|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Castle War** | ISEC – Programação Orientada a Objectos | Pedro Coelho a21200930 | 2016/2017 |  |
|  | | | | |

Conteúdo

[Introdução 4](#__RefHeading___Toc308_1707381404)

[Enquadramento 4](#__RefHeading___Toc390_1280653743)

[Objetivos 4](#__RefHeading___Toc522_40103207)

[Visão Geral 4](#__RefHeading___Toc524_40103207)

[Classes implementadas 5](#__RefHeading___Toc314_1707381404)

[Game 5](#__RefHeading___Toc316_1707381404)

[Builder 5](#__RefHeading___Toc318_1707381404)

[BoardPiece 6](#__RefHeading___Toc320_1707381404)

[Ser 6](#__RefHeading___Toc526_40103207)

[Edificio 6](#__RefHeading___Toc528_40103207)

[Castelo 6](#__RefHeading___Toc530_40103207)

[Torre 6](#__RefHeading___Toc532_40103207)

[Quinta 6](#__RefHeading___Toc534_40103207)

[Colonia 6](#__RefHeading___Toc392_1280653743)

[Perfil 6](#__RefHeading___Toc536_40103207)

[Modifier 7](#__RefHeading___Toc538_40103207)

[Command 7](#__RefHeading___Toc540_40103207)

[Descrição dos ficheiros 8](#__RefHeading___Toc330_1707381404)

[Ficheiros de Header(.h): 8](#__RefHeading___Toc332_1707381404)

[game.h 8](#__RefHeading___Toc396_1280653743)

[perfil.h 8](#__RefHeading___Toc398_1280653743)

[colonia.h 8](#__RefHeading___Toc400_1280653743)

[boardPiece.h 8](#__RefHeading___Toc402_1280653743)

[command.h 8](#__RefHeading___Toc542_40103207)

[utils.h 8](#__RefHeading___Toc544_40103207)

[Ficheiros de Source(.c): 9](#__RefHeading___Toc334_1707381404)

[main.c 9](#__RefHeading___Toc404_1280653743)

[game.c 9](#__RefHeading___Toc406_1280653743)

[perfil.c 9](#__RefHeading___Toc408_1280653743)

[boardpiece.c 9](#__RefHeading___Toc410_1280653743)

[command.c 9](#__RefHeading___Toc412_1280653743)

[utils.c 9](#__RefHeading___Toc546_40103207)

[Opções Tomadas 10](#__RefHeading___Toc340_1707381404)

[Game::Builder 10](#__RefHeading___Toc454_185386433)

[Modifiers 10](#__RefHeading___Toc548_40103207)

[Revisões 11](#__RefHeading___Toc550_40103207)

[Command 11](#__RefHeading___Toc552_40103207)

[Command::validate() 11](#__RefHeading___Toc554_40103207)

[board como atributo de Game 11](#__RefHeading___Toc556_40103207)

[ncurses 11](#__RefHeading___Toc558_40103207)

[Colónias controladas pelo computador 11](#__RefHeading___Toc880_1335843804)

[Considerações Finais 12](#__RefHeading___Toc344_1707381404)

# Introdução

## Enquadramento

Este trabalho foi realizado no âmbito da disciplina de Programação Orientada a Objetos do 2º ano, 1º semestre de Licenciatura em Engenharia Informática do ano letivo de 2016-2017. É pretendido com este relatório descrever todo o trabalho produzido e toda a organização do mesmo. O trabalho foi desenvolvido em sistema linux, utilizando o g++ como compilador.

## Objetivos

O objetivo principal do trabalho foi desenvolver um programa em C++ para visualizar o conhecimento adquirido resultante da matéria lecionada no âmbito da disciplina.

## Visão Geral

Na abordagem deste trabalho tentou-se tirar o máximo proveito do paradigma de programação orientada a objetos, para isso foram criadas diversas classes tentando representar o mais aproximadamente o problema proposto. Explico agora, o mais abstratamente possível esta abordagem.

Foi criada uma class Game que contem toda a informação referente a uma simulação, assim como uma class Colónia. De modo a representar as “peças” do jogo foi criada uma class BoardPiece que tem como filhas as classes Ser e Edificio, esta ultima, por sua vez, tem como filhas as classes Castelo, Quinta e Torre. Existe também uma class Perfil, que representa o perfil definido pelo utilizador, contendo vectores de modificadores, representados pela class Modifier. Foi ainda criada a class Command que é utilizada para a manipulação da “linha de comandos” que serve de interface da simulação.

# Classes implementadas

## Game

A class Game foi criada para conter toda a informação relativa a uma simulacão. Atributos relevantes:

* Class Builder (ver class seguinte);
* perfilList – vector de ponteiros para perfil, que contem os perfis criados pelo utilizador;
* board – é do tipo vector de column, que é um “alias” para ponteiros de BoardPiece, ou seja, a board é um vector de vectores de ponteiros para boardpiece (ver secção board);
* colonias – vector de ponteiros para colonia, que contem a lista de colonias existentes na simulação

Mais tarde deparei-me com vários problemas derivados da implementação da board como atributo da class game em vez de criar a sua própria class, como mais a frente será explicado.

## Builder

Esta class Permite que, durante a fase de configuração, os dados sejam alterados mais que uma vez, para que, uma vez concluída, se crie o objecto da class Game.

## BoardPiece

Representa todas as “peças” que tomam ações no tabuleiro, tendo como filhas as seguintes classes:

### Ser

### Edificio

A class Edificio tem por sua vez 3 filhas, representando os respetivos tipos de Edificio:

### Castelo

### Torre

### Quinta

## Colonia

Esta class contem a informação referente a cada colónia presente na simulação. Atributos relevantes:

* castelo – ponteiro para o castelo da colónia;
* perfilList – vector de ponteiros para os perfis definidos pelo jogador na fase de configuração;
* edificiosList - vector de ponteiros para os edifícios da colónia;
* seresList – vector de ponteiros para os seres da colónia

## Perfil

A class perfil representa o perfil definido pelo utilizador na fase de configuração, atributos relevantes:

* passiveMods – vector de ponteiros para os modificadores passivos;
* activeMods – vector de ponteiro para os modificadores ativos.

## Modifier

Representa todos os modificadores (características) que o utilizador pode escolher para colocar num perfil, cada tipo de modificador é, por sua vez, representado pela sua própria class (filha de Modifier) de modo a executar as suas funções. Tem como filhas as seguintes Classes: **Bandeira**; **Superior**; **PeleDura**; **Armadura**; **Faca**; **Espada**; **Agressão**; **Ecologico**; **HeatSeeker**; **BuildSeeker**; **Walker**; **Remedio**; **SecondChance** e **Aluno**.

## Command

Por fim temos a class Command que serve para representar os comandos introduzidos pelo utilizador, tanto na fase de configuração como na fase de simulação.

# Descrição dos ficheiros

## Ficheiros de Header(.h):

### game.h

* + É definida a class Game e Builder;

### perfil.h

* + São definidas as classes Perfil, Modifier e todas as suas filhas;

### colonia.h

* + É definida a class Colonia;

### boardPiece.h

* + É definida a class Boardpiece e todas as suas filhas;

### command.h

* + É definida a class Command;

### utils.h

* + É definida a estrutura Point;
  + são declarados os protótipos das funções utilitárias à execução do programa;

## Ficheiros de Source(.c):

### main.c

* + Contem o código de configuração de nova simulação;

### game.c

* + Contém o código de execução da simulação;

### perfil.c

### boardpiece.c

### command.c

### utils.c

# Opções Tomadas

## Game::Builder

Quando decidi criar uma class Game, na qual o jogo se ia desenrolar, achei que depois da configuração fazia sentido alguns dos atributos da class serem constantes, isto trouxe alguns problemas de implementação, pois durante a configuração o utilizador pode alterar valores já definidos, para isso, depois de alguma pesquisa, encontrei esta solução, também chamada de *builder design pattern*. A class Builder permite definir, arbitrariamente, os valores que vão ser usados quando chamado o construtor da class Game, e assim resolver o problema inicial.

## Modifiers

Decidi que a melhor maneira de implementar as características seria criar uma serie de classes que representam as mesmas e todas elas herdarem da class mãe Modifiers, de modo a poder colocar nos perfis criados toda a informação necessária para executar as ações. Quando chamados os respetivos métodos dessas classes é executada a ação respectiva.

# Revisões

Devido a alguma falta de tempo não me foi possível entregar um trabalho finalizado nem resolver da melhor maneira alguns problemas com que me deparei. Ficam algumas alterações e ideias que gostaria de ter implementado:

## Command

Esta class ficou um pouco sobrecarregada, isto porque trata dos commandos da configuração e da simulação ao mesmo tempo, deveria ter feito duas classes filhas, uma para cada fase.

## Command::validate()

Este método ficou confuso devido ao explicado em cima e também porque não usei funções em separado para validar diferentes tipos de input.

## board como atributo de Game

Deveria ter criado uma class Board de modo a facilitar a sua interação com as outras classes (principalmente Modifiers)

## ncurses

Conforme falado na apresentação, iria utilizar a biblioteca ncurses para tratar da interface com o utilizador, apesar de ter aprendido as bases para utilizar essa mesma biblioteca, o qual poderei exemplificar na apresentação, não tive tempo de implementar no meu trabalho.

## Colónias controladas pelo computador

Infelizmente não tive tempo de implementar as colónias controladas pelo computador, embora tenha deixado previsto no código, preparado funções e comentado ideias de como implementar, algumas das situações necessárias.

# Considerações Finais

O trabalho enviado está um pouco incompleto, devido a, entre outras coisas, ter ficado sem parceiro a meio do semestre. Gostaria de ter tido mais tempo para finalizar, e entregar um trabalho cujo resultado me deixasse mais satisfeito. Ainda assim, penso que estruturalmente entrego um bom trabalho, e aprendi bastante com a sua realização, terei todo o prazer de discutir as opções tomadas na apresentação.