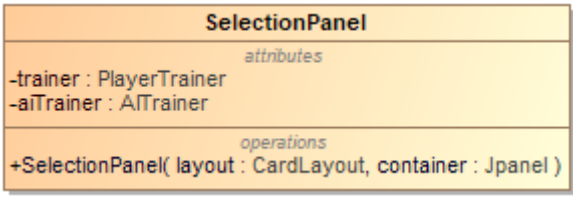


Figura 9 – Diagrama UML da Classe SelectionPanel

A classe *SelectionPanel* deriva da classe *JPanel*, isto será uma tendência que iremos verificar nas classes desenvolvidas ao longo deste projecto. Isto permite-nos duas coisas essencialmente, a manipulação do próprio componente ao longo dos seus métodos e a construção de novos objectos com as mesmas propriedades deste sem que seja necessário declarar mais informações.

Nesta classe recebemos dois argumentos que são provenientes da classe UI, o *CardLayout* do contentor principal e o próprio contentor. Isto vai permitir que adicionemos e alteremos entre painéis no contentor. Declaramos e iniciamos aqui também os nossos treinadores, *PlayerTrainer* e *AITrainer*.

Este painel vai ter como layout o BorderLayout que nos permite adicionar elementos por secções ao mesmo.

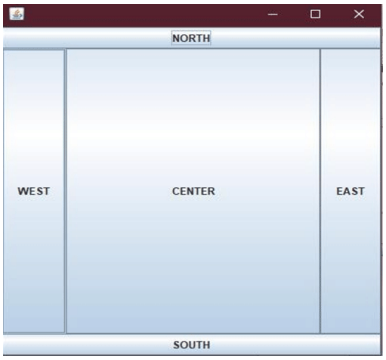


Figura 10 – Exemplo de BorderLayout

Com este layout conseguimos seccionar o conteúdo e dar um papel central ao que considerarmos mais importante. Neste caso na secção centra iremos colocar o painel PokemonPoolPanel que irá conter a nossa listagem de Pokémon. Os paineis laterais serão apenas declarados para que o conteúdo central fique nas dimensões que considerarmos apropriadas. O pain superior está reservado a um painel da classe ImagePanel que vai conter o logo da nossa aplicação. E por último no painel inferior teremos as regras para dar inicio ao combate e o botão que permite esta mesma acção.

Dentro do construtor inicializamos o nosso *trainer* e *aiTrainer*, tal como a nossa *PokemonPool*. Isto permite que sempre que inicializarmos este painel teremos novos treinadores e lista de Pokémon.

No painel inferior utilizamos um novo layout, *GridBagLayout*. Este layout permite-nos adicionar o conteúdo como se tivesse numa grelha em que indicamos os índices que queremos que cada componente ocupe, podendo ainda definir quantas linhas ou colunas que este ocupe ou o espaço entre elementos. Estas opções são obtidas através do objecto *GridBagContraints*. Uma das grandes vantagens deste layout e a razão principal pelo qual o utilizamos é que permite-nos centrar o conteúdo com grande facilidade.

```
JPanel bottomPanel = new JPanel( new GridBagLayout());
bottomPanel.setPreferredSize(new Dimension(800, 100));
GridBagConstraints gbc = new GridBagConstraints();
gbc.fill = GridBagConstraints.HORIZONTAL;
JLabel rule1 = new JLabel("Please choose 3 Pokémon to begin the battle");
rule1.setHorizontalAlignment(SwingConstants.CENTER);
gbc.gridx = 0;
gbc.gridy = 0;
bottomPanel.add(rule1, gbc);
```

Figura 11 – Exemplo da utilização do layout GridBagLayout

Nesta classe temos o nosso primeiro encontro com o componente *JButton* e a interface *ActionListener*. De forma a adicionarmos uma função ao nosso botão quando este é pressionado necessitamos de adicionar um *ActionListener* em que damos *override* ao seu método *actionPerformed*. Neste botão os eventos que associamos são os seguintes:

1. Iniciar uma nova *PokemonPool* para o *aiTrainer*.
2. Verificar se o utilizador já escolheu 3 Pokémon quando o botão é pressionado.
3. Se sim, adicionar aleatoriamente 3 Pokémon à equipa do nosso *aiTrainer*.
4. Definir os Pokémon activos para cada treinador.
5. Inicializar um *BattlePanel* (painel de Batalha) e adicionar este ao nosso contentor principal e definir como painel activo no nosso *CardLayout*.
6. Interromper o áudio do painel de selecção e iniciar o aúdio do painel de batalha.

Com todos os componentes necessários construídos só temos de adicionarmos os paineis nas secções respectivas do BorderLayout e verificar o resultado final.

```
add(poolpanel, BorderLayout.CENTER);
add(bottomPanel, BorderLayout.SOUTH);
add(leftPanel, BorderLayout.WEST);
add(rightPanel, BorderLayout.EAST);
add(northPanel, BorderLayout.NORTH);
```

Figura 12 – Exemplo de adição de componentes ao Borderlayout

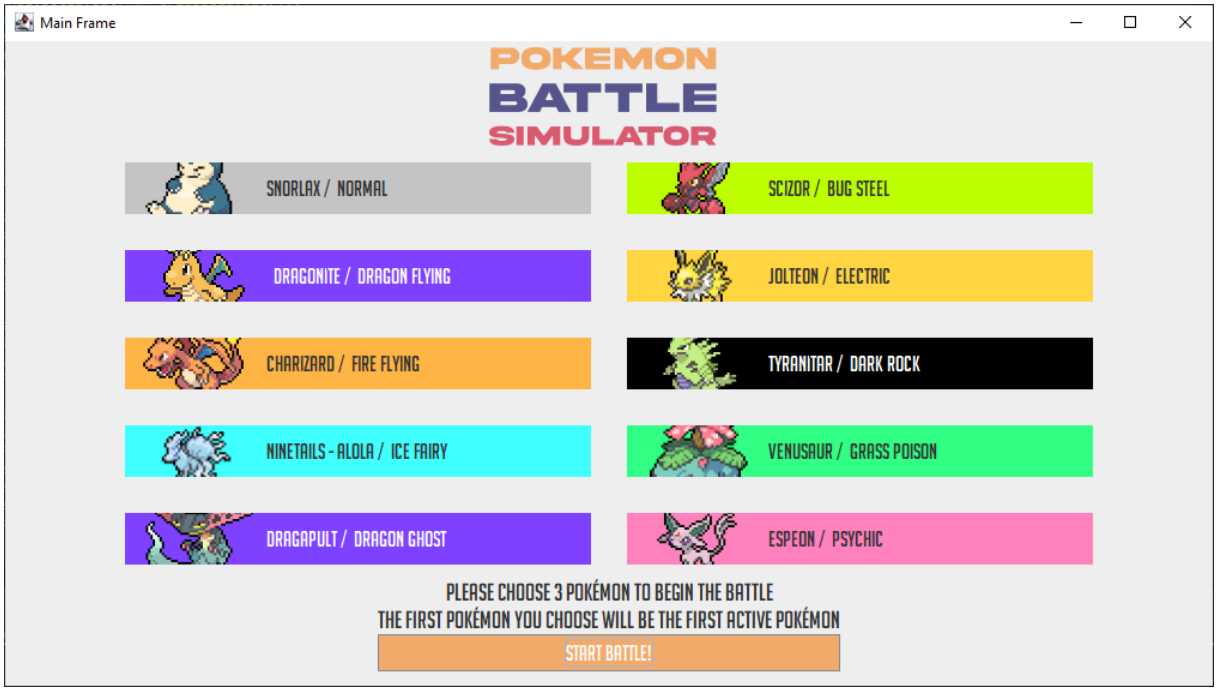


Figura 13 – Painel de Selecção

6. PokemonPoolPanel e PokemonLabel

Figura 14 – Diagrama da Classe PokemonLabel

A classe *PokemonLabel* deriva do componente *JLabel*. Nesta classe iremos definir a *JLabel* que vai conter a informação reduzida do nosso Pokémon: uma imagem minimizada, nome e tipos. De forma a conseguirmos adicionar imagem e texto numa *label*, utilizamos o método *setIcon* que nos permite adicionar uma imagem como *icon*.

Nesta classe temos um construtor que recebe o treinador do utilizador e um Pokémon que juntamente com o método *addPokemon* nos vai permitir adicionar o Pokémon associado à *label* directamente à lista do nosso treinador quando clicarmos nesta.

O segundo construtor permite a criação de uma *label* que será construída só para efeito de apresentação sem nenhuma funcionalidade associada. Na criação destas *labels* teremos então o nosso *Icon* juntamente com a *string* composta pelo nome do Pokémon e os seus tipos. Aqui manipulamos também a fonte do nosso componente através de *setFont*, exemplificado na seguinte linha de código.

```
setFont(new Font("BigNoodleTitling", Font.PLAIN, 18));
```

Usamos os métodos *setBackground* para alterar a cor de fundo da *label*, *setHorizontalAlignament* e *setVerticalAlignment* para centrar o conteúdo.

Figura 15 – Exemplo de PokémonLabel

Tal como já referido o método *addPokemon* permite-nos adicionar Pokémon à lista do treinador recebido como argumento. Ainda com este método alteramos o aspecto corrente da *label* para indicar-nos que esta se encontra selecionada.

Figura 16 – Exemplo de PokémonLabel seleccionada

Conseguimos controlar se esta está selecionada ou não através da variável booleana *enabled*, podendo então adicionar o Pokémon da *label* à lista se *enabled* for *true* e remover o Pokémon da lista se *enabled* se encontrar a false.

Por fim temos o método *getColor* que simplesmente nos retorna um objecto *Color* com a cor *RGB* pretendida dependendo do tipo de Pokémon recebido como argumento. Aproveitámos também para mudar a cor da letra da *label* através de *setForeground* para as cores dos tipos *DRAGON* e *DARK* de forma a obtermos um contraste mais perceptível.

Com a criação das *JLabel* que vão conter a informação do Pokémon e tendo este mesmo associado, podemos construir o *JPanel* que irá conter a nossa listagem de Pokémon para o menu de selecção.

Figura 17 – Diagrama da classe PokemonPoolPanel

Esta classe deriva então de *JPanel* e faz uso do layout *GridLayout* que permite definir em quantas colunas e grelhas queremos dividir o nosso painel e o espaço entre estas.

```
setLayout(new GridLayout(5,2, 30, 30));
```

Nesta linha de código adicionamos um novo *GridLayout* com 5 linhas e 2 colunas, com um distanciamento vertical e horizontal de 30 pixeis entre cada.

Com o método *buildPanel* percorremos toda a lista da nossa *PokemonPool* e construímos *PokemonLabel's* que são adicionadas ao painel. Neste método temos ainda de indicar que devem ser executadas acções quando clicarmos nestas *labels*. Enquanto no botão

tínhamos um *eventListener* para a *label* usamos um *mouseListener* onde damos *override* ao método *mouseClicked* da classe *MouseAdapter*. As operações que queremos executar é executar o método *addLabel* da *PokemonLabel* e ainda alterar o estado da sua variável booleana *enabled*.

7. TeamPanel

Figura 18– Diagrama da Classe TeamPanel

A classe *TeamPanel* deriva de *JPanel* e constrói os painéis laterais que serão aplicados no painel de batalha. Estes painéis irão representar as equipas de cada treinador e o estado das mesmas. Como os treinadores são de classes diferente são usados dois construtores para cada um destes para que cada painel receba tenha a lista de Pokémon correcta associada.

Neste painel utilizamos um *GridLayout* em que cada entrada da grelha será preenchida com uma *JLabel* da classe *PokemonLabel*. Tal como referido este painel pretende reflectir o estado actual do combate, pelo que necessita de ser actualizado. Aqui entram os métodos *refresh*. O que os métodos *refresh* fazem é remover todos os componentes, aplicá-los novamente e chamar 3 métodos:

- *Invalidate* – que invalida o contentor e todos os seus componentes.
- *Validate* - que revalida o contentor e todos os seus componentes, o que significa que todos os seus componentes são repostos.
- *Repaint* – que chama o método *paint* do componente e apresenta-o com os seus novos componentes.

Quando executamos os métodos *refresh* pretendemos dar *disabel* às *labels* de Pokémon que já não possam combater para dar essa indicação visual. De notar, no entanto, que este pode não ser o método mais correcto para actualizar um componente swing mas foi o método que encontrámos funcional.

Figura 19– Exemplo de TeamPanel e TeamPanel com uma label disabled

8. EventPanel

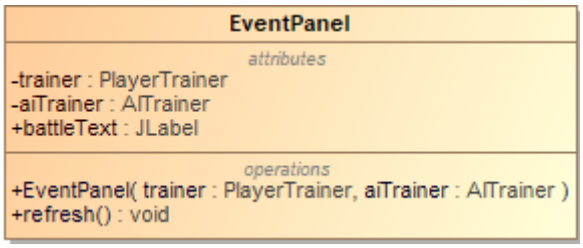


Figura 20– Diagrama da classe EventPanel

EventPanel deriva de *JPanel* e é a classe responsável por construir os elementos que irão representar os Pokémon activos em combate (imagem, nome, status e hp) e o texto de combate.

Como podemos verificar teremos então os dois treinadores como argumentos do construtor para que possamos manipular a informação de cada Pokémon activo. Aqui temos novamente o método *refresh* que funciona exactamente como que já foi descrito anteriormente no *TeamPanel*. No entanto aqui optámos por construir o painel directamente no método *refresh* sem que a sua construção inicial tenha sido descrita no construtor.

Neste painel utilizamos o layout *BorderLayout* e fazemos uso das secções *NORTH*, *CENTER* e *SOUTH*. A secção norte simplesmente recebe uma *JLabel* que irá conter o texto de batalha, ou seja, irá indicar os ataques utilizados e se um Pokémon é derrotado. Nesta *label* no entanto fazemos uso de linguagem HTML para apresentar o texto centrado e com quebras de linhas.

```
eventPanel.battleText.setText("<html><body style='text-align: center;'> You sent out " + trainer.getActivePokemon().name +
        "<br>Opponent sent out " + aiTrainer.getActivePokemon().name + "</html></body>");
```

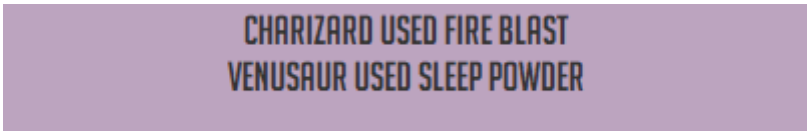


Figura 21– Exemplo da aplicação da linguagem HTML e representação de battleText

Na secção central do painel vamos ter um *JPanel* que fará uso do *GridLayout*, onde terá apenas uma linha e duas colunas. Cada uma destas entradas da grelha terá uma *label* onde será definido, tal como na *PokemonLabel*, uma imagem como *icon*, que neste caso será uma imagem que representa de forma mais detalhada o Pokémon associado.

Figura 21– Exemplo do Painel com as imagens dos Pokémon activos

Finalmente na secção inferior iremos introduzir um *JPanel* que fará uso do *GridBagLayout*. Neste painel teremos labels com o nome dos Pokémon activos que podem ainda apresentar o status destes, caso exista. Aqui introduzimos um novo componente da biblioteca *jswing*, *JProgressBar*. Este componente é usado para simular uma barra de progresso, mas neste caso será usada para simular uma barra de pontos vida. Quando criamos uma *JProgresBar* usamos com argumentos o valor mínimo e o valor máximo desta, podendo definir o valor actual com o método *setValue*. Podemos definir as cores da representação visual da barra através dos seguintes comandos e obter o resultado final:

```
UIManager.put("ProgressBar.background", Color.BLACK);
UIManager.put("ProgressBar.foreground", Color.GREEN);
UIManager.put("ProgressBar.selectionBackground", Color.RED);
UIManager.put("ProgressBar.selectionForeground", Color.WHITE);
```

Figura 22– Comandos para alterar o visual da JProgressBar

9. BattlePanel

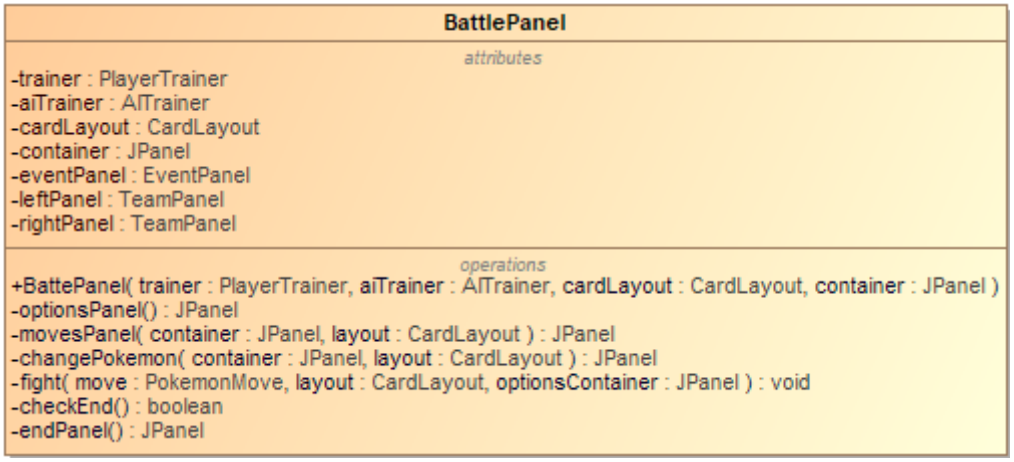


Figura 23– Diagrama da classe BattlePanel

A classe *BattlePanel* deriva de *JPanel* e é responsável por todos os menus e botões que controlam as acções da aplicação. Esta classe também acaba por substituir a classe *Battle* da Parte A fazendo uso adaptado dos seus métodos.

Esta classe tem uma extensa lista de argumentos:

- *trainer* e *aiTrainer* que são os nossos treinadores, *PlayerTrainer* e *AITrainer* respectivamente.
- *cardLayout* que é o layout do nosso contentor principal que já vem desde a classe UI
- *container* que é o próprio contentor principal da aplicação
- *eventPanel* que é o painel de eventos da classe *EventPanel*
- *leftPanel* e *rightPanel* que serão *TeamPanel's* que irão ocupar posições laterais

O painel desta classe irá assumir um *Borderlayout* em que a secção central terá o nosso *eventPanel* e as secções laterais terão os nossos *TeamPanel's* (*rightPanel* e *leftPanel*). A secção inferior será composta por um painel de opções que por sua vez será composto por vários painéis que vão ser acedidos através de um *CardLayout*.

Este primeiro painel será construído no método *optionsPanel*, onde teremos então um painel contentor com um *CardLayout* e um painel com os botões associadas a determinadas acções num *GridLayout*. Teremos 3 botões nesta secção, tal como na Parte A temos a funcionalidade de iniciar um novo jogo, lutar ou trocar de Pokémon.

Figura 24– Painel optionsPanel

O botão *new game* faz uso do *CardLayout* do nosso contentor da aplicação principal construindo um novo *SelectionPanel* e tornando este como painel activo do layout. Ao mesmo tempo termina o áudio actual e inicia o áudio do menu de selecção.

O botão *Fight* vai criar um novo painel através do método *movesPanel* e tornar este painel como painel activo do *optionsPanel* através do *CardLayout*.

O botão *Change Pokémon* vai criar um novo painel através do método *changePokemon* e tornar este como painel activo do *optionsPanel* através do *CardLayout*.

Ao clicarmos no botão *Fight*, chamamos então o menu *movesPanel* que constrói um painel com *GridLayout* em que cada entrada da grelha será um botão dos 4 moves do Pokémon. No *actionListener* destes botões chamamos o método *fight* que terá associado a este o move correspondente. Após concluídas as acções deste método fazemos *refresh* ao *eventPanel* para disponibilizar os efeitos do move executado.

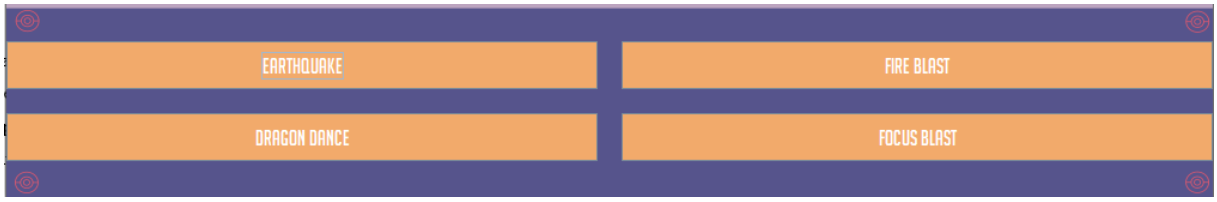


Figura 25– Painel movesPanel

O método *fight* é semelhante ao método que temos na classe *Battle* da Parte A do projecto, fazendo apenas algumas alterações sobretudo nas acções a tomar no fim de cada decisão. Enquanto na Parte A as acções eram exclusivamente tomadas para terem efeito na consola agora precisam de ser adaptadas para terem feito na interface. A cada decisão tomada vamos construído uma *String* que no final será usada no *battleText*. Caso o Pokémon activo do utilizador seja derrotado e este ainda tenha *Pokémon's* disponíveis somos levados para o painel *changePokémon*. Ao fim do move ser executado e todas as decisões associadas sejam tomadas, são efectuadas duas verificações, se algum dos treinadores foi derrotado e o jogo terminou, accionando a mudança de painel do *optionsPanel* para um painel final onde podemos começar um novo jogo, ou se a batalha irá prosseguir levando-nos de novo para o painel inicial do

optionsPanel, isto tudo com recuso ao *CardLayout*. No fim, é sempre feito um *refresh* ao *eventPanel*, *rightPanel* e *leftPanel* para que estes sejam actualizados e reflitam o estado actual do combate.

O método *changePokemon* constrói um painel com *GridLayout* que irá conter todos os Pokémon do utilizador e ainda um botão para este recuar para o painel anterior e seleccionar outra opção. Os Pokémon serão *JLabel* que terão um *mouseListener* associado para que ao clicarmos na *label* o Pokémon associado a esta se torne no Pokémon activo. Temos ainda a funcionalidade de a *label* do Pokémon activo ou de qualquer Pokémon se encontre derrotado estejam *disabled*.

Figura 26– Painel changePokemon

O método *checkEnd* é igual ao método da Parte A que simplesmente verifica se algum dos treinadores se encontra com a sua equipa toda derrotada.

Por fim temos o método *endPanel* que constrói o painel de fim de jogo, recorrendo a um painel contento com um *BorderLayout* em que o painel central usa um *GridBagLayout*. O painel central é composto por duas *JLabel* que indicam que o jogo terminou e quem é o vencedor e por fim um *JButton* que nos permite iniciar um novo jogo tal como o botão New Game do *optionsPanel*.

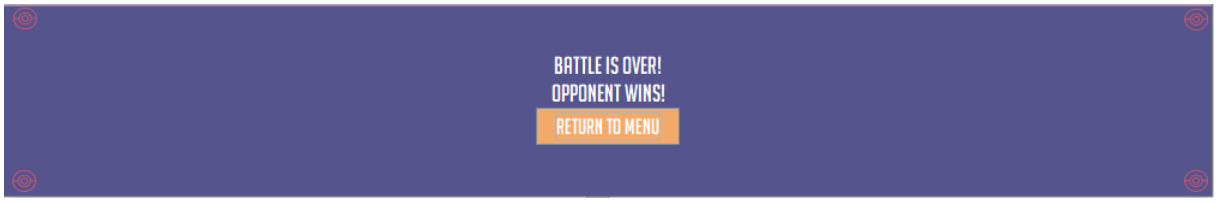


Figura 27– Painel endPanel

11. ImagePanel

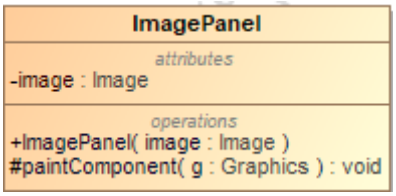


Figura 26– Diagrama da classe ImagePanel

A classe ImagePanel deriva de JPanel e recebe como argumento uma Imagem. Ao fazermos override ao método paintComponent de JLabel podemos pintar o background dos painéis desta classe com a imagem recebida como argumento. No entanto para podermos visualizar a imagem de fundo necessitamos de definir todos os outros painéis de nível superior com o método setOpaque a false, que faz com que o fundo destes não seja pintado.

```
File file = new File("../images/backgroundImage.png");
BufferedImage image = ImageIO.read(file);
JPanel container = new ImagePanel(image);
```

Figura 27– Exemplo da criação de um ImagePanel

11. Conclusões

Com a conclusão da Interface Gráfica damos como terminado o projecto final da unidade Curricular. A realização deste projecto permitiu-nos adquirir conhecimentos para construir qualquer tipo de interfaces gráficas para qualquer tipo de aplicação.

O uso do componentes jswing juntamente com a combinação de Layouts permitem-nos alcançar qualquer tipo de disposição de elemento que tenhamos interesse em desenvolver. Esta biblioteca é de uso fácil e que nos permite também facilmente adicionar eventos e acções através de listeners nos componentes, tornando a aplicação dinâmica.

Também conseguimos ver que a biblioteca base do Java nos permite alcançar diversos efeitos sem ser necessário recorrer a bibliotecas externas, isto verificámos especialmente quando conseguimos introduzir música na aplicação.

Podemos concluir que desenvolvemos uma interface gráfica que complementa todas as funcionalidades desenvolvidas na Parte A do projecto, com espaço para adicionar ainda mais funcionalidades se necessário.