Instituto de Ensino Superior

DOCUMENTAÇÃO

Game - Maxwell

Trabalho Final - Estrutura de Dados

Jogo de simulação de logística

EQUIPE:

- Ângela Riedel
- Ianna Melissa Osório de Sousa
- Lígia Karolinne
- Maria Eduarda Araújo

DISCIPLINA: ESTRUTURA DE DADOS

Professor: Dimmy Karson Soares Magalhães

INTRODUÇÃO

No jogo, o jogador atua como um viajante que precisa se deslocar do ponto A ao ponto B e encontrará muitas cidades ao longo do caminho. Em cada cidade, podem ocorrer eventos que irão ajudar e atrapalhar o jogador. Uma cidade pode aumentar o poder contido na gema, e se esse poder ultrapassar seu limite, o jogo termina. Por fim, os jogadores serão recompensados com base no número de moedas acumuladas no caminho até o destino.

DESCRIÇÃO DO CÓDIGO

classe PERSONAGEM

A classe `Personagem` é uma classe que representa o personagem Maxwell no jogo.

Essa classe possui:

5 propriedades privadas:

1. moedas: int representa o número de moedas que o personagem possui.
2. poderJoia: int que representa o poder da jóia do personagem.
3. limiarJoia: int que representa o limiar da jóia do personagem.
4. estaVivo: boolean que indica se o personagem está vivo ou não.
5. cidadeAtual: string que armazena o nome da cidade atual do personagem.

10 métodos públicos para acessar e modificar as propriedades do personagem:

1. getCidadeAtual(): retorna a cidade atual do personagem.
2. setCidadeAtual(String cidadeAtual): define a cidade atual do personagem.
3. getMoedas(): retorna o número de moedas do personagem.
4. setMoedas(int moedas): define o número de moedas do personagem.
5. getPoderJoia(): retorna o poder da jóia do personagem.
6. setPoderJoia(int poderJoia): define o poder da jóia do personagem. Se o poder for menor que 0, ele é ajustado automaticamente para 0.
7. getLimiarJoia(): retorna o limiar da jóia do personagem.
8. setLimiarJoia(int limiarJoia): define o limiar da jóia do personagem.

9. isEstaVivo(): indica se o personagem está vivo por boolean.
10. setEstaVivo(boolean estaVivo): define se o personagem está vivo ou não.

O construtor da classe `Personagem` é responsável por inicializar as propriedades do personagem com valores padrão, contidos em documento.

- `moedas` é inicializado com 3.
- `poderJoia` é inicializado com 0.
- `limiarJoia` é inicializado com 7.
- `estaVivo` é inicializado como verdadeiro.
- `cidadeAtual` é inicializado com o valor "Ubud".

O método `condicoes()` é um método vazio que representa as condições do personagem.

O método `checaStatusPersonagem()` retorna um valor booleano indicando se o personagem está vivo ou não.

Os métodos `getMoedas()`, `getPoderJoia()`, `getLimiarJoia()` e `isEstaVivo()` retornam os valores das propriedades correspondentes.

Os métodos `setMoedas(int moedas)`, `setPoderJoia(int poderJoia)`, `setLimiarJoia(int limiarJoia)` e `setEstaVivo(boolean estaVivo)` permitem modificar as propriedades correspondentes.

classe DIÁLOGO MERCADOR

A classe `DialogoMercador` representa um diálogo entre o Maxwell e o mercador.

A classe `DialogoMercador` possui:

5 propriedades públicas:

1. `moedasMaxwell`: um inteiro que parece representar o número máximo de moedas que Maxwell pode ter. No entanto, essa propriedade não é usada no código fornecido.
2. `moedas`: um inteiro que representa o número de moedas disponíveis no diálogo.

3. `distancia`: um inteiro que representa a distância entre Maxwell e o mercador.

4. `moedasAtual`: um inteiro que parece representar o número de moedas atualizado após o diálogo.

5. `querTrocar`: um valor booleano que indica se Maxwell quer realizar uma troca com o mercador.

   O construtor da classe `DialogoMercador` recebe um objeto `Personagem` chamado `maxwell`, um inteiro `moedas`, um inteiro `distancia` e um valor booleano `querTrocar`. No construtor, as propriedades `moedas`, `distancia` e `querTrocar` são inicializadas com os valores fornecidos e a propriedade `moedasAtual` é inicializada com 0.

   O método `atualizarBens` recebe um objeto `Personagem` chamado `maxwell`. Esse método contém uma série de condições que representam diferentes situações no diálogo entre Maxwell e o mercador, com base no número de moedas e na distância.

   - Se `moedas` for menor que 5:

     - Se `distancia` for menor que 3:

       - Se `querTrocar` for verdadeiro:

       - Reduz uma moeda de `maxwell` usando o método `setMoedas`.

       - Aumenta o limiar da joia de `maxwell` em 1 usando o método `setLimiarJoia`.

   - Se `querTrocar` for falso:

       - Reduz uma moeda de `maxwell` usando o método `setMoedas`.

     - Se `distancia` for maior ou igual a 3:

       - Se `querTrocar` for verdadeiro:

       - Reduz uma moeda de `maxwell` usando o método `setMoedas`.

       - Aumenta o limiar da joia de `maxwell` em 2 usando o método `setLimiarJoia`.

   - Se `querTrocar` for falso:

       - Adiciona 2 moedas a `maxwell` usando o método `setMoedas`.

   - Se `moedas` for maior ou igual a 5:

     - Se `distancia` for menor que 3:

       - Se `querTrocar` for verdadeiro:

- Reduz 3 moedas de `maxwell` usando o método `setMoedas`.

- Aumenta o limiar da joia de `maxwell` em 2 usando o método `setLimiarJoia`.

- Se `querTrocar` for falso:

- Reduz 2 moedas de `maxwell` usando o método `setMoedas`.

- Se `distancia` for maior ou igual a 3:

- Se `querTrocar` for verdadeiro:

- Reduz uma moeda de `maxwell` usando o método `setMoedas`.

- Aumenta o limiar da joia de `maxwell` em 3 usando o método `setLimiarJoia`.

- Se `querTrocar` for falso:

- Reduz 3 moedas de `maxwell` usando o método `setMoedas`.

As propriedades de `maxwell` são atualizadas com base nas ações tomadas durante o diálogo.

classe MISSÃO KINGDOMKALB

A classe `MissaoKingdomKalb` representa uma missão específica no jogo.

A classe declara várias variáveis de instância, como `nome`, `descricao`, `statusConcluida` e outras. Essas variáveis são usadas para armazenar informações sobre a missão. Por exemplo, `nome` armazena o nome da missão, `descricao` armazena uma descrição da missão e `statusConcluida` indica se a missão foi concluída.

A classe possui um construtor padrão `MissaoKingdomKalb()` que chama o método `popularMissao()`, que preenche os detalhes da missão, como o nome da missão, a descrição, o prêmio em moedas ao aceitar a missão e o prêmio em moedas ao finalizá-la.

O método `oferecerMissao(Personagem maxwell)` é responsável por apresentar a missão ao jogador. Ele exibe uma mensagem perguntando se o jogador deseja aceitar a missão e fornece informações sobre os prêmios que serão recebidos ao aceitar e finalizar a missão. O jogador é solicitado a fornecer uma resposta (1 para "Sim" e 2 para "Não") usando o objeto `Scanner`.

Com base na resposta do jogador, as variáveis `aceitou`, `emAndamento` e `finalizou` são atualizadas. Se o jogador aceitar a missão (resposta igual a 1), `aceitou` é definido como `true`, `emAndamento` é definido como `true` e

`finalizou` é definido como `false`. Além disso, o método `atualizarBensAceitarMissao(maxwell)` é chamado para atualizar os recursos do jogador com base na decisão. Caso contrário, se o jogador não aceitar a missão (resposta igual a 2), `aceitou` é definido como `false`, `emAndamento` é definido como `false` e `finalizou` é definido como `false`.

O método `popularMissao()` é responsável por preencher os detalhes da missão. Ele define os valores das variáveis `nome`, `descricao`, `statusConcluida`, `premioMoedaAceitar`, `premioLimiar` e `premioMoedaFinalizar` com os dados específicos da missão "KingdomKalb".

O método `podeRealizarMissao(Personagem maxwell)` verifica se o jogador pode realizar a missão. Ele primeiro verifica se o jogador está vivo chamando o método `checaStatusPersonagem()` do objeto `maxwell` (que é uma instância da classe `Personagem`). Em seguida, verifica se a missão ainda não foi concluída, verificando o valor da variável `statusConcluida`. Se ambas as condições forem verdadeiras, o método retorna `true`, indicando que o jogador pode realizar a missão. Caso contrário, retorna `false`.

O método `finalizarMissao(Personagem maxwell)` é chamado quando o jogador deseja finalizar a missão. Ele verifica se a missão já foi concluída, verificando o valor da variável `statusConcluida`.

Se a missão já foi concluída, o método define `finalizou` como `true`, `emAndamento` como `false` e chama o método `atualizarBensFinalizarMissao(maxwell)` para atualizar os recursos do jogador ao finalizar a missão. Uma mensagem de parabéns é exibida, juntamente com o total de moedas de transporte e o limiar da joia do jogador. Caso a missão não tenha sido concluída, o método verifica se o jogador aceitou a missão. Se o jogador tiver aceitado, uma mensagem é exibida informando que a missão não foi finalizada. Caso contrário, o método simplesmente retorna.

Os métodos `atualizarBensAceitarMissao(Personagem maxwell)` e `atualizarBensFinalizarMissao(Personagem maxwell)` são responsáveis por atualizar os recursos do jogador quando a missão é aceita e finalizada, respectivamente.

Eles verificam se o jogador atende aos pré-requisitos para realizar a missão, chamando o método `podeRealizarMissao(maxwell)`. Se o jogador atender aos pré-requisitos, os recursos do jogador são atualizados. Ao aceitar a missão, o método adiciona uma quantidade específica de moedas ao total de moedas de transporte do jogador. Ao finalizar a missão, o método adiciona uma quantidade específica de moedas e aumenta o limiar da joia do jogador.

A classe também possui métodos getters e setters para acessar e modificar os valores das variáveis de instância.

classe MISSÃO VUNESE

A classe declara várias variáveis de instância, como `nome`, `descricao`, `statusConcluida` e outras. Essas variáveis são usadas para armazenar informações sobre a missão. Por exemplo, `nome` armazena o nome da missão, `descricao` armazena uma descrição da missão e `statusConcluida` indica se a missão foi concluída.

A classe possui um construtor padrão `MissaoVunese()`, que é chamado quando um objeto da classe é criado. Esse construtor chama o método `popularMissao()`, que preenche os detalhes da missão, como o nome da missão, a descrição e os prêmios em moedas e limiar da joia.

O método `oferecerMissao(Personagem maxwell)` é responsável por apresentar a missão ao jogador. Ele exibe uma mensagem perguntando se o jogador deseja aceitar a missão e fornece informações sobre os prêmios que serão recebidos ao aceitar e finalizar a missão. O jogador é solicitado a fornecer uma resposta (1 para "Sim" e 2 para "Não") usando o objeto `Scanner`.

Com base na resposta do jogador, as variáveis `aceitou`, `emAndamento` e `finalizou` são atualizadas. Se o jogador aceitar a missão (resposta igual a 1), `aceitou` é definido como `true`, `emAndamento` é definido como `true` e `finalizou` é definido como `false`. Além disso, o método `atualizarBensAceitarMissao(maxwell)` é chamado para atualizar os recursos do jogador com base na decisão. Caso contrário, se o jogador não aceitar a missão (resposta igual a 2), `aceitou` é definido como `false`, `emAndamento` é definido como `false` e `finalizou` é definido como `false`.

Se o jogador aceitou a missão, uma mensagem é exibida com a descrição da missão e o prêmio em moedas que o jogador receberá ao aceitar a missão. Além disso, é exibido o total de moedas de transporte e o limiar da jóia do jogador antes de iniciar a missão.

O método `popularMissao()` é responsável por preencher os detalhes da missão. Ele define os valores das variáveis `nome`, `descricao`, `statusConcluida`, `premioMoedaAceitar`, `premioLimiar` e `premioMoedaFinalizar` com os dados específicos da missão "Vunese".

O método `podeRealizarMissao(Personagem maxwell)` verifica se o jogador pode realizar a missão. Ele primeiro verifica se o jogador está vivo chamando o método `checaStatusPersonagem()` do objeto `maxwell` (que é uma instância da classe `Personagem`). Em seguida, verifica se a missão ainda não foi concluída, verificando o valor da variável `statusConcluida`. Se ambas as condições forem verdadeiras, o método retorna `true`, indicando que o jogador pode realizar a missão. Caso contrário, retorna `false`.

O método `finalizarMissao(Personagem maxwell)` é responsável por finalizar a missão. Ele verifica se a missão já foi concluída. Se sim, define `finalizou` como `true`, `emAndamento` como `false` e chama o método `atualizarBensFinalizarMissao(maxwell)` para atualizar os recursos do jogador

ao finalizar a missão. Em seguida, exibe uma mensagem parabenizando o jogador por concluir a missão e mostra o total de moedas de transporte e o limiar da joia do jogador após a conclusão. Se a missão não tiver sido concluída, o método verifica se o jogador aceitou a missão. Se o jogador tiver aceitado, uma mensagem é exibida informando que a missão não foi finalizada. Caso contrário, o método simplesmente retorna.

Os métodos `atualizarBensAceitarMissao(Personagem maxwell)` e `atualizarBensFinalizarMissao(Personagem maxwell)` são responsáveis por atualizar os recursos do jogador quando a missão é aceita e finalizada, respectivamente. Eles verificam se o jogador atende aos pré-requisitos para realizar a missão, chamando o método `podeRealizarMissao(maxwell)`. Se o jogador atender aos pré-requisitos, os recursos do jogador são atualizados. Ao aceitar a missão, o método adiciona uma quantidade específica de moedas ao total de moedas de transporte do jogador. Ao finalizar a missão, o método adiciona uma quantidade específica de moedas e diminui o limiar da joia do jogador.

A classe também possui métodos getters e setters para acessar e modificar os valores das variáveis de instância.

classe MISSÃO DELFASIA

A estrutura da classe MissaoDefalsia foi utilizada para representar uma missão na cidade de Defalsia dentro do jogo.

Atributos: A classe possui vários atributos privados, como nome, descricao, statusConcluida, aceitou, finalizou, emAndamento, premioMoedaAceitar, premioLimiar, premioMoedaFinalizar e sc. Esses atributos armazenam informações sobre a missão, seu status, prêmios e um objeto Scanner para entrada de dados do jogador.

Construtor: A classe possui um construtor MissaoDefalsia() que chama o método popularMissao() para definir os valores iniciais da missão. Esse construtor é chamado quando um objeto MissaoDefalsia é criado.

Métodos:

oferecerMissao(): Esse método é responsável por oferecer a missão ao jogador. Ele exibe uma mensagem com a descrição da missão e pergunta ao jogador se ele deseja aceitá-la. Dependendo da resposta, o atributo aceitou é definido como true ou false, e o método atualizarBensAceitarMissao() é chamado para atualizar os bens do jogador caso ele aceite a missão.

popularMissao(): Esse método preenche os atributos da missão com valores pré-definidos, como o nome, descrição, premiações, etc.

podeRealizarMissao(): Esse método verifica se o jogador possui os pré-requisitos necessários para realizar a missão. Ele chama o método checaStatusPersonagem() do objeto maxwell (que deve ser uma instância da

classe Personagem) para verificar se o jogador está vivo, e também verifica se a missão já foi concluída.

finalizarMissao(): Esse método é chamado quando o jogador decide finalizar a missão. Ele verifica se a missão já foi concluída e, se sim, atualiza os bens do jogador chamando o método atualizarBensFinalizarMissao() e exibe uma mensagem de parabéns. Caso a missão não tenha sido concluída, o método verifica se o jogador aceitou a missão ou não.

atualizarBensAceitarMissao(): Esse método é responsável por atualizar os bens do jogador caso ele aceite a missão. Ele verifica se o jogador possui os pré-requisitos necessários chamando o método podeRealizarMissao() e, se sim, adiciona uma quantidade específica de moedas aos bens do jogador.

atualizarBensFinalizarMissao(): Esse método é responsável por atualizar os bens do jogador caso ele finalize a missão com sucesso. Ele também verifica se o jogador possui os pré-requisitos necessários chamando o método podeRealizarMissao() e, se sim, adiciona uma quantidade específica de moedas e aumenta o limiar da joia.

Outros métodos são getters e setters para acessar e modificar os atributos da classe.

classe JOGO

A estrutura da classe Jogo foi utilizada para representar o jogo em si, contendo a lógica principal do funcionamento do jogo.

Atributos:

mapa: Representa uma instância da classe Mapa, responsável por armazenar e exibir informações sobre o mapa do jogo.

maxwell: Representa uma instância da classe Personagem, que é o personagem principal do jogo.

transporte: Representa uma instância da classe Transporte, responsável por gerenciar as ações de transporte no jogo.

sc: Representa um objeto da classe Scanner para capturar a entrada de dados do jogador.

jogoAtivo: Indica se o jogo está ativo ou não.

personagemVivo: Indica se o personagem principal está vivo ou não.

Construtor:

O construtor Jogo() é responsável por inicializar o jogo e chamar o método iniciar().

Métodos:

exibeMenu(): Exibe o menu principal do jogo, mostrando opções disponíveis e permitindo que o jogador realize ações, como exibir o mapa, realizar transporte ou sair do jogo.

statusJogo(): Verifica o status do jogo com base nos atributos do personagem maxwell e toma ações apropriadas caso o personagem fique sem moedas ou atinja o limite de poder da joia.

iniciar(): Controla o fluxo do jogo, exibindo o menu e verificando o status do jogo até que o jogo seja encerrado.

personagemMorrePorMoeda(): Exibe uma mensagem quando o personagem fica sem moedas e oferece ao jogador a opção de recomeçar o jogo ou sair.

personagemMorrePorLimiar(): Exibe uma mensagem quando o personagem atinge o limite de poder da joia e oferece ao jogador a opção de recomeçar o jogo ou sair.

aguardar(int segundos): Método auxiliar para pausar a execução do jogo por um determinado número de segundos.

limparTela(): Método auxiliar para limpar a tela do console.

classe TRANSPORTE

A classe possui um atributo privado mapa, que representa o mapa do jogo.

Também possui um objeto Scanner chamado sc para ler a entrada do usuário.

Há uma variável booleana transporteRealizado que indica se o transporte foi realizado com sucesso.

Existem três objetos de missões (missaoKalb, missaoDefalsia e missaoVunese) que são utilizados para realizar e finalizar missões específicas.

O construtor Transporte() instancia um novo objeto Mapa e atribui à variável mapa.

O método iniciarTransporte recebe um objeto Personagem chamado maxwell como parâmetro e permite ao jogador escolher para qual cidade deseja ir.

O método realizarTransporte é chamado pelo método iniciarTransporte e efetua o transporte para a cidade escolhida pelo jogador. Ele verifica se o personagem possui moedas suficientes e realiza o transporte, atualizando a cidade atual do personagem e subtraindo uma moeda. Caso o personagem tenha apenas uma

moeda, é oferecida a opção de falar novamente com o mercador ou continuar o transporte.

O método atualizaBens é chamado após o transporte e atualiza os atributos do personagem com base na cidade de destino. Dependendo da cidade, os poderes e missões do personagem são afetados.

O método exibeDialogo simula um diálogo entre o personagem e um mercador na cidade de destino. Ele faz várias perguntas ao jogador e atualiza os atributos do personagem com base nas respostas.

O método fazDesenho exibe uma animação de viagem para dar uma sensação de movimento.

O método exibirFrameViagem exibe um frame da animação de viagem.

Os métodos aguardar e limparTela são métodos auxiliares para pausar a execução e limpar a tela do console, respectivamente.

classe MAPA

A classe "Mapa" é usada para representar um mapa de cidades e suas vizinhanças. A estrutura de dados escolhida para armazenar as informações é um HashMap com chaves do tipo String (representando o nome das cidades) e valores do tipo List<String> (representando as cidades vizinhas).

A escolha do HashMap permite o acesso eficiente aos valores com base nas chaves. Cada cidade é mapeada para uma lista de suas cidades vizinhas. Dessa forma, é possível obter rapidamente a lista de vizinhos de uma cidade específica.

Ao adicionar uma cidade, é criada uma nova entrada no HashMap com a chave sendo o nome da cidade e a lista de vizinhos inicialmente vazia. O método addVizinho é responsável por adicionar uma cidade vizinha à lista de vizinhos de uma determinada cidade.

O cálculo da distância entre duas cidades é feito utilizando uma busca em largura (BFS) no grafo representado pelo mapa. A distância é medida pelo número de arestas percorridas. Esse cálculo é feito no método calcularDistancia, que recebe as cidades de origem e destino como parâmetros.

Além disso, a classe possui outros métodos auxiliares para obter a lista de vizinhos de uma cidade, obter a lista de todas as cidades do mapa, imprimir o mapa de forma visual e verificar se duas cidades são vizinhas.

Essa estrutura de classe e a escolha do HashMap como estrutura de dados interna foram feitas com o objetivo de facilitar a representação e manipulação de um mapa de cidades e suas vizinhanças, permitindo realizar operações como cálculo de distância e verificação de vizinhança de forma eficiente.

Acesso ao código completo(GitHub):
https://github.com/melissaods/jogo-gr-merchant-adventures.git