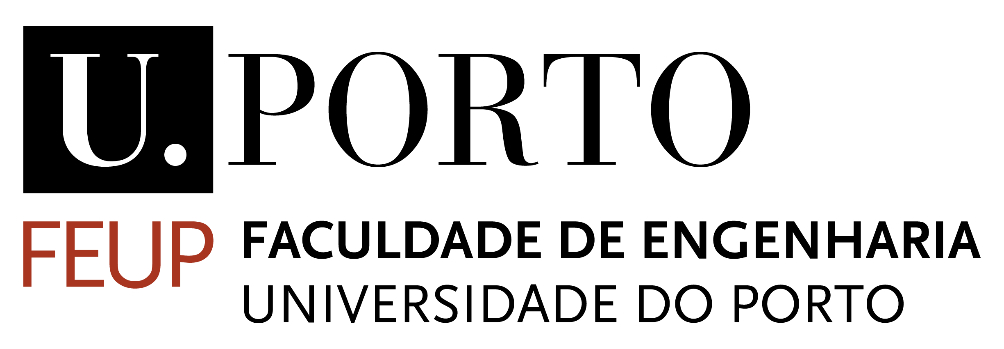
**Faculdade de Engenharia da Universidade do Porto**



**Laboratório de Programação Orientada por Objetos**

**MIEIC**

**Color Ninja**

**07/06/2016**

**Turma:** *2MIEIC05*

**Autores:**

José Aleixo Peralta da Cruz ([up201403526@fe.up.pt](mailto:up201403526@fe.up.pt))

José Miguel Matos Lopes da Costa ([up201402717@fe.up.pt](mailto:up201402717@fe.up.pt))

Índice

[Índice de Figuras 3](#_Toc453276804)

[1. Introdução 4](#_Toc453276805)

[2. Manual de Utilização 5](#_Toc453276806)

[Lista de Funcionalidades Suportadas 5](#_Toc453276807)

[Instalação e Arranque do Programa 5](#_Toc453276808)

[Modo de Utilização 6](#_Toc453276809)

[3. Conceção, Implementação e Teste 9](#_Toc453276810)

[Estrutura de Packages 9](#_Toc453276811)

[Estrutura de Classes 10](#_Toc453276812)

[*Package “states”* 10](#_Toc453276813)

[*Package “logic”* 11](#_Toc453276814)

[*Package “physics”* 11](#_Toc453276815)

[*Package “sprites”* 12](#_Toc453276816)

[*Classe “MyColorNinja”* 13](#_Toc453276817)

[Padrões de Desenho Utilizados 13](#_Toc453276818)

[Mecanismos e Comportamentos Importantes 14](#_Toc453276819)

[Ferramentas, Bibliotecas e Tecnologias Utilizadas 15](#_Toc453276820)

[Dificuldades Encontradas e sua Resolução 15](#_Toc453276821)

[4. Conclusões 16](#_Toc453276822)

[5. Referências 17](#_Toc453276823)

# Índice de Figuras

[Figura 1 - Menu de opções 6](file:///C:\Users\josea\Documents\GitHub\LPOO-Feup\%23FinalProject\Relatório.docx#_Toc453277228)

[Figura 2 - Menu inicial 6](file:///C:\Users\josea\Documents\GitHub\LPOO-Feup\%23FinalProject\Relatório.docx#_Toc453277229)

[Figura 3 - Exemplos de formas no jogo 7](file:///C:\Users\josea\Documents\GitHub\LPOO-Feup\%23FinalProject\Relatório.docx#_Toc453277230)

[Figura 4 - Menu de pausa 7](file:///C:\Users\josea\Documents\GitHub\LPOO-Feup\%23FinalProject\Relatório.docx#_Toc453277231)

[Figura 5 - Ecrã de jogo 7](file:///C:\Users\josea\Documents\GitHub\LPOO-Feup\%23FinalProject\Relatório.docx#_Toc453277232)

[Figura 6 - Ecrã de final de jogo 8](file:///C:\Users\josea\Documents\GitHub\LPOO-Feup\%23FinalProject\Relatório.docx#_Toc453277233)

[Figura 7 - Relações entre packages 9](file:///C:\Users\josea\Documents\GitHub\LPOO-Feup\%23FinalProject\Relatório.docx#_Toc453277234)

# Introdução

Com este relatório pretendemos apresentar o nosso trabalho realizado no âmbito do segundo projeto de Laboratório de Programação Orientada a Objetos, um jogo chamado *Color Ninja*.

O jogo consiste em eliminar figuras “atiradas” para o ecrã numa certa sequência. A nossa inspiração foi o famoso jogo *Fruit Ninja*.

Apresentamos as instruções básicas para a execução da aplicação, descrevendo as suas diversas funcionalidades, e ainda a descrição dos modelos de conceção em UML e dos padrões de desenho utilizados.

# Manual de Utilização

## Lista de Funcionalidades Suportadas

A nossa aplicação permite ao utilizador jogar em modo *single-player*, podendo escolher até três níveis de dificuldade do jogo. Poderá pausar o jogo, navegar pelo menu de definições e retomar o jogo que tinha sido parado ou inicializar outro. Além disso, todas as melhores pontuações em cada dificuldade são armazenadas pela aplicação.

No menu de opções, também é permitido ao jogador alterar o volume dos sons e da música do jogo.

## Instalação e Arranque do Programa

Para instalar o programa do projeto num dispositivo Android, basta inicializar a funcionalidade *debugging* do dispositivo, copiar o ficheiro “android-debug.apk” para o terminal e executá-lo, de forma a instalar a aplicação.

A aplicação foi desenhada tendo como versão destino do Anrdoid a versão 6.0 (*Marshmallow*), mas confirma-se a sua funcionalidade na versão 4.4 (*KitKat*).

Depois de instalada, a aplicação pode ser corrida através do ícone “ColorNinja”, que deverá aparecer no menu das aplicações.

## Modo de Utilização

Ao iniciar a aplicação, é apresentado o menu inicial ao utilizador. A partir daqui este pode optar por começar um novo jogo, selecionando a opção *New Game,* ir para o menu das definições, clicando em *Settings* ou ver as pontuações mais altas feitas anteriormente, clicando em *Scores*.



Figura 1 - Menu de opções



Figura 2 - Menu inicial

No menu *Settings* o utilizador pode, alterar o volume dos efeitos sonoros através do *slider* abaixo de *Sounds Volume,* ou da música em *Music Volume*.

Pode também alterar a dificuldade selecionando uma das *checkbox*es abaixo de *Difficulty*.

Ao premir o botão no canto superior esquerdo, volta ao menu inicial.

Ao clicar em *New Game* o jogo inicia com o ecrã da figura 5, onde os objetos são lançados aleatoriamente. Os dois valores na parte superior de ecrã representam e o número de vidas (figuras que podem cair sem serem destruídas). O jogo acaba quando as vidas acabarem.

O símbolo no canto superior direito representa o botão de pausa, que após clicado leva ao ecrã da figura 5. No menu de pausa os quatro botões permitem, retomar o jogo, recomeçar um novo jogo, ir para o menu das definições (previamente descrito), e ir para o menu inicial.



Figura 3 - Exemplos de formas no jogo



Figura 4 - Menu de pausa



Figura 5 - Ecrã de jogo

O jogo termina quando o utilizador perde as 3 vidas que tem, não tocando nos objetos antes de eles voltarem a cair. Caso a melhor pontuação associada à dificuldade do jogo tenha sido batida, aparece a devida mensagem ao utilizador. É depois possível reiniciar um jogo ou regressar ao menu inicial.



Figura 6 - Ecrã de final de jogo

# Conceção, Implementação e Teste

## Estrutura de Packages



Figura 7 - Relações entre packages

O pacote “states” é o *backbone* do jogo. Lá definem-se os atributos da aplicação, consoante o estado em que se encontra. Esses atributos fazem parte ou são manipulados pelas classes dos outros pacotes.

O pacote “physics” contém uma classe que implementa a física do LibGDX no jogo. A “logic” pertencem certos campos que guardam a informação do jogo. Em “sprites” está desenvolvida a interface e a classe que representa as formas que são atiradas durante o jogo. No pacote “ui” estão elementos que são utilizados na interface gráfica do jogo, também baseados nas bibliotecas LibGDX.

## Estrutura de Classes

### *Package “states”*

|  |  |
| --- | --- |
| Classe | Descrição |
| GameStateManager | Classe constituída por uma pilha (*stack*) que armazena todos os estados de jogo, inserindo-os ou eliminando-os conforme necessário. O estado no topo desse pilha é o estado atual da aplicação. |
| State | Classe abstrata que define os métodos necessários a um estado de jogo. |
| GameState | Representa o estado da aplicação em que o jogo se encontra a decorrer e todas as alterações que o jogo acarreta. |
| EndGameState | Representa o final do jogo, que exibe se foi batida a melhor pontuação ou não. |
| MainMenuState | Representa o primeiro estado da aplicação, que é o menu inicial, e que dá a possibilidade de escolher começar o jogo, alterar as opções de jogo ou visualizar as melhores pontuações. |
| PauseMenuState | Representa o menu que é apresentado ao pausar o jogo, que permite retomá-lo, recomeça-lo, alterar-lhe as propriedades ou voltar ao menu inicial. |
| ScoresMenuState | Representa o menu com as melhores pontuações do jogo para cada dificuldade. |
| SettingsMenuState | Representa a interface que permite alterar o som da música e dos tons da aplicação, assim como da dificuldade do jogo. |

### *Package “logic”*



|  |  |
| --- | --- |
| Classe | Descrição |
| Score | Classe que representa a pontuação de um jogo, segundo um conjunto <pontuação, dificuldade>, e que possui métodos para criar, comparar e mostrar estas pontuações. |
| Highscores | Classe que contém as maiores pontuações associadas às várias dificuldades e que é responsável por adicionar, gerir e mostrar novas pontuações. |
| IntegerCounter | Classe que representa um contador gráfico de valores inteiros, utilizado para mostrar a pontuação atual do jogo e as vidas do jogador. |

### *Package “physics”*



|  |  |
| --- | --- |
| Classe | Descrição |
| MyBox | Classe que o retângulo que rodeia o ecrã de jogo e onde as formas colidem. É um conjunto de três linhas (duas laterais e uma no topo) que implementam a física do LibGDX. |

### *Package “sprites”*

|  |  |
| --- | --- |
| Classe | Descrição |
| AbstractFactory | Interface que declara os métodos para criar os objetos que são lançados no jogo. |
| ShapeFactory | Classe implementa a *AbstractFactory*, tendo métodos que são responsáveis por criar formas com determinadas características, sejam elas definidas ou aleatórias. |
| Shape | Classe que representa o objeto que é atirado durante o jogo, e que pode variar na forma e na cor. Possui também os atributos e os métodos físicos e lógicos destes objetos. |
| Animation | Classe que define cria uma animação a partir de uma *spritesheet* e de um tempo de animação. |

### *Classe “MyColorNinja”*



|  |  |
| --- | --- |
| Classe | Descrição |
| MyColorNinja | Classe que representa a aplicação Android em si. É um *singleton* que determina aquilo que é mostrado no ecrã e aquilo que já é dispensável. Possui as variáveis globais do programa e armazena informações que sejam necessárias em vários *packages*. |

## Padrões de Desenho Utilizados

Para o desenvolvimento das figuras que são atiradas para o ecrã foi usado

***Factory Method*** para permitir vários tipos e várias cores diferentes.

Foi utilizado o padrão ***Singleton*** na definição do gestor de recursos do jogo

(MyColorNinja.java) por questões de eficiência, cada recurso utilizado pelo jogo deve ser carregado uma única vez e todos os recursos deverão estar acessíveis a partir de um único gestor de recursos, para garantir que não há redundância nesse carregamento.

Foi também utilizado o padrão ***State*** com o package “states” e o MyColorNinja.java, para permitir a alteração entre os diferentes estados possíveis (*MainMenu,Settings,Game*,etc)

E por último, ***Memento*** com o *GameStateManager* para capturar o estado interno dos objetos para após pausar o jogo, conseguir retomar o mesmo.

## Mecanismos e Comportamentos Importantes



O ponto de partida para a elaboração do nosso código foi considerar o jogo como sendo uma máquina de estados, em que cada estado seria o equivalente a uma *Activity* do Android.

Quando o jogo é inicializado, o primeiro menu a aparecer é o menu inicial. Ao clicar num qualquer botão, faz-se a transição entre os estados do jogo.

Para gerir os estados, usamos uma *stack* de *State*, implementada na classe *GameStateManager*. Quando interessa guardar um determinado estado antes de mudar o estado do jogo, fazemos *push* do novo estado. Quando é dispensável, fazemos um *pop* primeiro. Assim é possível recuar de um estado para um estado guardado previamente, que é útil, por exemplo, quando se pausa o jogo.

## Ferramentas, Bibliotecas e Tecnologias Utilizadas

Para a Conceção do Projeto, o IDE utilizado foi o **Android Studio** e utilizou-se a biblioteca **LibGDX,** de forma a facilitar a implementação do módulo de física do jogo e o desenvolvimento para Android.

## Dificuldades Encontradas e sua Resolução

Relativamente a dificuldades, foi particularmente mais complicado perceber como os sliders eram usados para controlar o volume, mas após uma certa pesquisa foi conseguido.

Além disso, não nos foi possível concretizar quaisquer testes unitários usando o JUnit. Deliberamos que isto se deve à utilização de certas funcionalidades do LibGDX, que não permitem que os testes unitários analisem o código de forma correta e nunca deem luz verde.

# Conclusões

Ao início, pensámos fazer o jogo de forma diferente, impondo uma sequência segundo a qual as figuras tinham de ser destruídas, mas depois optamos por não o fazer e ser apenas necessário apenas destruir as figuras que fossem lançadas.

Algumas melhorias possíveis seriam a adição de modos de jogo diferentes, incluindo o *multiplayer* e o contra-relógio, a possibilidade de alterar a *skin* dos objetos que são lançados e adicionar um conjunto de “proezas” que permitiriam visualizar o progresso do jogador.

As tarefas foram divididas igualmente por ambos os membros do grupo tendo o nível de participação e de contribuição para o projeto sido igual por parte de ambos.

# Referências

Grill, T. (Junho de 2016). *TomGrill - GDX Testing*. Obtido de GitHub: https://github.com/TomGrill/gdx-testing

Zechner, M. (Abril de 2016). *LibGDX Bad Logic Games*. Obtido de https://libgdx.badlogicgames.com/