**Puzzles/Salas:**

O PuzzleDialog funciona como um enquirer para o componente Puzzle, de forma que para cada instância de um dado Puzzle, existe um PuzzleDialog associado a ele, pensei ser útil isso caso quiséssemos fazer um Puzzle depender do outro (numa relação de continuidade, por exemplo). Ainda, uma instância de cada "Enquirer" poderia ser justificada pelo estado independente de cada puzzle, que ficaria salvo no componente PuzzleDialog que gerencia o Puzzle a ele associado.

Cada um dos 2 componentes implementa uma interface, com os seus respectivos nomes (IPUzzleDialog e IPuzzle). Assim do diagrama de componentes podemos ver que a função da interface implementada pelo PuzzleDialog é se comunicar com o player (mundo externo ao modo sala), recebendo os inputs digitados pelo player e outputs advindos do puzzle.

Vamos aos métodos:

**Componente PuzzleDialog** -> Implementando a interface IPuzzleDialog, ele define:

*Atributos:*

**- int**maxTime: para cada puzzle um tempo máximo se for caso.

**-** **int**dPuzzleId: Um modo de identificar este componente quando for instanciado, numericamente, puzzle 1, 2, 3, ... quantos houverem .

*Métodos:*

+inputMsg(String msgIn): recebe o comando digitado pelo jogador, e retorna de acordo com a situação do puzzle (pode fazer isso, não pode, etc etc..)

+outputMsg(String puzzleInString): recebe uma string do vetor de strings existentes no puzzle para repassá-la ao ao player, podendo ser aplicada na mensagem a função da zoeira, que iria embaralhar a mensagem original a critério da engine do game, ou simplesmente adicionar algo à mensagem do puzzle em função de um evento ocorrido durante a resolução do puzzle.

+setMaxTime(): Altera o valor de maxTime.

+setDPuzzleId(): Altera o valor de dPuzzleId.

+getDPuzzleId(): Método para pegar o numero do Puzzle.

+getCorridorPosition(Int gameProgress): Método que vai decidir qual puzzle o player vai entrar em função da posição dele no mapa, podemos definir valores fixos, posicao 15 do mapa, puzzle 1, 35 puzzle 2, e assim por diante. Ou se o mapa for gerado proceduralmente, então percentualmente (pos.atual/pos.final), quando chegar a x% será carregado o enésimo puzzle).

+getMaxTime(): Acessa o tempo limite estabelecido.

**Componente Puzzle** -> Implementa a interface IPuzzle, e instância cada Puzzle de acordo com o tipo dele implementa os métodos da interface conforme for conveniente.

*Atributos:*

**-** **int**puzzleId: Número que representa o puzzle corrente.

**-** **HashTable<String, Boolean>** Tasks**:** Permite a alocação de um pequeno texto descritivo das tarefas demandadas pelo puzzle e ainda associa a cada uma delas (HashTable extendsDictionary) um valor booleano (inicialmente false para todos). Independentemente do tipo do puzzle criado, ele tem uma sequência de tarefas que pode ser abstraída e armazenada dessa forma.

**- Vector<String>** messages: Armazena, de forma sequencial, as mensagens que seriam enviadas ao player referentes a cada etapa do puzzle. As mensagens viriam de um banco de dados conforme demanda do puzzle em corrente funcionamento.

*Métodos:*

+getRoomPos(): retorna a posição atual do jogador dentro da room para o puzzle, caso seja um puzzle labiríntico ou algo do gênero.

+getPuzzleId(): Método de acessar o puzzleId.

+getItems(): *Este método pode não ser necessário a depender de como será dado o acesso aos items de puzzle, que foram criados como classes e podem assim ser acessados diretamente, o método pode no entanto servir para padronizar o acesso aos itens para todos os tipos de puzzles.*

+combineItems(): em puzzles no estilo first times, items podem precisar ser combinados, o que a depender da implementação dos itens pode ser apenas a mudança de um status do item, ou um adjetivo, como está sugerido pela equipe dos Itens.

+checkCurrTask(): Pode ser usado para dar alguma dica ao jogador sobre a tarefa que está correntemente realizando, e para verificar internamente no programa se a tarefa foi terminada por completo, essencialmente verifica já pode trocar o estado de uma task da HashTable para true (= Terminada).

+setSequenceOfTasks(): Este método basicamente adiciona a descrição e o estado das tarefas que seriam realizadas sequencialmente. Ele pode ser implementado pela manipulação de um arquivo externo (.txt) que teria as definições do puzzle.

+setNewStatus(): método para efetivamente alterar a Hastable(que é private).

+Timer(): implementa o temporizador (tempo transcorrido dentro do puzzle) se o puzzle necessitar, e também implementa uma verificação periódica se o tempo do Timer não ultrapassa o tempo definido no PuzzleDialog, por meio do método getMaxTime(). O método provavelmente chamará o timer da biblioteca javax.swing.Timer definindo os parâmetros corretamente do temporizador.

+addMessage(): Insere as mensagens de cada puzzle no Vector<String>. Ele pode ser implementado pela manipulação de um arquivo externo (.txt) que teria as definições do puzzle.

**Interface IPuzzleDialog** -> Demanda a implementação dos métodos inputMsg(String msgIn) e outputMsg(), que em resumo serão responsáveis por fazer a comunicação com o modo sala, definido como um dos 5 macro módulos do jogo. Este modo sala necessita receber as mensagens digitas pelo jogador no console ou terminal (será desenvolvido graficamente por meio da manipulação da biblioteca gráfica LibGDX). As mensagens serão recebidas pelo componente que efetivamente implementa estes métodos e possui função de gerenciar todos os parâmetros necessários à instanciação do Puzzle. Este por sua vez poderá gerar os Puzzles em si por meio do Pattern Factory.

**Interface IPuzzle** -> Demanda a implementação dos 4 métodos que formam um conjunto de características mínimas para qualquer tipo de puzzle, estas características são:

- Sequência de tarefas a ser realizada para a conclusão do puzzle. (método setSequenceOfTasks());

- Informações tanto para o player quanto para o macro módulo Sala em si sobre a tarefa que está pendente no momento (não são esperadas tarefas concorrentes num mesmo Puzzle) (checkCurrTask(String msgIn));

- Método para alterar a HashTable que guarda o estado atual da tarefa corrente (estados: completa = true; incompleta = false; dependendo do estado desta tarefa uma mensagem diferente pode ser passada ao usuário (Hints) ) (setNewStatus());

- Um timer deve ser implementado internamente para cada Puzzle, o tempo será contabilizado independentemente do puzzle possuir um tempo para finalização, de modo que ao final do jogo uma forma de Scores de conclusão de cada Puzzle possa ser exibida ao término do jogo. (Timer() – implementará um temporizador da biblioteca *javax.swing.Timer*);