

Programação Orientada a Objectos 2014/2015

Trabalho prático

1. Descrição geral do problema

Descrição inicial muito rápida

Pretende-se implementar um jogo constituído por elementos num mapa tipo quadriculado. Existem vários tipos de elementos, e cada tipo tem características e comportamentos próprios. Alguns elementos são fixos e nunca mudam de posição. Outros são móveis e mudam de posição no mapa com o passar do tempo. Alguns elementos são controlados pelo utilizador (jogador), outros são controlados pelo computador (fazem coisas aleatórias). O utilizador indica ao programa o que deseja que os seus elementos façam escrevendo comandos no teclado. Também é possível ler os comandos de um ficheiro de texto. O mapa é representado no ecrã. Cada posição no mapa ocupa mais do que um carácter. Não se consegue ver o mapa todo ao mesmo tempo, por isso vê-se apenas uma parte e pode-se mudar a parte que se pretende ver.

O trabalho é essencialmente o que foi descrito acima. O resto deste documento descreve os detalhes. Existem vários detalhes, e pretende-se dar uma descrição que cubra todos os aspectos possíveis para evitar ao máximo as dúvidas, sendo a descrição necessariamente alongada. Havendo muitos detalhes, se alguns não forem implementados, os restantes (os que são implementados) deverão ainda assegurar uma nota razoável.

Descrição completa do trabalho

Pretende-se implementar uma aplicação que permita ao utilizador jogar um jogo que faz lembrar vagamente a versão 1 (que faz agora 20 anos) do *Warcraft®* (exemplo disponível em www.classicdosgames.com/online/war1demo.html). Este jogo simula a luta entre populações inimigas (no jogo original: humanos e orcs, mas aqui só vai haver humanos) pelo domínio de território e de recursos. O cenário é um reino medieval anónimo, onde as várias populações (ou povoações) vivem harmoniosamente, pelo menos até encontrarem uma população inimiga. Seguem-se alguns tópicos descritivos, gerais e rápidos, sobre o ambiente do jogo/simulador:

- Existem várias personagens com características (comportamento e habilidades) específicas ao seu tipo (por exemplo, camponeses, soldados, etc.). Utilizar-se-á neste enunciado o termo “unidade” para referir genericamente um membro de uma população (exemplos: “um soldado é uma unidade de ataque”, ou “a população Vermelhos tem 40 unidades em jogo”).
- Pode haver duas, três ou mais populações. São todas inimigas entre si (as alianças ficam para a época especial, por exemplo).

- As unidades recebem ordens do jogador para desempenhar tarefas. As unidades das populações que não são controladas pelo jogador agem como se recebessem ordens (essas ordens são geradas de uma forma meio aleatória-meio inteligente a descrever mais adiante).
- Poderá haver um conjunto inicial de unidades para cada população (isto são pormenores de definição do cenário inicial do jogo). No entanto uma população poderá criar novas unidades, desde que tenha os recursos e edifícios necessário (ex.: para construir uma unidade “caçador” é preciso ter o edifício “academia” e 400 de “madeira”). Ao construir uma nova unidade, o edifício envolvido não se gasta, mas os recursos necessários são consumidos e desaparecem (no exemplo dado, ao fazer um caçador, desaparecem mesmo os tais 400 de madeira).
- Cada população poderá fazer edifícios para poder evoluir e expandir o seu domínio (castelo, quinta, etc.). Os edifícios possíveis são, essencialmente, os mesmos para cada raça e população.
- No reino medieval em que se desenrola a acção do enunciado, existem recursos que podem (e devem) ser apanhados (“minados”) para se poder fazer alguma coisa (por exemplos, os edifícios). Os recursos são naturais (ex.: pedra, madeira, ouro) e para os obter bastará ir até ao local onde eles se encontram (por exemplo, o ouro está numa mina), e transportar para o castelo da população.

Já a pensar em termos de implementação, adianta-se que uma coisa são membros da população, os quais são de vários tipos (soldado, camponês, cozinheiro, etc.), outra coisa são edifícios. Ambos têm comportamentos e podem fazer coisas, mas são, evidentemente, entidades substancialmente diferentes entre si. Ainda outra coisa diferente de ambas as anteriores são os recursos.

O objectivo do jogo é a sobrevivência da população e o extermínio total das restantes. É uma realidade dura, mas os reinos medievais, tal como agora, são locais duros para se viver.

2. Mecânica do jogo/simulador

Todas as populações são inimigas umas das outras. Uma delas será, em princípio, controlada pelo jogador. Uma vez que as unidades têm um comportamento autónomo, em vez de um jogo, até se poderia ter um simples simulador, sem qualquer jogador humano (para ver o que acontece com o decorrer do tempo).

Assumindo que há de facto um jogador, este irá controlar uma das populações. O jogo consistirá então em comandar as unidades dessa população e ordenar a construção de novas unidades, edifícios, e de forma geral, definir a estratégia de evolução e conquista.

No início do jogo são colocadas duas ou mais populações no mapa. A cada população é dado um conjunto inicial de edifícios, unidades e recursos. O local “origem” da população é representado pelo seu castelo, que em princípio será um dos edifícios iniciais da população. Esse local pode ser sorteado, ou pode ser uma característica inicial do mapa (cenário inicial do jogo). Mais adiante são descritos os pormenores das unidades e edifícios.

Cada unidade pode desempenhar um conjunto de acções específico (ex., o camponês pode ir apanhar madeira à floresta e fazer novos edifícios, mas o soldado já não fará nenhuma dessas acções). Cada edifício tem também um determinado conjunto de propriedades e acções (ex., uma quinta permite a existência de uma determinada quantidade de unidades; um quartel permite “fazer” soldados, o castelo pode “fazer” camponeses, etc.). A produção de novas unidades e edifícios possibilita a obtenção de novas capacidades, mas gasta recursos, e o jogador deve definir uma estratégia de construção que lhe permita obter rapidamente vantagens, mas tendo em atenção que os recursos são limitados. São possíveis diversas estratégias. Por exemplo, pode-se criar um número elevado de camponeses para obter rapidamente recursos para uma expansão explosiva da população, ou então pode-se apostar na criação de soldados, que pouco contribuem para a evolução da população, mas permitem uma política de agressão e conquista sobre as populações vizinhas (por exemplo, para eliminar rapidamente a vizinhança enquanto ainda têm poucas unidades).

Os recursos naturais encontram-se inicialmente dispostos no mapa (aleatoriamente ou então seguindo uma estratégia inteligente qualquer que faça mapas “interessantes”). A forma como esses recursos aparecem está relacionada com a natureza do recurso. O ouro encontra-se em minas, as pedras encontram-se em pedreiras, e a madeira encontra-se em florestas. Neste enunciado, as minas, pedreiras, florestas e outros locais onde existam recursos serão colectivamente referidos como *fontes de recursos*. Estes elementos (mina, pedreira e floresta) contêm inicialmente um número finito destes recursos, o qual deve ser mais ou menos aleatório e que torne o jogo interessante (nem de menos, nem demais). O número e posição inicial das minas, pedreiras e florestas é também mais ou menos aleatório (pode ser aleatório, mas também pode seguir um algoritmo qualquer que torne o jogo mais interessante).

O território do tal reino medieval onde a acção do jogo/simulador decorre tem um terreno bastante regular: basicamente, é tudo terreno “normal”, que pode ser atravessado pelas unidades, pode conter edifícios, fontes de recurso, etc. Conceptualmente o território do reino é um rectângulo organizado em quadriculado e cada edifício, unidade, e fontes de recursos ocupam uma determinada quadrícula.

O mapa é inicialmente desconhecido do jogador, à excepção das imediações das suas unidades e edifícios iniciais, tornando-se conhecido à medida que as unidades da população vão deambulando pelo mapa. Cada unidade tem um raio de visão que “descobre” o mapa à sua volta, ficando essa zona do mapa conhecida definitivamente. Aquilo que existe e se passa nas zonas do mapa que não foram descobertas não é apresentado ao jogador. Importante: as populações que não são controladas pelo jogador vêm sempre o mapa todo. Pode dizer-se que a questão da visibilidade só afecta o jogador humano.

Os elementos principais do jogo são então: o mapa, as populações, os edifícios, as unidades, os recursos, as fontes de recursos.

3. Elementos do jogo

Território. O território representa o espaço onde o jogo/simulador decorre. Está organizado como um quadriculado. Pode haver mais do que uma unidade na mesma quadrícula (posição do mapa), mas cada quadrícula só pode ter, no máximo, um edifício ou uma fonte de recursos (ou um, ou outro). Pode haver unidades em simultâneo com um edifício ou com uma fonte de recursos. Apenas não pode haver em simultâneo na mesma posição: mais do que um edifício, mais do que uma fonte de recursos, ou um edifício com uma fonte de recursos.

Uma vez que pode haver mais do que uma unidade na mesma quadrícula, e que o facto de haver um edifício ou uma fonte de recursos não impede que a unidade ocupe essa mesma posição, depreende-se que as unidades podem desloca-se livremente por qualquer quadrícula, não havendo obstáculos ao seu movimento que não os limites do mapa.

As quadrículas correspondentes às zonas do mapa já conhecidas são visíveis. A parte do território ainda não explorada é invisível. Ser “invisível” significa que aparecerá no ecrã de forma obscurecida (mais pormenores sobre isso serão especificados mais adiante). As dimensões do território (número de linhas e colunas do quadriculado) são definidas no início do jogo. Estas dimensões, depois de definidas, não são modificadas durante o decorrer do jogo. Não existe um limite máximo pré-definido para estas dimensões.

População. Uma população é, como o nome indica, um conjunto de unidades e edifícios. A cada população está associado um nome.

Edifício. Um edifício é uma construção que está localizada numa das quadrículas do território. Pertence a uma população. Existem vários tipos de edifícios, e cada um possibilita diferentes aspectos durante o jogo (por exemplo o castelo pode gerar camponeses e o quartel pode gerar soldados). Os edifícios têm um nome único formado por 3 caracteres: uma letra e dois dígitos. A letra identifica o tipo de edifício (por exemplo, Q para quartel e F para quinta – estes detalhes são dados numa tabela mais adiante). Os dígitos representam a ordem pela qual foram criados, sendo que cada tipo de edifício tem a sua própria numeração (segundo uma sequência que é partilhada por todas as populações). Por exemplo, os primeiros quartéis têm o nome Q01, Q02, as primeiras quintas chamar-se-ão F01, F02, independentemente da população a que pertencem. Os edifícios têm também um estado de conservação. Quando criados, cada edifício tem o valor máximo (que depende do tipo de edifício – pormenores fornecidos mais adiante). O valor inicial significa um estado de 100% bem conservado, o valor 1 corresponde a um estado de destruição quase total, e 0 significa que o edifício foi destruído, devendo ser removido (pode ser feito outro se houver recursos e mão de obra).

Unidade. As unidades são seres vivos pertencentes a uma população. Têm um nome único segundo uma sequência que é partilhada para todas as populações. O nome das unidades é formado por 3

caracteres: uma letra e dois dígitos. A letra identifica o tipo de unidade (por exemplo, C para camponês e S para soldado). Os dígitos representam a ordem pela qual foram criados, existindo uma numeração independente para cada tipo de unidade (mas partilhada por todas as populações). Por exemplo, os primeiros camponeses têm o nome C01, C02, os primeiros soldados chamar-se-ão S01, S02. As unidades são feitas por edifícios, sendo que cada tipo de unidade é feito num tipo específico de edifício. Existem vários tipos de unidades. As unidades têm comportamento. Por exemplo, o camponês constrói estruturas e recolhe recursos e o soldado ataca. As unidades podem mover-se em cada instante para quadriculas adjacentes.

Fontes de recursos. As fontes de recursos são: Minas (ouro), pedreiras (pedra) e floresta (madeira). São parecidos com edifícios, mas não pertencem a nenhuma população, podendo ser usados por quem calhar. Não podem ser construídos. Existem inicialmente no mapa em número e posições aleatórias. Inicialmente possuem uma determinada quantidade de ouro/pedra/madeira e de cada vez que se “mina” um desses recursos, uma unidade do recurso é removida da fonte e passa para a posse da unidade que está a minar. Quando o número de recursos restantes na fonte chega a zero, a fonte é automaticamente removida.

Recursos. Os recursos são necessários para construir edifícios ou fazer novas unidades. Podem ser obtidos por uma unidade que tenha a capacidade de “minar”, e são consumidos quando se usam para as construções ou unidades. Demoram tempo a minar e não são um recurso renovável. Convém gerir com cuidado o gasto desses recursos. Neste enunciado existem apenas os recursos: ouro, pedra, madeira. No início são atribuídas determinadas quantidades desses recursos a cada população (quanto? → *experimente e veja quais são as quantidades que tornam o jogo mais interessante*).

Acção e decorrer do tempo. A acção do jogo decorre da seguinte forma: o tempo é medido em *instantes*, sendo que se passa de um instante para o seguinte por ordem do utilizador (através de um comando). Os vários elementos do jogo (edifícios, unidades, etc.) encontram-se dispostos no mapa e a cada passagem de um *instante* efectuam uma acção, que depende do tipo de elemento que são e do que os rodeia. No final de todos os elementos agirem é apresentada no ecrã a nova situação do mapa (e coisas que nele existem).

4. Interacção e Interface com o utilizador

A interface com o utilizador encontra-se, naturalmente, restringida pelo funcionamento em modo consola. Assim, não vão existir (a não ser que o queira fazer) *bitmaps*, nem o movimento será feito a nível do *pixel*. Ao invés, vai-se utilizar a lógica do quadriculado inerente ao um ecrã em modo texto (uma posição no ecrã = um carácter = um quadrado), e o quadrado será a unidade de construção básica da interface.

Não existe uma relação directa *uma quadrícula do mapa – um caracter no ecrã*. Uma coisa é a representação em memória da informação do jogo, e outra coisa é aquilo que se vê no ecrã. O mapa pode corresponder a uma estrutura muito simples (uma simples matriz?) ou mais complicada (uma matriz tridimensional de ponteiros para vectores de ponteiros para objectos com quatro ponteiros cada um, por exemplo?), dependendo da estratégia seguida por cada um, não havendo (nem sequer seria útil que houvesse) a necessidade pensar que “*uma vez que vejo caracteres no ecrã, segue-se que o mapa é uma matriz de caracteres*”.

Independentemente da forma como é internamente construído, o mapa deve ser apresentado de uma forma que se consiga entender. Dificilmente poderá ser de outra forma que não esta: uma posição (quadrícula) no mapa corresponderá a um conjunto de caracteres no ecrã (por exemplo, 3 x 3 caracteres), de forma a que se consiga entender o que está nessa posição do mapa. É mais ou menos óbvio que não se conseguirá ter todo o mapa em simultâneo no ecrã. Vê-se apenas uma parte dele, tal como acontece nos jogos reais. O utilizador decidirá que parte do mapa deseja ver, quer explicitamente (indicando as coordenadas), quem implicitamente (seleccionando uma unidade ou edifício e o mapa aparecerá centrado em redor dessa unidade ou edifício).

Sendo o programa segundo o paradigma de consola, não se vai usar o rato, por isso toda a interacção com o programa será feita através de comandos (cadeias de caracteres que se escreve e que dizem ao programa o que se pretende fazer).

Assim, o ecrã do jogo tem três componentes:

- A área que contém informações acerca do jogo
- A área em que são digitados os comandos
- A área que contém a parte visível do mapa

4.1. Área que contém informações acerca do jogo

Nesta zona do ecrã são apresentadas informações acerca dos recursos (quantidade de ouro, madeira e pedra), unidades existentes, etc.. Se estiver uma posição de mapa seleccionada (há um comando para isso), então aparece aqui a lista e detalhes do conteúdo dessa posição. Se estiver uma unidade ou edifício seleccionado, então aparecem informações sobre essa unidade ou edifício.

4.2. Área em que são digitados os comandos

A interacção com o jogo é feita através de “comandos” cuja existência já foi referida, e que serão descritos mais adiante.

Nota: existem dois modos de interacção: o modo normal e o modo *scroll*. No modo normal, escrevem-se comandos e a acção do jogo decorre normalmente. No modo *scroll*, a acção do jogo fica suspensa (não acontece nada) e as teclas das setas fazem com que o programa apresente outras partes do mapa. Podemos visualizar este modo imaginando que existe uma janela que apresenta a parte visível do mapa (só se consegue apresentar uma pequena parte do mapa no ecrã). As teclas das setas fazem essa

janela deslizar, passando a apresentar outra parte do mapa. O deslizamento é de uma posição (quadrícula) de cada vez, na direcção da seta premida.

Inicialmente está-se no modo normal. O comando “scroll” fará mudar para o modo *scroll*. No modo *scroll*, se o utilizador premir a tecla ‘c’, o jogo comuta para o modo “comandos”.


A lista completa de comandos é dada mais adiante.

4.3. A área que contém a parte visível do território

O território é representado por uma grelha bidimensional, em que cada quadrícula dispõe de espaço para 3x3 caracteres (três linhas e três colunas). Caso o mapa seja demasiado grande para a área disponível do ecrã, deve ser possível fazer *scroll* através da utilização do teclado (teclas das setas) quando em modo *scroll*.

As quadrículas podem representar estruturas, unidades e outros elementos do território como floresta ou minas. A primeira linha da representação da quadrícula dispõe de três caracteres onde se pode indicar o tipo mais abrangente do elemento que a quadrícula representa. Por exemplo E ou Est para estrutura, U ou Uni para unidade, T ou Terr para elementos do terreno. As estruturas e unidades têm um identificador único com três caracteres que podem ocupar a segunda linha da representação da quadrícula. A terceira linha pode ser ocupada com o estado de saúde do elemento quando for adequado.

Para facilitar a visualização, todos os elementos do jogo que pertençam a uma determinada população devem ser mostrados com a mesma cor de fundo (cada população terá uma cor diferente).

A zona desconhecida do território é invisível. Na representação do território, as quadrículas correspondentes devem reflectir esta situação (por exemplo, aparecem representados com o carácter )

A parte do mapa que se apresenta deve ter uma dimensão de pelo menos 9x9 quadrículas de mapa (ou seja 27x27 caracteres no ecrã). Pode aumentar esta dimensão, se o desejar e puder (quanto mais, melhor), mas só pode diminuir por indicação dos professores.

Não é obrigatório mas fica mesmo bem (por exemplo, na nota), e é mais agradável de utilizar, se usar um esquema de cores de fundo que distinga as posições do mapa como acontece, por exemplo, nos tabuleiros de xadrez. A cor de fundo não deve dificultar a leitura do conteúdo nem confundir as cores das várias populações.

É possível que haja vários elementos na mesma posição do mapa. Por exemplo, várias unidades, ou então uma unidade e um edifício. Nesta situação, apenas se mostra um elemento. Deve-se dar precedência aos edifícios e fontes de recurso. Havendo várias unidades e nenhum edifício ou fonte de recurso, mostra-se uma das unidades. Se se seleccionar essa posição do mapa, a área de informações apresentará a lista com a identificação das unidades e edifício/fonte que existam nessa posição.

5. Edifícios

Os edifícios são:

| Nome | Exemplo ID | Faz / Permite | Estado de conservação inicial | Custo |
|------------------------|------------|---|-------------------------------|-----------------------------|
| Castelo Max: 1 | C01 | Fazer camponeses | 200 | Madeira: Pedra: Ouro: |
| Quartel Max: 1 | Q01 | Fazer soldados | 100 | Madeira: Pedra: Ouro: |
| Estábulo Max: 1 | E01 | Fazer cavaleiros | 120 | Madeira: Pedra: Ouro: |
| Quinta Max: sem lim | F01 | Cada quinta permite ter 4 unidades na população | 80 | Madeira: Pedra: Ouro: |

Se pretender modificar estes valores em nome de uma jogabilidade mais interessante (é muito fácil, é muito difícil, etc.), pode fazê-lo, mas tem que o referir e justificar no relatório.

Não existe limite de quintas. Cada uma permite mais 4 unidades na população. Se as quintas forem destruídas, as unidades não são removidas, mas só se poderão fazer novas unidades desde que não se ultrapasse o limite permitido pelas quintas existentes.

Os edifícios são essencialmente passivos: fazem apenas as ações que forem especificadas por comandos.

6. Unidades

As unidades são descritas por duas tabelas. Atributos gerais, e comportamento.

6.1. Atributos gerais.

| Nome | Exemplo ID | Faz / Permite | Saúde | Custo |
|---------------------|------------|---|-------|-------------------------------------|
| Camponês | P01 | Minar pedra, ouro e madeira Velocidade de movimento: 1 Capacidade de carga: 5 | 20 | Madeira: 10 Pedra: 0 Ouro: 5 |
| Soldado | S01 | Atacar e defender Velocidade de movimento: 2 Capacidade de carga: 0 | 40 | Madeira: 10 Pedra: 0 Ouro: 10 |
| Cavaleiro Max: 1 | K01 | Atacar e defender Velocidade de movimento: 2 Capacidade de carga: 0 | 60 | Madeira: 20 Pedra: 5 Ouro: 10 |
| Camponês a cavalo | B01 | Minar pedra, ouro e madeira Velocidade de movimento: 1 Capacidade de carga: 10 | 40 | Madeira: 20 Pedra: 5 Ouro: 5 |
| Mistério Max: ? | M01 | A definir pelo grupo/aluno. Grupos diferentes com unidades mistério iguais será muito estranho | ? | Madeira: ? Pedra: ? Ouro: ? |

Nota: as siglas vêm do nome que as unidades teriam em inglês, para não se confundir com as siglas dos edifícios.

Se pretender modificar estes valores em nome de uma jogabilidade mais interessante (é muito fácil, é muito difícil, etc.), pode fazê-lo, mas tem que o referir e justificar no relatório.

Todas as unidades têm raio de visão 2. Esta informação é relevante para ir “descobrimo” o mapa à medida que as unidades se vão deslocando.

6.2. Comportamento.

| Nome | Feito por | Pode fazer (comandos suportados) | Comportamento automático | Força ataque | Defesa |
|-------------------|-----------|---|--|--------------|--------|
| Camponês | Castelo | Movimentar Minar pedra, ouro e madeira Fazer edifícios, consertar | Se atacado foge em direcção ao castelo (não ataca) | 0 | 0 |
| Soldado | Quartel | Movimentar Atacar | Se atacado riposta | 5 | 20 |
| Cavaleiro | Estábulo | Movimentar Atacar Defender | Se atacado, riposta | 8 | 40 |
| Camponês a cavalo | Estábulo | Movimentar Minar pedra, ouro e madeira | Se atacado foge em direcção ao castelo (não ataca) | 0 | 0 |
| Mistério | ? | ? | ? | ? | ? |
| Max: ? | | | | | |

6.3. Acções das unidades

As acções das unidades são desencadeadas através de comandos. O significado das acções é (os comandos são descritos mais adiante):

- **Movimentar.** A) Identifica-se a posição da pretendida e a unidade desloca-se para lá em linha “recta” à sua velocidade. O comando só se esgota quanto a unidade atinge o destino. B) Identifica-se o nome de outra unidade / edifício / fonte de recursos e a unidade desloca-se para a sua posição (vai dar ao mesmo que A).
- **Consertar.** Identifica-se o edifício a consertar e a unidade desloca-se para a sua posição e quando lá chegar conserta-o. Depois da unidade atingir a posição do edifício, a acção de consertar realiza-se instantaneamente. O custo é uma unidade de ouro por cada 10 pontos de saúde recuperados.
- **Minar recursos.** Identifica-se o nome da fonte de recursos. A unidade desloca-se para a fonte de recursos, e quando lá chega inicia a acção de minar, obtendo 1 unidade do recurso por cada instante. Quando tiver esgotar a sua capacidade de carga, desloca-se ao castelo, onde

depositará os recursos (todos de uma vez, assim que lá chegar). De seguida voltará à fonte de recursos e repetirá o comportamento até receber nova ordem, ou esgotar os recursos.

- **Fazer edifícios.** Identifica-se o tipo de edifício, a posição desejada, e se existirem recursos suficientes e pelo menos um camponês, o edifício será feito.
- **Atacar.** Identifica-se o edifício ou unidade e a unidade atacante desloca-se até atingir uma posição adjacente. Quando isso acontecer, será desferido um golpe sobre o adversário por cada instante. O adversário ripostará (se for uma unidade com capacidade de ataque). O ataque é descrito com mais pormenor mais abaixo.
- **Deambular.** Desloca-se ao calhas. Se for atacado dirige-se o castelo.

Utilidade das acções

- **Movimentar.** É necessário a todas as unidades.
- **Consertar.** Serve para reparar edifícios que foram atacados mas não totalmente destruídos. É mais barato reparar um edifício que reconstruir um do zero.
- **Minar recursos.** Serve para recolher recursos. O comportamento automático e repetitivo liberta o utilizador de estar sempre a repetir comandos. Se for implementado de uma forma que apenas faça uma “viagem de recolha de recursos”, já não será mau de todo.
- **Fazer edifícios.** Os edifícios são necessários para poder construir outras unidades.
- **Atacar.** Necessário para eliminar as outras populações e ganhar o jogo.
- **Deambular.** Serve para descobrir o mapa e de uma forma geral, espiar o inimigo.

6.4. Ataque

O ataque tem o seguinte comportamento:

- O ataque ocorre quando o atacante e o atacado estão na mesma posição.
- Por cada instante é desferido um ataque e um contra-ataque pela unidade atacada. Se o atacado for um edifício, não há contra-ataque. Se a unidade atacada não tiver a capacidade de ataque, então também não há contra-ataque.
- Cada unidade tem uma força de ataque (se puder atacar – nem todas podem). O valor de defesa do atacado representa a percentagem da força de ataque (do atacante) que é anulada (foi defendida). O valor de ataque que sobrar após essa anulação é subtraído à saúde do atacado. Por exemplo, se um soldado (força de ataque 5) atacar um cavaleiro (defesa 40), este evita 40% do ataque, sofrendo uma diminuição de 3 pontos na sua saúde.
- O contra ataque é despoletado automaticamente se o atacado tiver a característica de ataque. Funciona como um ataque (ver pontos anteriores).
- Mesmo que o atacado morra, tem sempre direito ao contra-ataque (se for dos que atacam).
- O ataque e o contra ataque decorrem no mesmo instante no tempo do jogo.

As unidades que têm a capacidade de ataque sempre que vêm um inimigo, atacam-no (primeiro têm que se deslocar para a sua posição).

6.5. Movimento

O movimento entre duas posições é muito fácil de fazer. Basta ver quantas posições segundo o eixo das linhas e das colunas é que é preciso deslocar para ir da origem até ao destino, e em cada instante do tempo do jogo, a unidade desloca-se nessa direcção, de acordo com a sua velocidade (uma certa quantidade de posições em linha e coluna de cada vez, na direcção pretendida). Não há obstáculos que possam impedir o movimento da unidade, por isso o algoritmo é muito simples. Serão dados esclarecimentos adicionais sobre este ponto nas aulas se for caso para isso.

7. Configuração

Todo o controlo do jogo é feito por comandos. Isto inclui tanto a criação inicial do jogo (definição de mapa, de populações, edifícios etc.), como o seu controlo (ordens a unidades, por exemplo). Um comando é uma linha (uma cadeia de caracteres) que se digita e é lida pelo jogo, o qual a processa/interpreta e em função do que se escreveu faz algo (esta parte da aplicação é de implementação directa, sem nada de especialmente complexo). Deve ser possível obter os comandos tanto a partir do teclado (situação habitual e intuitiva), como a partir de um ficheiro. Este último cenário funciona da seguinte forma: existirá um ficheiro de texto, e um comando **load**. O ficheiro tem os comandos, um em cada linha, sendo cada linha tal e qual o que se escreveria no teclado. Ao escrever o comando **load nome_do_ficheiro**, o jogo irá ler o ficheiro e processar cada uma das suas linhas, executando os seus comandos tal como se viessem do teclado. A parte do programa que interpreta e executa o comando é a mesma, quer o comando venha do teclado, quer o comando venha de um ficheiro:

Teclado → String → processaComando(...) → executaComando(...)

Ficheiro → String → processaComando(...) → executaComando(...)

O comando **load** pode ser utilizado a qualquer altura: no início do jogo, para configurar um cenário/mapa, ou no decorrer do jogo para executar um qualquer conjunto de comandos gravados e simplificar a tarefa de controlo do jogador. Pode haver muitos ficheiros diferentes já preparados, sendo apenas necessário indicar o nome daquele que se pretende “executar”. Não há nada contra “executar” o mesmo ficheiro repetidamente.

8. Comandos

A interface com o utilizador não segue a lógica de menus: cada comando é uma linha de texto, formada pelo nome do comando e 0 ou mais parâmetros, separados por espaços. Os parâmetros são apresentados aqui sob a forma <significado>, o que significa que o jogador deverá digitar letras e números cujo significado é *significado*, mas sem digitar os ‘<’ e ‘>’ (obviamente).

Os comandos a suportar pela simulação são:

Comandos relativos à criação/finalização do jogo e às populações:

- **mkgame <linhas> <coluna>** Faz um mapa vazio, todo invisível, com as dimensões indicadas em **linhas** e **colunas**.

- **pop <nome>** Cria a população **<nome>** sem edifícios nem unidades. A primeira população a ser acrescentada ao jogo é controlada pelo jogador.
- **setf <oque> <linha> <coluna>** Coloca uma fonte de recursos na posição **linha** e **coluna**. A posição de terreno alvo não pode conter outras fontes de recursos nem edifícios, e a fonte de recursos é a indicado em **oque**: *min* → mina, *ped* → pedreira, *flo* → floresta. Recordar que as fontes de recurso não pertencem a população nenhuma: são de todos e de ninguém.
- **sete <nome> <oque> <linha> <coluna>** Coloca um edifício novo na posição **linha** e **coluna**, e pertencendo à população indicada em **nome**. Não gasta recursos e não precisa de haver camponeses. Esta funcionalidade é para testes e configurações. A posição de terreno alvo não pode conter outro edifício ou fonte de recursos e o edifício é o indicado em **oque**: *cas* → castelo, *quar* → quartel, *est* → estábulo, *quin* → quinta. Quando a população **nome** é controlada pelo utilizador, esta colocação vai afectar a “descoberta” das posições que envolvem o edifício. Uma vez que cada edifício permite “descobrir” as posições à volta, o jogador passa a ter mais umas posições “descobertas” (se ainda não estivesse descobertas).
- **setu <nome> <oque> <linha> <coluna>** Acrescenta uma unidade na posição **linha** e **coluna**, pertencendo à população indicada em **nome**. Não gasta recursos e não precisa de haver edifícios. Esta funcionalidade é para testes e configurações. A unidade acrescentada é indicada em **oque**: *camp* → camponês, *sold* → soldado, *cav* → cavaleiro, *cv* → camponês a cavalo. Quando a população **nome** é controlada pelo utilizador, as posições que envolvem a nova unidade tornam-se visíveis. A criação da unidade torna as posições em redor do mapa “descobertas”, de forma semelhante à colocação de um edifício. Recordar que a “descoberta” de mapa só afecta o jogador - o computador vê o mapa todo.
- **load <filename>** Lê comandos do ficheiro designado por **filename**, um por linha, e executa-os. Este comando pode ser utilizado para configurar o estado inicial do jogo, mas também para executar um conjunto gravado de comandos quando o jogo já está a decorrer. Cada linha do ficheiro tem um comando com a mesma estrutura dos comandos que se digitam através do teclado, e portanto sujeita à mesma interpretação, e utilizando as mesmas partes do programa que fazem essa interpretação e processamento.
- **mostra <nome>** Mostra, na zona de informação do ecrã, uma listagem de todos os elementos da população identificada pelo **nome**.
- **sair** Termina a execução do programa.

Comandos relativos à interacção com o jogo:

- **sel <id>**. Selecciona o edifício ou unidade com nome **id**. Os atributos do elemento seleccionado são apresentados na área do ecrã que contém informações acerca do jogo. Se a parte visível (no ecrã) do mapa não contiver a unidade ou edifício seleccionado, então a parte visível deverá ser modificada de forma a que a unidade ou edifício seleccionado passe a estar representado no centro da área (ecrã) que contém a parte do mapa (de forma mais simples: *a janela que define a parte visível do mapa desloca-se e fica centrada na unidade ou edifício seleccionado*).
- **scroll** muda para o modo “scroll”, tal como descrito acima.

- **next** Avança um instante na simulação. Isto implica calcular os efeitos da passagem de tempo para todos os intervenientes na simulação, incluindo efectuar o comportamento automático para todos os elementos (das duas facções) que não tenham recebido ordens explícitas.
- **next <n>** Simula a passagem de **<n>** instantes de tempo sem precisar da interacção do utilizador entre cada um. No final de cada instante é apresentado a nova situação (mapa), podendo ser feita uma brevíssima pausa.
- **Tecla “enter”** (a linha que contém o comando, lida com *getline()*, tem 0 caracteres) Faz o mesmo que o comando **next**.

Comandos para as unidades e edifícios:

- **go <id> <delta_linha> <delta_coluna>**. Ordena à unidade com identificador **id** que se mova para a quadrícula do território na linha que corresponde à linha onde estava + **delta_linha** e na coluna que corresponde à coluna onde estava + **delta_coluna** (ou seja, é dado o deslocamento a fazer, não a posição destino). O movimento pode envolver vários passos, cada um para uma quadrícula adjacente. Em cada instante, a unidade dá o número de passos correspondente à sua velocidade. O movimento pedido pode precisar de vários instantes. Se **id** for '.', então a unidade será aquela que tiver sido seleccionada antes.
- **goto <id> <linha> <coluna>**. Semelhante em tudo ao comando **go**, mas aqui é dado a linha e coluna do destino, em vez do deslocamento.
- **mina <id> <idfonte>** Ordena à unidade **id** que vá obter recursos na fonte **idfonte**. Se **id** for '.', então, a unidade será aquela que tiver sido seleccionada antes. Se a unidade identificada não tiver a capacidade de minar, então a ordem é ignorada.
- **ataca <id> <idvitima>** Ordena à unidade **id** que vá atacar a unidade ou edifício **idvitima**. Se **id** for '.', então, a unidade será aquela que tiver sido seleccionada antes. Se a unidade identificada não tiver a capacidade de atacar, então a ordem é ignorada.
- **conserta <id> <ide>** Ordena à unidade com identificador **id** que conserte o edifício com identificador **ide**. Se **id** for '.', então a unidade será aquela que tiver sido seleccionada antes. Se a unidade identificada não tiver a capacidade de consertar, então a ordem é ignorada.
- **deambula <id>** Ordena à unidade com identificador **id** que deambule. Se **id** for '.', então a unidade será aquela que tiver sido seleccionada antes. Se a unidade identificada não tiver a capacidade de deambular, então a ordem é ignorada.~
- **mke <id> <oque> <linha> <coluna>** Ordena à unidade **id** que faça um novo edifício na posição **linha** e **coluna**, de acordo com **oque**: *cas* → castelo, *quar* → quartel, *est* → estábulo, *quin* → quinta. Se **id** for '.', então, a unidade será aquela que tiver sido seleccionada antes. Se a unidade não tiver a capacidade de construir ou se não existirem recursos, a ordem é ignorada. O camponês desloca-se para a posição pretendida e só quando lá chega é que faz o edifício. A posição de terreno alvo tem que estar vazia de edifícios e fontes de recurso.
- **mku <oque>** Manda fazer uma nova unidade, de acordo com **oque**: *camp* → camponês, *sold* → soldado, *cav* → cavaleiro, *cv* → camponês a cavalo. Se não existir o edifício ou recursos necessário, o comando é ignorado.

Comandos para os recursos:

- **ouro** <nome> <n> A população **nome** recebe **n** unidades de ouro¹.
- **pedra** <nome> <n> A população **nome** recebe **n** unidades de pedra².
- **madeira** <nome> <n> A população **nome** recebe **n** unidades de madeira³.

Nota: Considere que poderiam existir comandos para copiar, atribuir e apagar populações. Não é necessário fazer esses comandos, mas o código das classes envolvidas deve ser compatível com essas operações.

9. Controlo de populações e unidades pelo computador

Não se pretende ter neste trabalho estratégias muito elaboradas acerca do tema “inteligência artificial”. Pretende-se apenas que as unidades das populações do computador façam qualquer coisa para o jogo não ficar demasiado aborrecido. As acções tomadas podem não ser as melhores, mas deve acontecer alguma coisa.

A forma mais fácil de implementar o controlo do computador sem entrar em matéria de Inteligência Artificial, será esta: as unidades e edifícios já têm a capacidade para obedecer a ordens (por exemplo, obedecem aos comandos escritos pelo jogador). Então, vai-se simular a existência de jogadores virtuais que enviam comandos às unidades e edifícios das suas populações, da seguinte forma:

- Sempre que o jogador ordena ao jogo para passar ao instante seguinte (comandos **next**, **next** <n> ou simplesmente **enter**), o jogo irá dar ordens às suas populações (e depois, volta a dar o controlo ao jogador).
- Para cada população pertencendo ao computador (=“para cada jogador virtual”), o jogo irá dar ordens a algumas das suas unidades e edifícios. Isto pode ser feito de diversas formas. Por exemplo, mas sem ter que ser assim (é um exemplo), cada unidade / edifício pode ter um valor N específico a ela, e de de N em N instantes, ser-lhe-á dada uma ordem.
- A ordem poder será sorteada. Qual a ordem, quanto é N, quais as probabilidades de cada ordem, tudo isso deixa-se ao critério dos alunos. Pode haver inclusivamente um conjunto de ordens fixas ou previamente preparadas, e simplesmente escolhe-se uma delas ao acaso. Se se escolher uma ordem que não é possível, os mecanismos da unidade ignoram a ordem, tal como acontece para o jogador humano. Também pode acontecer que haja um conjunto predefinido de ordens para cada tipo de unidade e edifício.
- O mecanismo de execução da ordem é exactamente o mesmo que para o jogador humano. Simplesmente, a ordem, em vez de ter vindo do teclado ou de um ficheiro, foi sortada/predefinida/outro-mecanismo.
- A estratégia aqui indicada é um exemplo. Há mais formas de fazer isto, melhores e sem grande trabalho. Não se pretende que o computador jogue de forma inteligente. Apenas que faça

¹ Pode ser uma quantidade negativa (para testes).

² Pode ser uma quantidade negativa (para testes).

³ Idem.

alguma coisa. Deve-se experimentar outras alternativas e ver como é que se consegue tornar o jogo mais interessante.

Nota: Tem uma ideia melhor para esta parte? É muito bem vindo. Se conseguir implementar um comportamento interessante e que dê um bocadinho de luta, contará como extra para compensar outros aspectos que não estejam tão bem.

10. Requisitos Específicos

Não existe nenhum limite predeterminado quanto ao número de elementos. Pode existir um ou um milhão de edifícios ou unidades. Idem quanto ao tamanho do mapa. Estas características são limitadas apenas pela quantidade de memória do computador.

11. Faseamento do trabalho

11.1. Primeira fase

- **Data de entrega: 7 de Dezembro de 2014**
- **Material a entregar:** Projecto em Visual Studio, incluindo todo e qualquer ficheiro auxiliar que seja necessário para a sua execução imediata. Nesta fase o relatório ainda não é necessário

Funcionalidade requerida

- Na primeira fase são consideradas as populações, unidades e edifícios. Nesta meta, considera-se que há apenas Soldados e Castelos. Os edifícios não têm ainda nenhuma funcionalidade. As unidades movem-se apenas por ordem do utilizador e não fazem nada de especial.
- Deve ser possível ler e executar o ficheiro de configuração.
- Deve ser possível criar um jogo com duas populações.
- Apresentar a interface com o utilizador com as características que foram descritas. Os soldados e castelos já aparecem correctamente.
- Deverá ser implementado o *scroll* e a introdução dos comandos descritos. Os comandos devem ser validados (ver se os parâmetros fazem sentido), mas apenas os de criação de mapa, de movimento e de criação de unidades e edifícios (soldados e castelos) devem ser implementados.
- As unidades e os edifícios devem ser representados por objectos em memória dinâmica (válido para ambas as metas).

11.2. Segunda fase

- **Data de entrega: 4 de Janeiro de 2014**
- **Material a entregar:** Projecto em Visual Studio, incluindo todo e qualquer ficheiro auxiliar que seja necessário para a sua execução imediata. O relatório é entregue nesta fase.

Funcionalidade requerida: tudo.