CES-28 Exame – 2017
Dicksiano C. Melo

QUESTÕES

1)

a) Como um drone, eu gostaria de conhecer a Localização e a Velocidades (em coordenada gloais) dos Drones mais próximos de mim, assim eu conseguiria utilizar a função avoid-colision apenas quando fosse necessário.

b) Como GCS, eu gostaria de enviar comandos de velocidades nos 3 eixos (x,y,z) para o Drone que controlo, assim eu conseguiria manusear a trojetória do Drone de forma mais ágil e segura.

c) Como UTM-CTR, eu gostaria de receber uma confirmação de que os GCSs da minha área de atuação estão recebendo a informação de forma atualizada, assim eu poderia ativar um protocolo de falhas se algum GCS não confirmasse o recebimento .

2)

A)

Independente: o item elaborado não depende de outros fatores, basta conhecer localização/velocidades dos drones mais próximos.

Negociável: os desenvolvedores poderiam negociar para que, por exemplo, em vez de coordenadas globais, fossem utilizadas coordenadas locais. Essa user story não está fechada para mudanças.

Valiosa: as informações envolvidas são bem eficazes: **desejo** comandar velocidade **com o objetivo** de utilizar a função avoid-colision. O user story é simple e direto e não coloca informações desnecessárias como, por exemplo, qual critério seria utilizado para chamar a função avoid-colidance, pois os desenvolvedores não precisam dessa informação, eles só precisam saber que é necessário que o drone conheça localização/velocidade dos outros drone mais próximos.

Estimável: a tarefa está bem clara: **saber localização/velocidade dos drones mais próximos**. Assim, é facil para os desenvolvedores estimarem o tempo que irão gastar.

Pequeno: esta US é bem simples e curta, sendo direta de ser desenvolvida em um sprint.

Testável: fácil de testar, bastaria verificar se o drone alvo conhece a localização/velocidade corretas de alguns drones próximos dele.

B)

Independente: o item elaborado não depende de outros fatores, basta que GCS seja capaz de comandar velocidades, é bastante desacoplado do resto do sistema.

Negociável: os desenvolvedores poderiam negociar para que, por exemplo, em vez de coordenadas x,y,z fosse x,z e giro em torno do próprio eixo. Ainda assim o objetivo seria cumprido.

Valiosa: as informações envolvidas são bem eficazes: **desejo** comandar velocidades em cada eixo **com o objetivo** de guiar o drone eficientemente. O user story é simple e direto e não coloca informações desnecessárias como, por exemplo, velocidades mínimas ou máximas e etc. O user story foca no objetivo único de comandar velocidades.

Estimável: a tarefa está bem clara: **comandar velocidades nos eixos**. Assim, é facil para os desenvolvedores estimarem o tempo que irão gastar.

Pequeno: esta US é bem simples e curta, sendo direta de ser desenvolvida em um sprint.

Testável: fácil de testar, bastaria medir se o drone possui as velocidades que são comandadas pelo GCS.

C)

Independente: o item elaborado não depende de outros fatores, basta que UTM seja capaz de receber uma confirmação, é bastante desacoplado do resto do sistema.

Negociável: os desenvolvedores poderiam negociar para que, por exemplo, em vez de coordenadas sempre receber uma confirmação, o UTM recebe uma confirmação apenas a cada 10 mensagens, por exemplo.

Valiosa: as informações envolvidas são bem eficazes: **desejo** receber confirmação **com o objetivo** de chamar um protocolo de falhas quando necessário. O user story é simplea e direto e não coloca informações desnecessárias.

Estimável: a tarefa está bem clara: **receber confirmação de que CGS rebebe mensagens**. Assim, é facil para os desenvolvedores estimarem o tempo que irão gastar.

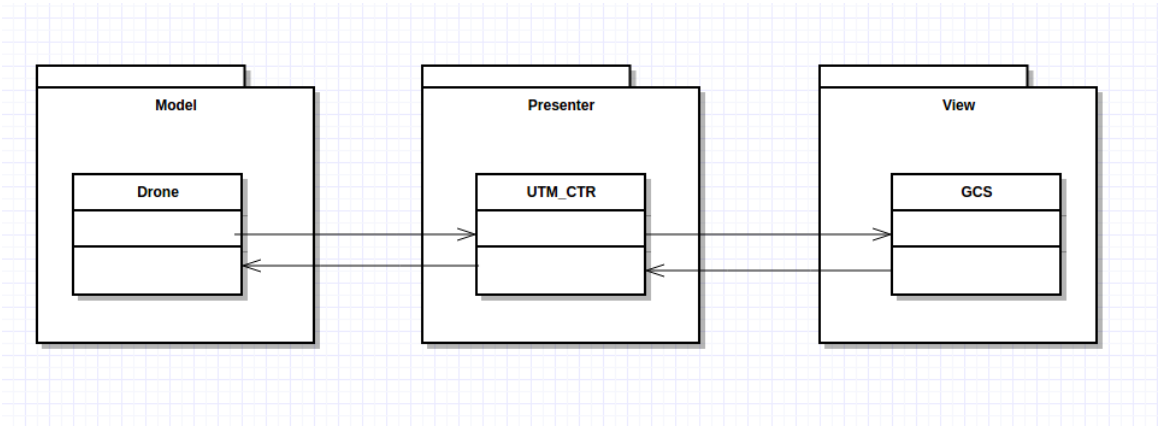
Pequeno: esta US é bem simples e curta, sendo direta de ser desenvolvida em um sprint.

Testável: fácil de testar, bastaria checar se as mensagens de confirmação estão sendo enviadas sem atraso.

3) O padrão **Singleton**. Esse padrão deve ser utilizado, em geral, quando a classe precisa ter uma única instânica e ela deve ser acessada de forma global.

   Nesse problema, não faria sentido existir mais de uma instância de UTM-CTR, pois ela é um ponto de comunicação único, com o qual todos os GCSs irão se comunicar (isso fica explícito pelo desenho). Além disso, ela possui acesso global, pois cada GCS consegue comunicar-se com a mesma.

4)



5) Veja o código em anexo: exame/utm_v0

Observe que a classe UTM_CTR (detro do package Presenter) é um **Singleton**. Isso resolve o problema 5.c

A classe Drone (package Model) guarda o seu estado que é a posição (x,y,z). Sempre que ele muda de estado avisa pro UTM_CTR (presenter) que por sua vez avisa essa mudança para todos os GCS (View). Inicialmente, apenas o GCS que possuia o Drone sabia dessa mudança. Porém, eu percebi que todos os GCS devem saber a posição dos Drones (ou seja, os GCSs observam os Drones).

A classe GCS requisita uma mudança na posição do Drone. Como o enunciado foi bem explicito e não existe o canal entre Drone e GCS, sempre que GCS requisita uma mudança na posição, ele envia isso para UTM (presenter) que passa isso para os Drones. Quando o drone muda sua posição (estado) ele avisa pra UTM que se encarrega de avisar GCS.

Perceba que segui a arquitetura MVP: Drone conhece apenas UTM, GCS conhece apenas UTM e UTM conhece os outros dois.

A Main imprime no Console as informações sobre as posições dos GCS. Estas posições estão corretas. Observe que cada Drone inicia em (0,0,0) e cada um dos 3 GCSs comanda mudanças de posição em cada eixo.

A classe GCS guarda a posicçaõ do seu Drone e o conjunto de posições dos outros Drones (mas ele não conhece a Classe Drone, assim guarda apenas a posição).

6) Veja dentro do projeto em: exame/src/utm_v1

Os testes fazem o mesmo que estava feito na Main, só que agora utilizando teste de unidade. Cada teste possui um Javadoc explicando sua funcionalidade.