Trabalho DSO1

Intrigas Digital : Apresentação

Objetivo do Sistema

Intrigas é um card game do grupo G2E, da UFSC. O sistema procura implementá-lo para que seja possível jogar no computador. Ele pode ser usado para divulgação do produto físico, ou como um novo produto digital.

UML Visão Geral

Está sendo usado o padrão MVC. Tem poucas entidades, e não são complexas, como Baralho, Carta, Jogador, Bot. Os controladores são a parte mais complexa. A ControlGame armazena e tem alguns métodos referentes ao estado do jogo.

-A Fluxo é dedicada a tratar o Input recebido, mas também a situações que norteiam o fluxo de execução.

-A ControlJogador também chama métodos de Fluxo, mas é principalmente para implementar atitudes que seriam de um jogador ou um bot.

-A Main contém objetos de várias classes, conectando elas. A maioria das suas funcionalidades são estáticas.

UML Visão Geral

-A ControlAcaoPersonagem em geral contém o poder de cada carta, e só é chamada depois de uma série de constatações (se houve dúvida, se houve carta de defesa).

-ControlAcaoComb e EnumAcaoComb seriam para as ações combinadas entre as cartas, mas não foram implementadas.

-Os limites são as telas, a TelaGame não é trivial. Como foi implementada, ela tem vários métodos que renderizam partes do seu conteúdo. Algumas considerações são feitas também no próprio ActionListener do botão Duvidar e Próximo

Principais Métodos

O principal, sem dúvida é o Fluxo.passaVez(). Ele descobre o index do jogador da vez, então descobre o próximo jogador da vez e atualiza a variável ControlGame.jogadorDaVez. Se ela for uma instância de Bot, ali começa a execução de uma ação do Bot. Se não, a execução deve continuar a partir do input na TelaGame. Ela também verifica se o jogo acabou, para abrir a TelaEnd, e pula uma linha no console da TelaGame.

Temos também o Fluxo.duvidar() que analisa se o ControlGame.jogadorDeQuemSeDuvida tem as cartas que declarou ter. Então ele decide se ele perde uma carta, ou se quem perde é o ControlGame.jogadorDuvidando

Principais Métodos

fluxo.btnAcaoPersonagem() é chamado quando alguém declara ter um personagem. Ele armazena a ControlGame.ultimoTipoAcao, e ControlGame.ultimoPersUsado, e também desvia a execução apropriadamente para os casos em que é o solicitante é o jogador humano, ou um bot, e considerando se a carta é de ataque, ou uma carta de defesa.

fluxo.veSeTemDuvidaEChamaAcaoPers acaba fazendo várias considerações acerca de se é ou não blefe, e que carta está sendo usada para então chamar os métodos de ControlAcaoPers ou não

Os Action Listeners dos botões Próximo e Duvidar também são importantes para o fluxo de execução

Principais Desafios

No começo eu não tinha ideia de como implementar o fluxo das ações. Não tinha idéia de como descrever os comportamentos de que, em sua jogada, um jogador poderia decidir fazer uma entre várias opções, e ainda, uma vez que ele escolhesse, não estaria perto do fim, ainda haveria uma série de decisões a tomar antes do próximo turno.

Ainda, antes de ele decidir, é necessário saber: ele é um Bot, ou é o jogador humano, o jogador com index zero?

Principais Desafios

Digamos que seja a vez de um jogador: se ele for bot, são chamados os métodos Bot.decideAcao(). Se ele decidir por uma ação de personagem, chama Bot.decidePersonagem(). Digamos que ele decidiu usar o poder da personagem “Nineta”. Isso significa que ele precisa decidir um alvo, o que não seria necessário se tivesse optado pelo personagem “Kane”. Bom, já que ele é um bot, não vai ser iniciada uma rodada de dúvidas: vamos esperar para ver se o Botão Duvidar é pressionado, ou o próximo. Mas, veja bem, se o alvo escolhido for o jogador humano, ele pode ainda escolher usar o personagem “Pistone”, que se defende da “Nineta”. Ah, ok, mas esperem: ele pode ter o “Pistone” ou não. E os bots vão decidir entre duvidar dele ou não. Dependendo de tudo isso, aí seria chamado o método ControlAcaoPers.acaoPistone() ou acaoNineta(). E a cereja do bolo: o “Pistone” pode ser usado como defesa, mas ele também pode ser usado como ataque (ele vê uma carta de outro jogador).

Principais Desafios

Então, sem dúvida o maior desafio foi fazer esse fluxo de ações acontecer de uma maneira coerente para todos os casos. E a maneira que foi feita acabou cheia de ifs e elses, pulando de um método para outro. Para corrigir os vários bugs era necessário ir anotando por onde a execução já tinha passado e ir percorrendo vários métodos. Se eu fosse implementar de novo, tentaria encontrar um grande modelo do qual a maioria das opções fizesse parte. Outra coisa seria tentar organizar a execução mais em um lugar só, com os métodos indo e voltando pro mesmo lugar, não indo pra lá, e indo de lá pra não sei onde, de não sei onde pra cá, de cá pra não sei onde, de não sei onde pra lá, etc, etc.