DOG
doing online games

DOG Framework Cookbook

No contexto da abordagem de frameworks orientados a objetos, um cookbook (livro de receitas) é um manual de utilização de um framework, composto por um conjunto de "receitas", que estabelecem os passos a serem seguidos para produzir uma aplicação ou parte de uma aplicação, a partir do framework. O presente documento contém o cookbook de DOG Framework.

Download do framework e procedimentos preliminares

1. Baixar e descompactar o arquivo do framework (ver página do framework)
2. Seguir as instruções do arquivo 'readme'

Com relação a 2, especial atenção à criação do arquivo de identificação do jogo sob desenvolvimento, sem o que não será possível conectar-se ao servidor.

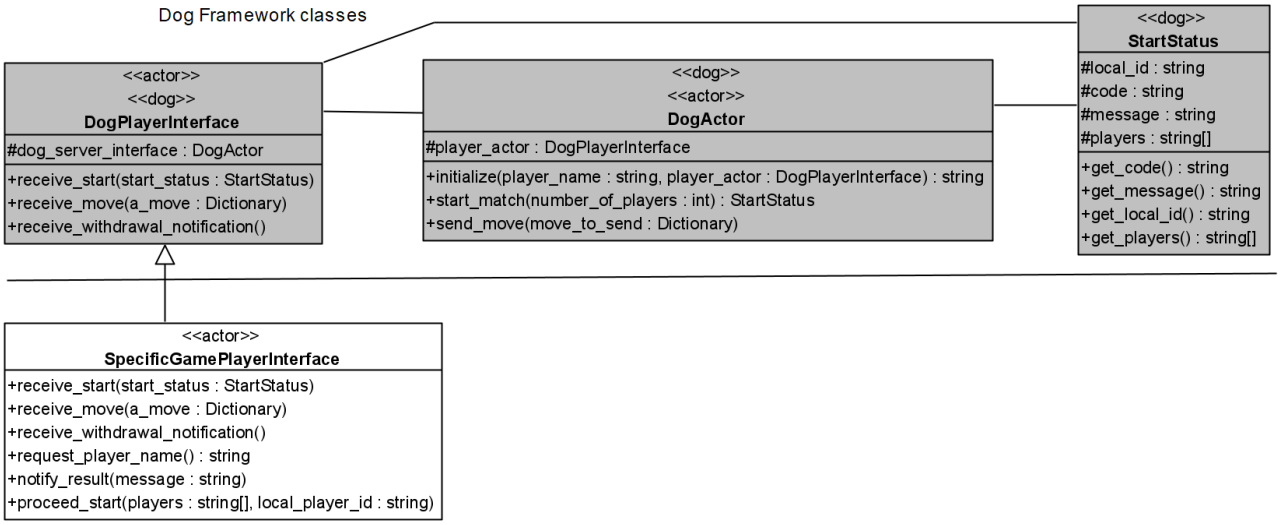
Caso precise aprender a usar ambiente virtual (virtual environment), use o roteiro a seguir:

https://www.inf.ufsc.br/~ricardo.silva/dog/Python_com_virtual_environment.pdf

Classes reusadas e criadas

Um dos objetivos a alcançar durante o desenvolvimento de um framework orientado a objetos é torná-lo simples sob o ponto de vista do seu usuário – mesmo que ele não seja um artefato de software de baixa complexidade. Para isso, minimiza-se a quantidade de classes do framework que o usuário precisa conhecer para conseguir desenvolver software sob o framework.

No caso de DOG Framework, um usuário precisa conhecer e manusear três de suas classes, mostradas na imagem a seguir, a saber, DogActor, DogPlayerInterface e StartStatus.



DogActor – é a fachada da estrutura interna do framework e, em última análise, a interface do jogo em desenvolvimento com o servidor (DOG Server). É uma classe que o desenvolvedor precisará instanciar (tratado a seguir, no contexto do caso de uso correspondente) e que invocará da instância criada os seus três métodos presentes no diagrama de classes acima (também tratados na descrição dos casos de uso correspondentes).

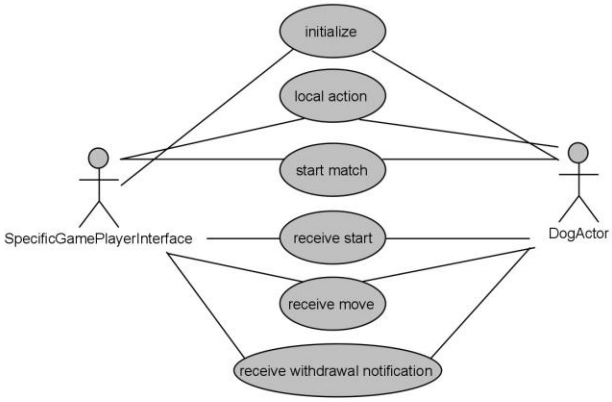
DogPlayerInterface – representa a interface gráfica do usuário (GUI) da aplicação em desenvolvimento. Cada jogo terá sua interface específica. Assim, ao criar a classe correspondente ao ator associado ao usuário da aplicação (jogador), o desenvolvedor deverá observar que ela deverá ser subclasse de DogPlayerInterface e deverá sobrescrever os três métodos previstos nesta classe, presentes no diagrama de classes acima.

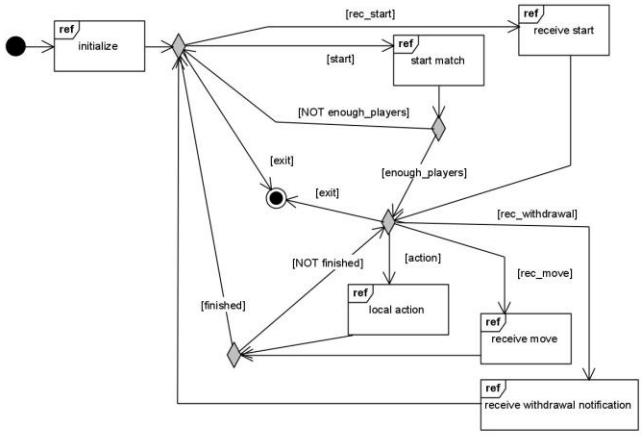
StartStatus – classe voltada a conter as informações necessárias ao início de uma partida. O usuário receberá uma instância dessa classe, com essas informações em seus atributos. Mais detalhes na descrição dos casos de uso correspondentes.

SpecificGamePlayerInterface - identificação genérica da subclasse de DogPlayerInterface (não é uma classe do framework, mas da aplicação sob desenvolvimento). Ao criar a subclasse de DogPlayerInterface em seu jogo, o desenvolvedor dará a ela a denominação que julgar adequada.

Cenários do framework

Os diagramas de casos de uso e de visão geral de interação a seguir ilustram os possíveis cenários de atuação de DOG Framework.



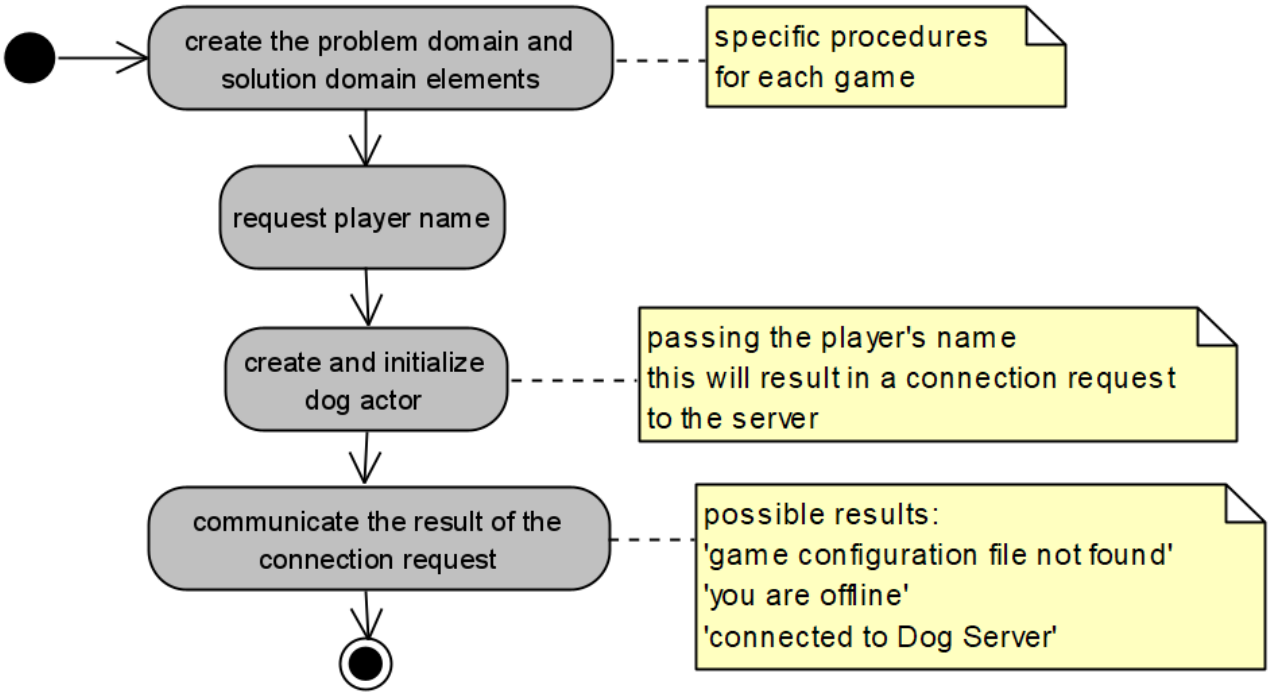


Caso de uso	Iniciativa	Situação
initialize	local	Conexão ao servidor
start match	local	Início de partida
receive start	remota	Início de partida
local action	local	Partida em andamento
receive move	remota	Partida em andamento
receive withdrawal notification	remota	Partida em andamento

Caso de uso 'initialize'

Deve ser previsto no projeto do jogo um caso de uso "initialize" ou denominação equivalente, responsável pelos procedimentos iniciais da execução da aplicação, a saber, a instanciação dos elementos da interface gráfica, de outros eventuais aspectos do domínio da solução computacional, assim como elementos do domínio do problema.

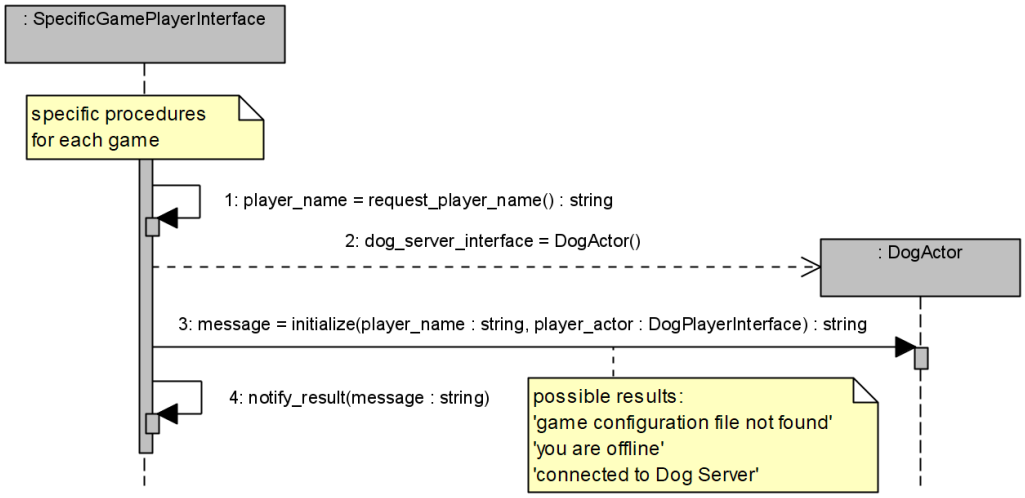
Neste caso de uso, além das questões específicas do jogo, deve ser feito o acréscimo referente a DOG framework, descrito no diagrama de atividades a seguir.



O primeiro procedimento específico de DOG Framework é a necessidade de obter do usuário o seu nome (por meio de uma janela de diálogo, por exemplo). Observe-se que isso já poderia estar previsto no desenvolvimento, independente do uso ou não do framework.

O segundo procedimento é a instanciação de DogActor e a iniciação da instância criada (detalhada no diagrama de sequência a seguir). O resultado desse procedimento é o retorno de uma mensagem (string) a ser apresentada ao usuário, com o resultado da tentativa de conexão ao servidor. Possíveis resultados (mensagens):

- 'Arquivo de configuração do jogo não encontrado'
- 'Você está sem conexão' (offline)
- 'Conectado a Dog Server'

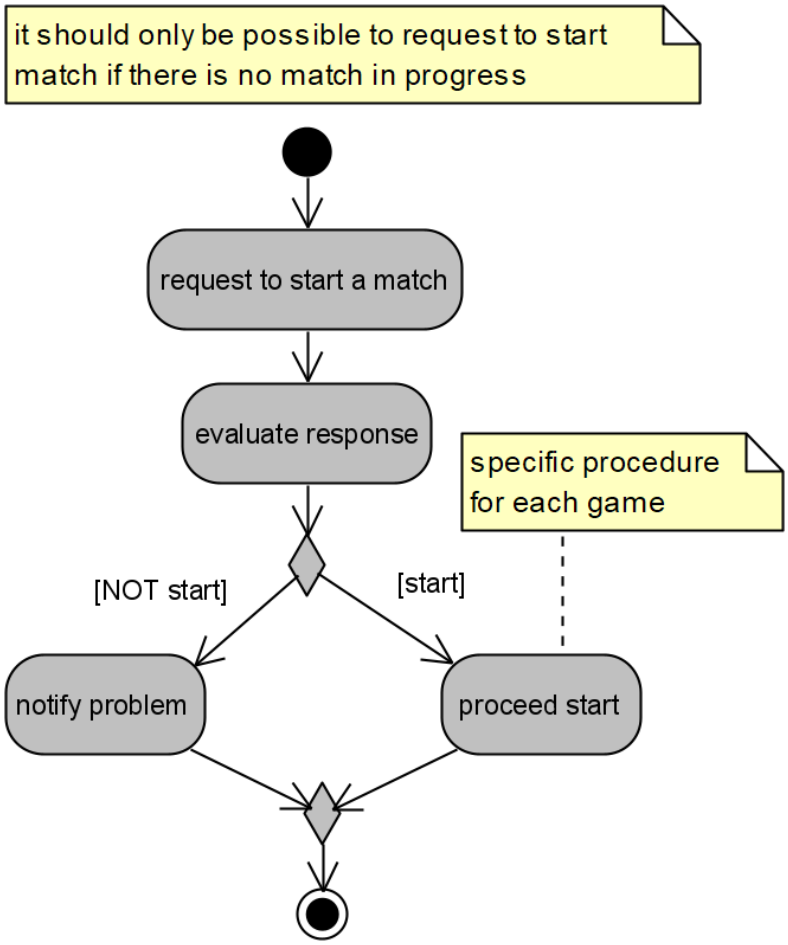


O diagrama de sequência acima detalha o procedimento descrito:

- A primeira mensagem que aparece no diagrama é precedida pelos procedimentos específicos de cada jogo, sendo o tratamento de DOG a última parte da rotina de iniciação da aplicação;
- A mensagem com número 1 corresponde à obtenção do nome do jogador;
- A mensagem 2 é a invocação do construtor de DogActor;
- A mensagem 3 é a invocação do método initialize de DogActor, que prevê 2 parâmetros, o nome do jogador (string) e a referência da subclasse de DogPlayerInterface (o objeto que invoca passa sua própria referência – self – como parâmetro);
- A mensagem 4 é a notificação ao usuário do resultado da tentativa de conexão ao servidor, com o resultado obtido da tentativa de conexão, isto é, o retorno do método initialize().

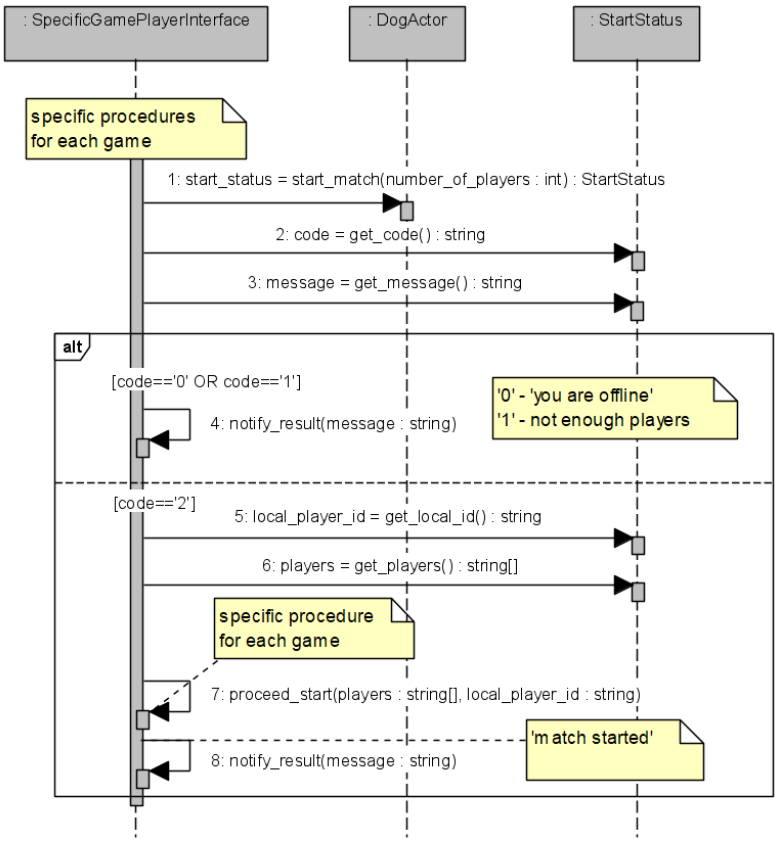
Caso de uso 'start match'

O caso de uso 'start match' é o responsável pelo procedimento de início de uma partida. O primeiro passo, como pode ser observado no diagrama de atividades a seguir, é a solicitação de início por parte do usuário, cuja resposta definirá se é possível iniciar uma partida ou não. Caso seja possível, segue-se o procedimento específico de cada jogo.

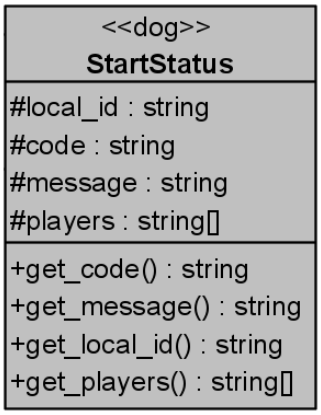


O diagrama de sequência a seguir detalha o caso de uso, abaixo comentado.

- O caso de uso é disparado por uma ação do usuário em sua interface (aqui representada pela classe 'SpecificGamePlayerInterface') e pode prever procedimentos preliminares antes da solicitação de início propriamente dita;
- A solicitação de início é feita pela instância da subclasse de 'DogPlayerInterface' (aqui, da classe 'SpecificGamePlayerInterface') à instância de DogActor que ela referencia por meio de atributo, invocando o método start_match(), que prevê como parâmetro a quantidade de jogadores – mensagem identificada com número 1. Observe-se que há jogos com quantidade fixa de jogadores e jogos com quantidade variável e sempre haverá a necessidade de informar essa quantidade, pois o servidor não conhece a lógica dos jogos que trata e, consequentemente, a quantidade de jogadores de uma partida;



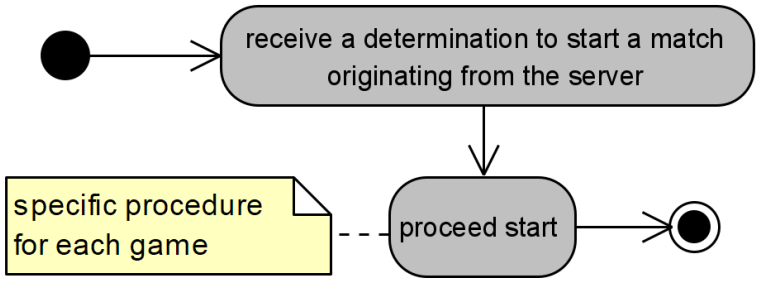
- Como resultado da invocação, será retornada uma instância da classe StartStatus, abaixo apresentada.



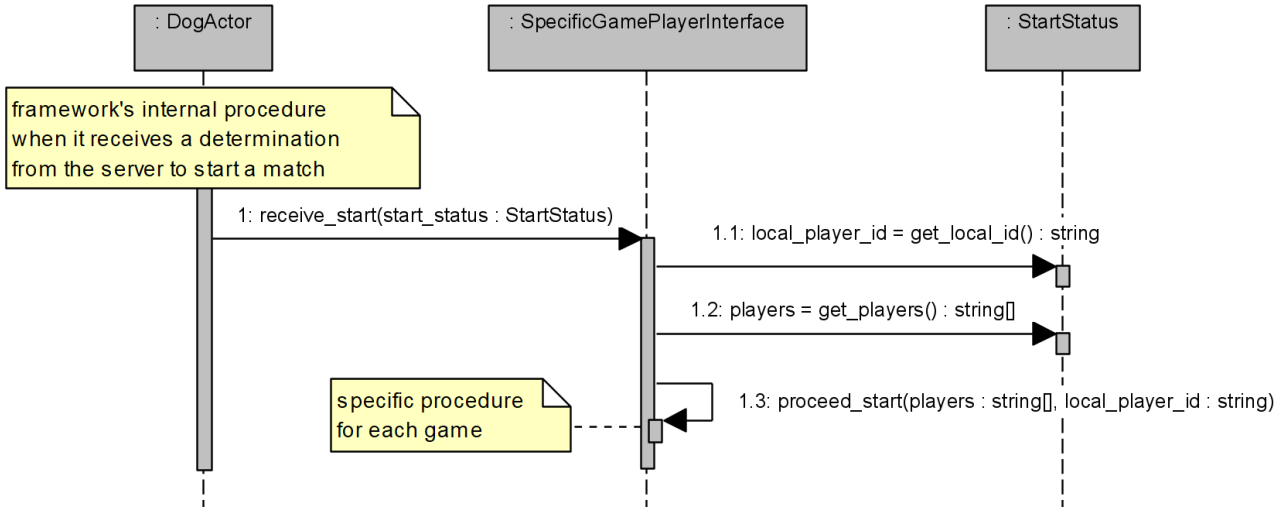
  - O atributo 'local_id' é o id único do jogador local. O jogador local é identificado pelo nome informado pelo usuário e pelo id único gerado automaticamente por DOG Framework;
  - O atributo 'code' contém o resultado da solicitação de início e pode ser um dos seguintes valores:
    - '0' - 'Você está offline';
    - '1' - 'Jogadores insuficientes';
    - '2' - 'Partida iniciada'
  - O atributo 'mensagem' pode conter um dos três textos acima;
  - O atributo 'players' é uma lista ou vazia ou contendo listas, sendo cada uma correspondente a um dos jogadores (neste caso, apenas quando partida iniciada). O conteúdo de cada lista de jogador será *[nome do jogador, id único do jogador, ordem do jogador na partida]*, sendo todos os valores do tipo string. A primeira lista sempre será correspondente ao jogador local (*observar a provável necessidade de converter a ordem do jogador de string para inteiro*).
- A partir da avaliação do resultado da solicitação de início de partida (valor de 'code'), ou o usuário será informado da impossibilidade (parte superior do fragmento combinado do diagrama de sequência acima), ou será executado o início de uma partida, procedimento específico para cada jogo (mensagem 7) e, em seguida, notificado o usuário da partida iniciada (mensagem 8) – parte inferior do fragmento combinado.

Caso de uso 'receive start'

O caso de uso 'receive start' é disparado por DOG Framework (e não pelo usuário) e também é responsável pelo procedimento de início de uma partida. O disparo de 'receive start' decorre de uma ação remota: algum usuário remoto executou o caso de uso 'start match' e o servidor selecionou o usuário local para participar da partida e envia a ele uma determinação de início. A partir disso, procedimento específico de início de partida deve ser executado, de forma similar ao descrito para o caso de uso 'start match'. O diagrama de atividades a seguir descreve esse procedimento



O diagrama de sequência a seguir detalha o caso de uso.

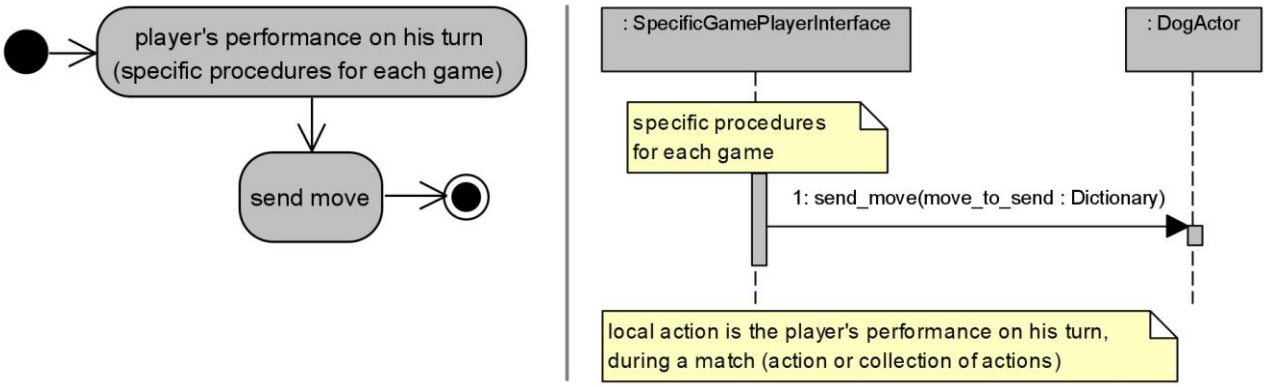


- A primeira mensagem do diagrama (mensagem 1) corresponde à invocação do método receive_start(), que deve ser sobrescrito na subclasse de DogPlayerInterface criada pelo desenvolvedor do jogo (classe 'SpecificGamePlayerInterface', no diagrama acima). O diagrama omite os procedimentos anteriores executados por DOG Framework, para verificação da solicitação de início originada em DOG Server. O método receive_start() tem como parâmetro uma instância da classe StartStatus, anteriormente descrita, sendo que neste caso o valor do atributo code é '2', isto é, necessariamente a comunicação de uma partida iniciada, não sendo necessário testar a possibilidade de outros valores;
- Com isso, é disparado o procedimento de início de partida, similar ao já descrito para o caso de uso 'start match'.

Caso de uso 'local action'

O caso de uso 'local action' representa a ação do jogador em uma partida durante seu turno, isto é, o seu lance, sua jogada. Esse procedimento é específico de cada jogo: tanto a ação do jogador em si quanto a quantidade de ações. Num jogo da velha, por exemplo, um jogador apenas define a posição a ocupar (apenas uma ação); num jogo de xadrez, por outro lado, precisa definir as posições de origem e destino de sua peça (2 ações). Há jogos com quantidades de ações variáveis (Banco Imobiliário, por exemplo).

O que os diagramas abaixo descrevem é que após executar sua ação local, especifica do jogo tratado, a jogada deve ser enviada aos demais jogadores (de fato, enviada à instância de DogActor), por meio da invocação do método send_move().



Algumas questões precisam ser detalhadas em relação ao envio de jogada por meio do método send_move():

- No caso de jogos em que o usuário executa mais de uma ação em seu turno, o desenvolvedor pode escolher se ocorrerá o envio de cada ação ou do conjunto de ações de uma só vez. Por exemplo, no caso de um jogo de xadrez, enviar duas vezes (uma para a seleção da posição de origem e outra para a seleção da posição destino) ou apenas uma vez (posições origem e destino contidas em um mesmo envio);
- O conteúdo de um lance a enviar será um dicionário (Python dictionary), com o conteúdo que o desenvolvedor quiser. Por exemplo, para o jogo de xadrez poderia ser:
  - {'linha': <valor>, 'coluna': <valor>} ou
  - {'linha_origem': <valor>, 'coluna_origem': <valor>, 'linha_destino': <valor>, 'coluna_destino': <valor>}, conforme a opção de envio;
- Além das informações específicas do seu jogo, o desenvolvedor deve incluir no dicionário a chave **'match_status'**, com um dos seguintes valores:
  - 'next' – sinalizando que o jogador está passando o turno com o envio da jogada;
  - 'progress' – sinalizando que o jogador ainda não está passando o turno com o envio da jogada (e vai fazer pelo menos mais um novo envio);
  - 'finished' – sinalizando que, com a jogada efetuada, a partida está encerrada (esta informação é necessária para informar o final da partida ao servidor).

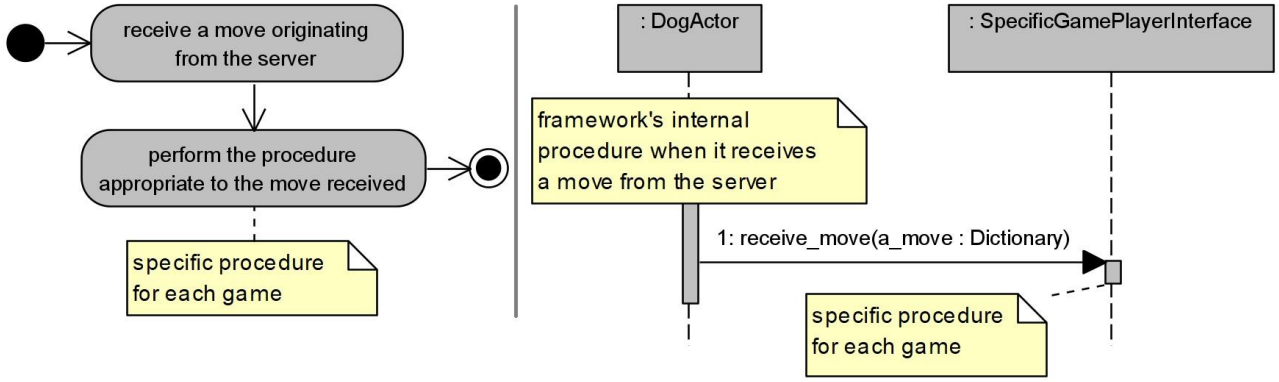
Além do envio de jogada durante uma partida, o método send_move() também pode ser usado para o envio de informações necessárias ao início de uma partida. Suponha um jogo que envolva dois ou mais jogadores, em que de um mesmo baralho devam ser distribuídas cartas a todos os jogadores. Como fazer isso? Como DOG Server não trata a lógica dos jogos, é necessário que a distribuição de cartas seja feita por um e apenas um dos jogadores (um dos clientes). Suponha que seja escolhido para a tarefa o jogador que inicia a partida (ordem: '1'). Esse jogador instanciará o baralho, definirá aleatoriamente a 'mão' de cada jogador e, com isso, o conjunto de cartas restante no baralho. A forma de fazer essa informação chegar aos demais jogadores é por meio do método send_move() que não estará enviando uma jogada propriamente dita neste caso, mas dados essenciais para o início das ações dos jogadores.

Observar que se, ao invés do procedimento acima descrito cada jogador instanciasse um baralho e definisse apenas sua mão, cada jogador teria um baralho diferente, ao invés de todos usarem o mesmo baralho.

Caso de uso 'receive move'

O caso de uso 'receive move' é a contrapartida do envio de jogada descrito no caso de uso anterior. Quando um jogador envia uma jogada – método send_move() – o(s) adversário(s) devem receber a jogada enviada e isso é tratado no caso de uso 'receive move'. Ele é disparado por iniciativa de DOG Framework, como tratamento de uma jogada enviada a DOG Server.

Os diagramas abaixo descrevem o recebimento da jogada, seguido pelo seu tratamento, o que é específico para cada jogo

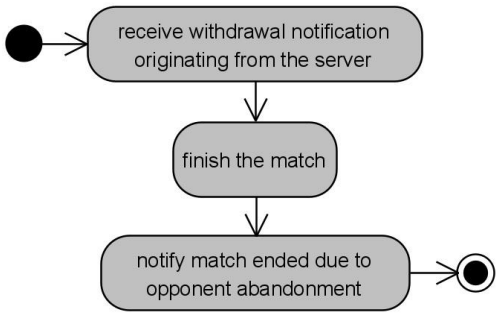


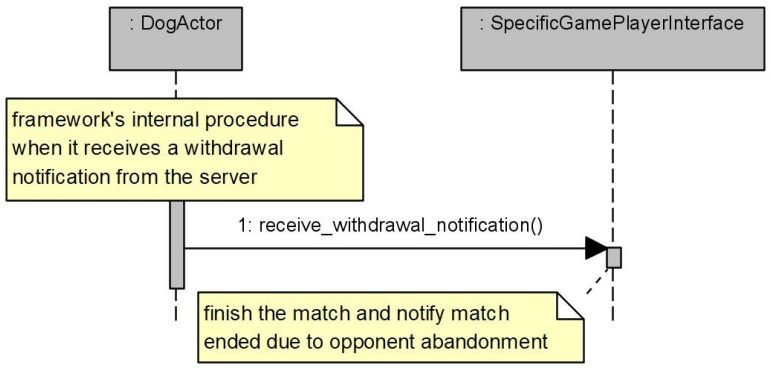
O método receive_move(), invocado pela instância de DogActor, tem como parâmetro o dicionário que contém a jogada, descrito no caso de uso anterior. Sobre o dicionário recebido, devem ser feitas as seguintes ressalvas:

- Além das chaves já descritas, o dicionário conterá a chave **'player'** cujo valor corresponderá ao id único do jogador autor da jogada (lembrando que no início da partida é recebida a relação de nomes e identificadores de todos os jogadores);
- Todos os valores recebidos no dicionário são do tipo string. Assim, se um valor foi enviado como numérico, será recebido como string e terá que ser convertido para numérico.

Caso de uso 'receive withdrawal notification'

Caso de uso disparado por iniciativa de DOG Framework, como tratamento de um abandono de partida percebido por DOG Server. Quando um dos jogadores de uma partida abandona a partida fechando o programa, DOG Server percebe e sinaliza isso aos demais jogadores, indicando o encerramento da partida por desistência de um dos jogadores. Os diagramas abaixo detalham o caso de uso.





Essencialmente, a instância de DogActor invoca o método receive_withdrawal_notification() e, como resposta, a aplicação deve levar a partida ao estado encerrada e deve notificar o usuário que a partida foi encerrada devido ao abandono de um dos jogadores (não é informado qual jogador abandonou a partida).