UNIVERSIDADE ABERTA
www.uab.pt

Ministério da Ciência, Tecnologia e Ensino Superior

Resolução / Resumo de correção do exame

U.C. 21025

Desenvolvimento de Software

12 de julho de 2017

1.ª Parte (4 Valores)

1.

a) Linhas do código de referência onde há operações de *input*: `01, 20`

b)Linhas do código de referência onde há operações de *output*: `11, 18, 19`

2. **a)** Segundo a abordagem de Krasner & Pope (1988): `M:02, C:09, V:11, M:12, V:18, C:20, M:23, M:25`

b) Segundo a abordagem de Curry & Grace (2008): `M:02, C:09, V:11, M:12, V:18, V:20, M:23, M:25`

c) Dúvidas ou dilemas com que se debateu para responder às alíneas a) e b) e justificações: Espera-se dos alunos a identificação de aspetos do código cuja classificação como M, V ou C possa ser alvo de argumentação lógica a favor de mais do que uma destas componentes, apresentando esses dilemas e fundamentando as opções que tomaram. Apresentam-se aqui alguns exemplos de aspetos que poderiam mencionar como oferecendo dúvidas:

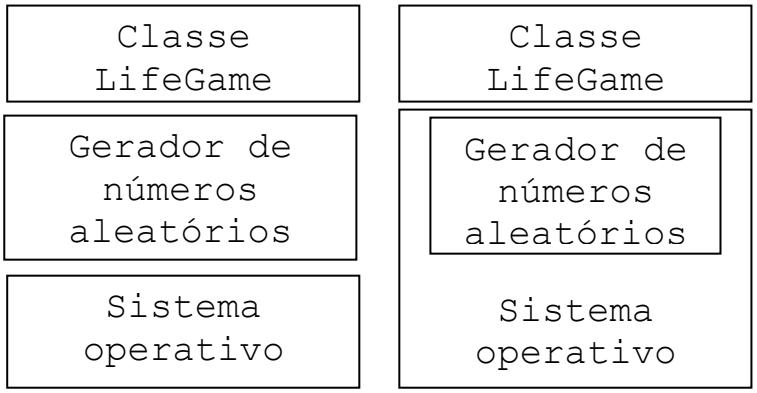
- existência de variáveis usadas para controlo do fluxo, como "exit" e "first", na secção "Valores iniciais"; a minha opção é por integrá-las no Model, pois este pode contar o estado do programa, sendo o controller apenas responsável por reagir a ele; mas poderia ser interpretado por um aluno que estas variáveis deveriam constar do Controller, atribuindo por isso esta secção a ele (justificação que leva a ser considerada correta essa resposta nas alíneas anteriores);
- considerar que embora a linha 18 seja claramente View, por mostrar o estado do jogo, a linha 19, que faz output mas apenas para instruir o processo de input, possa ser feita no controller na abordagem de Krasner & Pope, em particular em sistemas onde seja necessário processar esta instrução e a leitura do input no mesmo módulo (justificação que leva a ser considerada correta essa resposta nas alíneas a).

3. O código anterior depende de vários componentes não explicitados: (1) o sistema operativo, que fornece argumentos de linha de comandos, processa os pedidos de escrita no ecrã e leitura do

teclado; (2) a classe LifeGame, que fornece a lógica do jogo da vida; (3) o gerador de número aleatórios do sistema, a que apoia a classe LifeGame.

a) Desenhe um diagrama onde o código anterior e estes três componentes surjam num padrão de desenho do tipo "Arquitetura estratificada".

Consideram-se como sendo corretas respostas que mostrem e hierarquia indicada no enunciado e as que optem por incluir o gerador dentro do sistema. Exemplos à direita.



b) Um dos objetivos da etapa de desenvolvimento dedicada à produção de código-fonte é *desenvolver os aspetos que não tenham sido cobertos pela fase de análise e conceção do sistema de software* (Guerreiro, 2015, p. 155). Considere que a fase de análise e conceção produziu, como únicos resultados, a lista de *inputs* e *outputs* da pergunta 1 e a arquitetura estratificada da alínea 3 a). Que aspetos do código-fonte fornecido não foram cobertos? Se a análise e especificação só produziu a lista de inputs e outputs, não cobriu o desenho arquitetural, aspeto central da resposta. Pode ser complementado ou substituído por exemplos como sejam: a forma como os componentes se relacionam ou "comunicação entre intervenientes", os tipos de requisitos não contidos na mera lista de inputs e outputs, etc._____

4. Considere o código de referência sob a perspetiva da verificação e validação do software. Dê alguns exemplos de falhas, erros e faltas que possam ocorrer nesse código.
O aspeto essencial da resposta é dar exemplos que demonstrem a compreensão do significado concreto destes três conceitos. São aceitáveis exemplos que sejam baseados em conceitos gerais de programação, ainda que o comportamento exato na plataforma .Net possa ser diferente. Por exemplo, pode não haver memória para criar o objeto LifeGame na linha 3. Considera-se correto indicar como "falta" tanto a ausência de código para verificação do resultado do new como a não inclusão do new dentro de um bloco try. Considera-se correto indicar como "erro" tanto o facto de lifeGame ter ficado a null como indicar que foi gerada uma exceção de falta de memória. E considera-se correto tanto indicar como "falha" que o programa termina abruptamente ou 'crasha' por não ser apanhada a exceção de falta de memória como pela tentativa de aceder a um método de uma referência null na linha 05. Outros exemplos poderiam ser usados.

2.ª Parte (8 Valores)

5. Tomando como base a sua resposta à pergunta 2, opte por uma das variantes do estilo arquitetónico MVC. Reescreva o código no estilo arquitetónico MVC, considerando o seguinte:
   a) Indique a variante escolhida por si para responder a esta pergunta e desenhe o diagrama MVC correspondente, **substituindo os rótulos genéricos que encontra nos materiais de referência com os aspetos específicos deste caso**.
   Usar os diagramas de referência da última página do enunciado e alterar o texto. Por exemplo, se a opção for pelo estilo de Krasner & Pope, solucionado em 2a) como: "M:02, C:09, V:11, M:12, V:18, V:19, C:20, M:23, M:25" então a bola "M" deveria

ter os números 02, 12, 23 e 25, e as V e C os restantes. Os rótulos genéricos da comunicação e controlo também deviam ser substituídos. Há várias soluções admissíveis, sendo o essencial que se respeite a ligação entre módulos. Por exemplo, para "Mensagens de visualização" (ligação Controller-View) admitir-se-ia "Limpar o ecrã" (pois há uma transição controller->view no código). Para ligação controller->model, podia-se indicar a reação ao "if (key == 'r'", que sendo processamento de input é feito no controller: indicando "Reiniciar o jogo da vida" em "Mensagens de edição e acesso ao modelo". A alteração de exit, que depois é consultada em "while (!exit)" (controller) podia ser usada para "Mensagens de alterações de dependências" entre o Model e o Controller. E assim sucessivamente.

b) Defina as classes que pertencem aos componentes Model, View e Controller. Não utilize eventos, delegados, interfaces, nem exceções, apenas os atributos e métodos habituais.

Usar a divisão da resposta 2a). Há várias respostas válidas. Por exemplo, na classe Model poder-se-ia ter o atributo exit, bem como os métodos para o controller solicitar a sua alteração e consultar o valor atual. A View pode ter um método "Atualizar erã" ou métodos separados para isso e para apresentação do menu de input.

c) Reescreva em conformidade o método Main.

O método depende da resposta à alínea anterior, mas o essencial é que demonstre compreensão do funcionamento. Por exemplo, Main(){ Model.NewGame(); While(!Model.Exit()){…

3.ª Parte (8 Valores)

6. Tomando como base a sua resposta à pergunta 5, reescreva o código para obter independência de componentes e separação de interesses, da seguinte forma:

a) Defina e utilize eventos e delegados para reportar alterações nos módulos.

b) Defina e utilize interfaces para que a comunicação de dados ocorra com menos dependências entre módulos.

c) Defina exceções e escreva o código que as lança em caso de erro. Escreva igualmente o código que as apanha, respeitando as responsabilidades de cada módulo no MVC.

A resposta depende da pergunta anterior e tem muitas soluções possível. O essencial é demonstrar domínio da aplicação das três técnicas.

a) Por exemplo, em vez do Controller mandar atualizar o Model e depois mandar atualizar a View, pode ser definido um evento "NovaGeração" no Model e o controller associar a esse evento, como delegado, o método da View, "AtualizarEcrã".

b) Em vez de passar um objeto LifeGame à View que depois lhe faz apenas "ToString", mas nada a impede de chamar também métodos como "Step", poder-se-ia definir uma interface iLifeGame que permite apenas usar o ToString. O controller (ou o evento do Model) passaria à View o objeto lifeGame mas ocultado como sendo apenas um objeto de uma classe que implementa iLifeGame.

c) Qualquer aspeto que possa eventualmente gerar um erro pode ser escolhido. Por exemplo, pode-se assumir que o "ToString()" do iLifeGame pode gerar um erro, fazer-se um try à volta dele e no catch da exceção lançar-se nova exceção, personalizada (por ex. ExcecaoVistaSemDados). No controller ter-se-ia encapsulado a chamada à View dentro de um try com um catch para a eventualidade desta exceção personalizada e reagia-se, por exemplo, mudando o funcionamento para nova interação com o utilizador: "Não foi possível mostrar o jogo, quer tentar com outro?"