DEPARTAMENTO DE ENGENHARIA
INFORMÁTICA E DE SISTEMAS
INSTITUTO SUPERIOR DE ENGENHARIA DE
COIMBRA
2016/2017

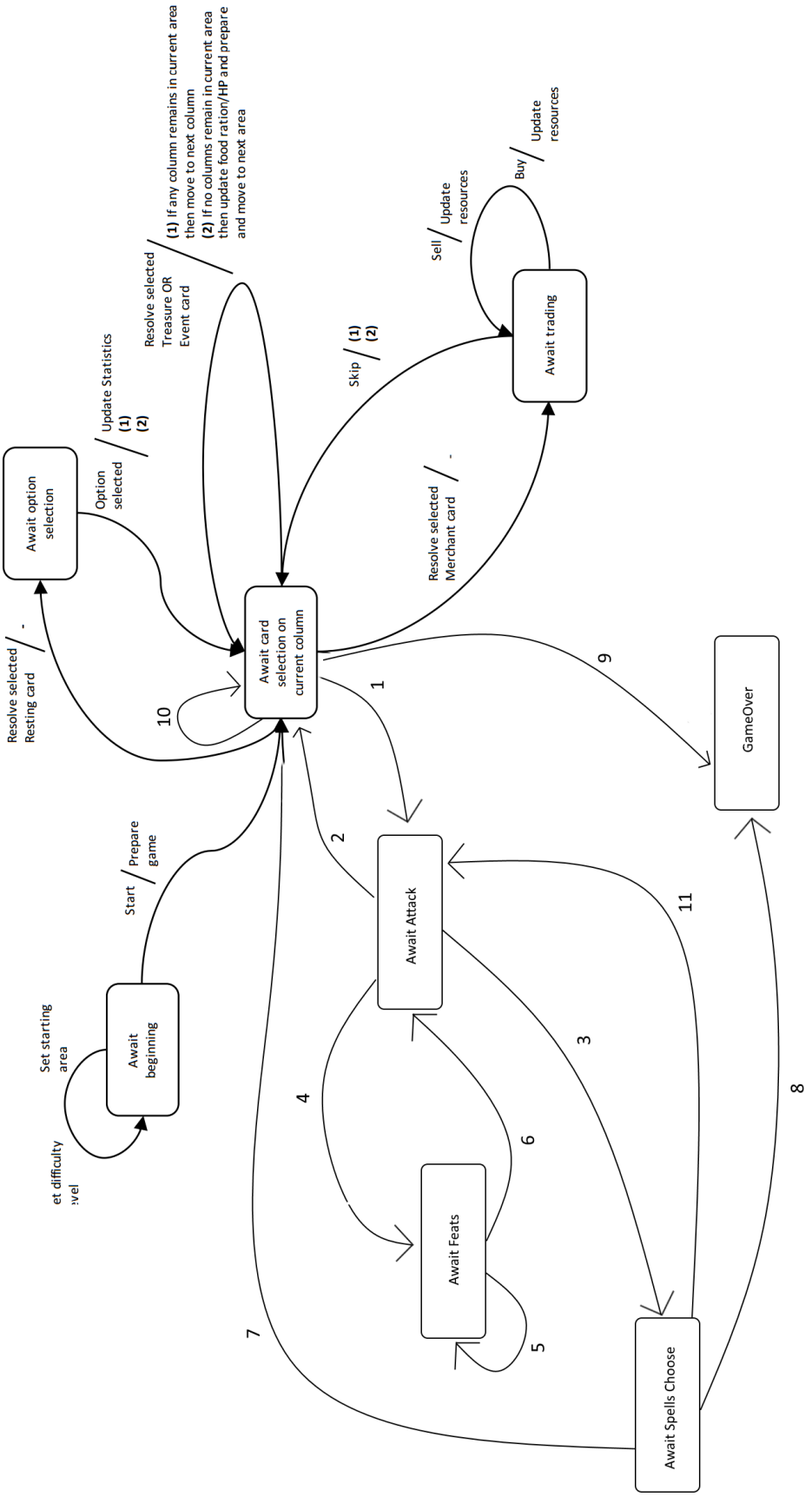
Programação Avançada
Trabalho Prático
-Fase 1-
Engenharia informática

Eduardo Fernandes nº 21250192

Tiago Coutinho nº 21250388

Coimbra, 12 de maio de 2017

Máquina de Estados



Descrição da Máquina de Estados:

1. Resolve Selected Monster Card or Boss Card / Lança dados
2. Win Battle / Se monstro morre
3. Attack Monster / Se monstro não morre
4. Choose Dice to Feat
5. Choose Another Dice / Se ainda tem mais dados para Feat
6. Return to Battle / Se não tem mais dados para Feat
   / Se escolher sair
7. Battle Won / Se monstro morre
8. Death / Se personagem não tem suficiente HP para ataque do Monstro
9. Fim do Jogo / Caso personagem não tem HP.
10. Resolve Trap Card
11. Next Round / Se nem Personagem e Monstro morrem

Classes Utilizadas

Personagem: Classe responsável por armazenar informação relativa à personagem do jogo. Contém os vários *stats* da personagem (*armor, hp, gold, food, rank* e *xp*) bem como um *ArrayList* que armazena os vários *spells* da personagem. Além disso contém também uma *flag* que indica se a personagem tem o *spell poison* ativo.

Area: Classe responsável por armazenar a informação relativa às diversas áreas do jogo. Contém o baralho das *Room Cards*, a coluna em que a personagem se encontra, assim como uma *flag* que indica se a personagem já derrotou um monstro ou não a ser verificada na carta Treasure.

Caverna: Classe responsável por armazenar a informação relativa à Caverna. Contém um *ArrayList* com todas as áreas do jogo, bem como o nível e número da área atual. É nestas classes que existem os métodos para gerar as áreas, bem como o *Pit*, opção 6 da carta *Trap*, entre outros métodos menos relevantes.

Dado: Classe responsável por armazenar informação relativa aos dados. Contém o valor atual do dado, o total de valor do dado (para os *rerolls*) bem como uma *flag* que indica se o dado já teve Feat ou não. Contém também todos os métodos responsáveis pela manipulação destas variáveis.

Export: Classe que apenas contém dois métodos responsáveis pelo o Export/Pausa do jogo, a leitura e a escrita do ficheiro binário, o qual guarda o objeto GameData e o estado atual, ambos presentes na classe Jogo.

GameData: Principal classe do jogo para armazenar informação. Contém a personagem do jogo, caverna, nível de dificuldade e área inicial. Para além disso contém também um *ArrayList* com os dados que a personagem tem ao dispor. Contém ainda uma variável do tipo Carta que guarda o monstro alvo ao longo do jogo e contém também uma *string* msg para simplificar o envio de mensagem para a *Interface* com o utilizador.

Jogo: Classe responsável por armazenar o estado atual bem como o *gamedata*. Contém vários métodos responsáveis pela alteração de estados bem como um conjunto de métodos que permite aceder à interface aceder à informação. No fundo, esta classe funciona como intermediária entre os dados e a *interface*.

Carta: Classe abstrata que corresponde às cartas do jogo.

BossMonster e Monster: Classes derivada da classe Carta. Contém a *hp* e damage dos *monsters*, assim como as recompensas por ultrapassar esta carta. Para além disso contém os métodos responsáveis com estas variáveis, contendo ainda os métodos *toString()* e *infoCarta()* que serão depois úteis na *interface*.

Event: Classe derivada da classe Carta. Contém o mesmo tipo de informação que as 2 classes anteriores, uma vez que existe a possibilidade de ao resolver esta carta, ter de enfrentar um monstro.

Merchant, Resting, Trap e Treasure: Classes derivadas da classe Carta. Estas classes contêm apenas os métodos *toString()* e *infoCarta()*.

AdaptadorCartas: Classe que contém todas as funções abstratas da classe Carta de modo a que não seja preciso estarem nas classes derivadas que não precisem delas.

Spell: Classe abstrata que corresponde aos *spells* do jogo.

Fire, Healing, Ice e Poison: Classes derivadas da classe *Spell*. Contém métodos para obter o nome, a descrição do *spell* e principalmente o efeito de cada um dos *spells*.

StateAdapter: Classe que contém todas as funções de todos os estados de modo a que não seja preciso estarem nos estados que não precisem delas.

AwaitBeggining: Estado inicial do jogo onde são definidas algumas definições. Contém os métodos necessários para alterar a área inicial, alterar o nível de dificuldade e começar o jogo.

AwaitCardCardSelectionOnCurrentColumn: Estado em que é escolhida uma carta da coluna em que o jogador se encontra. Contém os métodos necessários para resolver cada um dos tipos de *room cards* existentes. Para além disso contém a função *skill check*.

AwaitOptionSelection: Estado em que o utilizador escolhe a opção da carta *resting*. Contém o método que resolve a *room card resting*.

AwaitTraiding: Estado em que o utilizador efetua as trocas comerciais que desejar. Contém o método que resolve a *room card traiding*.

GameOver: Estado final do jogo, quer em caso de vitória ou derrota por parte do utilizador.

AwaitAttack: Estado em que o utilizador pode decidir atacar o monstro, fazer *rerrol* ou *feat* um determinado dado.

AwaitFeats: Estado em que o utilizador escolhe o dado que pretende fazer *feat*, bem como a forma de pagamento.

AwaitSpellChoose: Estado em que o utilizador pode escolher utilizar um dos seus *spells*. Se o utilizador não tiver *spells* então este estado é ignorado.

Trab: Classe que contém a função *main*.

TextUI: Classe que contém toda a interface com o utilizador. Nesta classe é feita a interação com o utilizador, mostrando o estado do jogo e pedindo, ocasionalmente, *inputs* do teclado.

Constants: Para além das classes, existe também um ficheiro que guarda todas as constantes do jogo, tais como os níveis de dificuldade e respetivos *stats*, o mapa dos níveis da caverna de jogo, o número de níveis, o *ranking* de *xp*, informação da carta *monster* e *boss monster* e também o nome do ficheiro para gravar o jogo.

Funcionalidades não implementadas

Apenas a variante de jogo *"The Dungeon Keys"* não foi implementada.