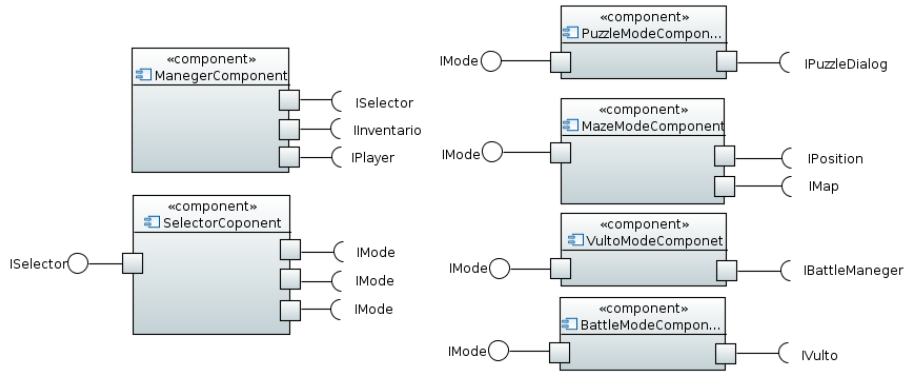
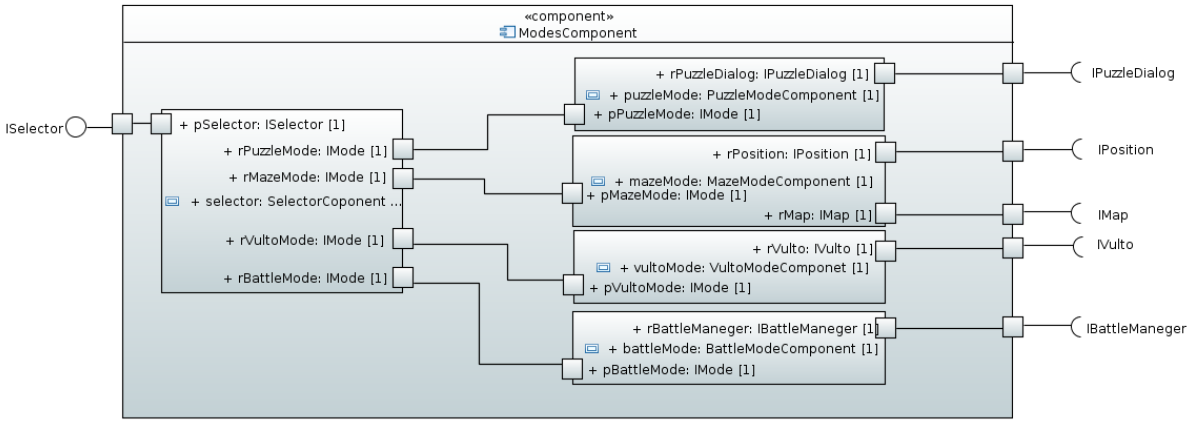
Main Game:

Responsável pelo gerenciamento do jogo, esse módulo irá se encarregar de realizar as principais comunicações entre os demais e a comunicação final com o usuário, bem como gerenciar os principais recursos da framework libgdx utilizada.

A classe principal requer as interfaces IInventario, IPlayer e ISelector. Esse módulo, porém, apresenta o componente Modulos, que irá prover a interface ISelector e requer as interfaces IPuzzleDialog, IPosition, IMap, IVulto e IBattleManeger.





<u>Princípios de Projeto utilizados:</u>

<u>Dependency Inversion Principle:</u> Toda a comunicação do módulo é feita através de interfaces, tendo objetos que, mesmo se comportando diferente internamente, oferecem os mesmos métodos com um comportamento genérico.

<u>Open Closed Principle:</u> Novos modos podem ser acrescentados sem que haja a necessidade de alterar os demais.

<u>Interface Segregation Principle:</u> Cada interface carrega pouca responsabilidade, como por exemplo há uma interface apenas para enviar as informações para o usuário, outra apenas para selecionar e passar informações para o modo correto e assim por diante.

Padrões de Projeto utilizados:

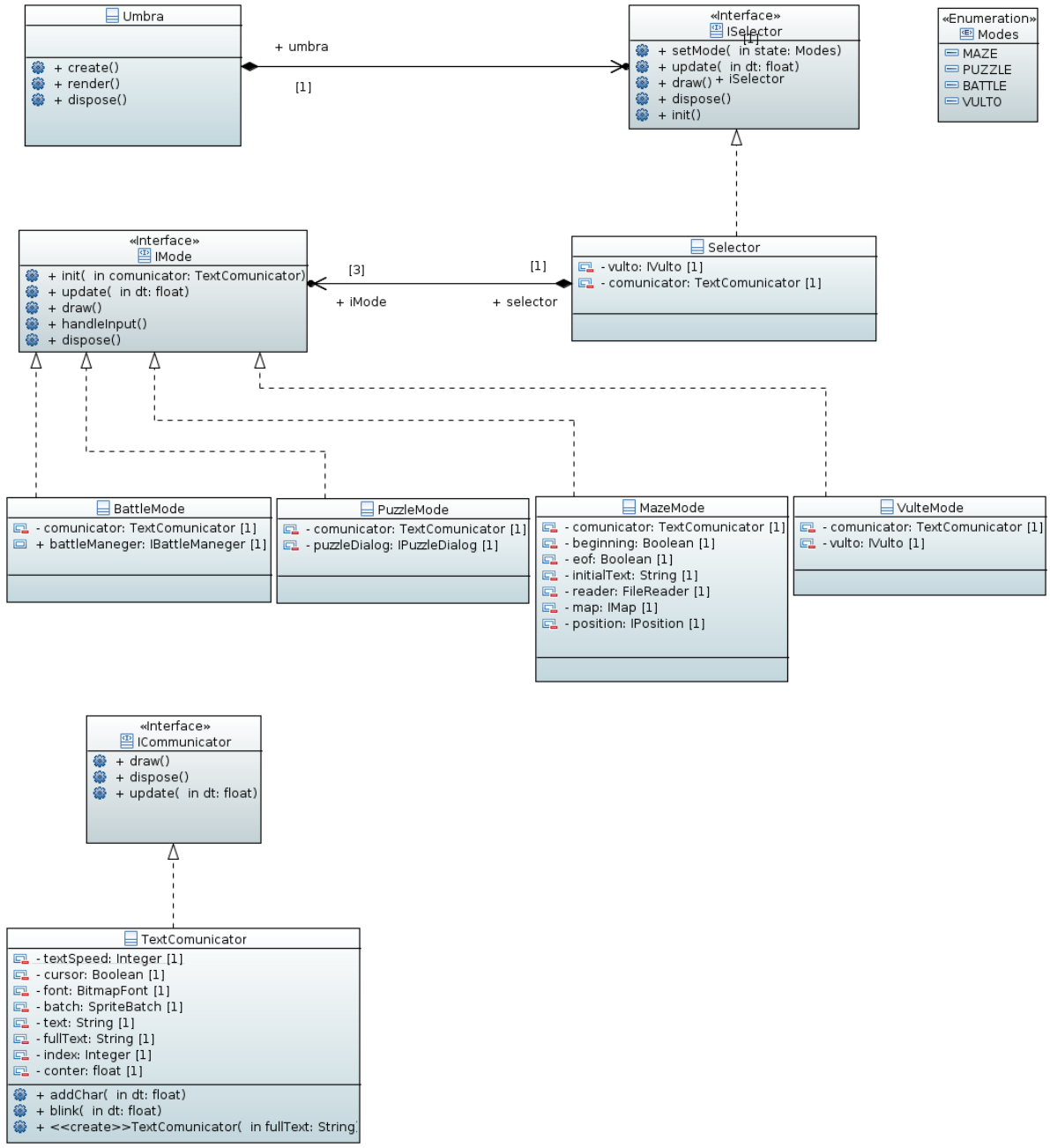
State: O padrão State é adotado para alterar o comportamento do jogo de acordo com seu modo. Isso é feito ao trocar a instância de modo em Selector.

```
17  public void setMode(Modes state){
18      switch (state){
19          case BATLLE:
20              mode = ModesInstantiator.battleModeInstance();
21              break;
22          case MAZE:
23              mode = ModesInstantiator.mazeModeInstance();
24              break;
25          case PUZZLE:
26              mode = ModesInstantiator.puzzleModeInstance();
27              break;
28          case VULTO:
29              mode = ModesInstantiator.vultoModeInstance();
30              ModesInstantiator.vultoModeReset();
31              break;
32      }
33  }
34  public void update(float dt){
35      if(vulto.checkVulto()){
36          setMode(Modes.VULTO);
37      }
38      mode.handleInput();
39      mode.update(dt);
40  }
```

Singleton: O padrão Singleton é utilizado para se garantir que só haja uma única instância de cada modo.

```
6   static IMode puzzleModeInstance(){
7       if(uniquePuzzleMode == null){
8           uniquePuzzleMode = new PuzzleMode();
9           uniquePuzzleMode.init();
10      }
11      return uniquePuzzleMode;
12  }
13  static IMode mazeModeInstance(){
14      if(uniqueMazeMode == null){
15          uniqueMazeMode = new MazeMode();
16          uniqueMazeMode.init();
17      }
18      return uniqueMazeMode;
```

Diagrama de classes:



Sub componentes:

SelectorComponent: Componente responsável pelo controle dos modos de jogo e suas interações.

Interfaces:

- Provida: ISelector
- Requeridas: Um IMode para cada modo do jogo (no caso quatro IModes).

PuzzleModeComponent: Cuida da comunicação com o componente Puzzle.
Interfaces:

- Provida: IMode.
- Requerida: IPuzzleDialog.

MazeModeComponent: Cuida da comunicação com o componente Map
Interfaces:

- Provida: IMode.
- Requerida: IPosition, IMap.

VultoModeComponent:Cuida da comunicação com o componente Vulto.
Interfaces:

- Provida: IMode.
- Requerida: IVulto.

BattleModeComponent:Cuida da comunicação com o componente Battle.
Interfaces:

- Provida: IMode.
- Requerida: IBattleManeger.

Interfaces:

ISelector: interface para seleção e gerenciamento do módulo correto, além da transição entre modos.
Métodos:

- init: chamado apenas uma vez, na inicialização. Inicializa os atributos e chama setMode para o modo Maze.
- setMode: recebe o modo a ser selecionado e muda para esse modo.
- update: chamado a cada atualização do jogo, chama update e handleInput do modo selecionado.
- draw: chamado a cada atualização do jogo, chama draw do modo selecionado.
- dispose: chamado apenas uma vez, quando o selector não for mais utilizado. Descarta o modo atual.

IMode: Interface de comunicação com os modos do jogo.
Métodos:

- init: chamado na inicialização do modo, inicializa os atributos e realiza outras configurações iniciais que sejam necessárias.
- update: chamado a cada atualização do jogo, é responsável por atualizar o valor dos atributos e envia e recebe informações necessárias dos demais componentes.
- draw: chamado a cada atualização do jogo, é responsável por atualizar a imagem da tela vista pelo usuário.
- handleInput: Verifica a entrada fornecida pelo usuário e a redireciona para o componente que irá utilizá-la.
- dispose: chamado apenas uma vez, quando o modo não for mais utilizado. Descarta os atributos que precisam ser descartados e realiza outras ações finais.

Implementação interna dos componentes:

IComunicator: Classes que implementam esta interface são responsáveis pela comunicação direta com o usuário:
Métodos:

- update: atualiza a mensagem a ser impressa, retorna verdadeiro quando a mensagem estiver completa.
- draw: imprime a mensagem.
- dispose: para o descarte da instância.

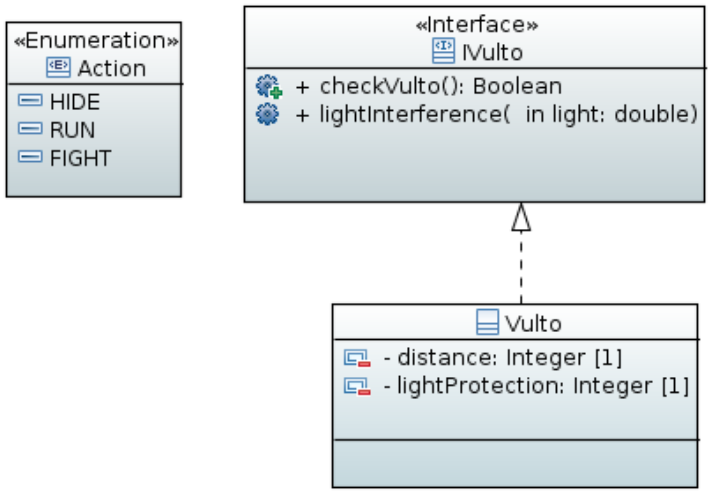
TextCommunication: Implementa IComunicator, usado para imprimir textos.
Métodos internos:

- addChar: adiciona o próximo caractere da mensagem.
- blink: faz o cursor piscar alterando o estado dele para apagando ou aparecendo.

ModesInstanciator: Responsável por instanciar os modos e os reiniciar quando necessário.

Vulto

Esta módulo é responsável por guardar o estado do vulto e simular as alterações que ocorram nele no decorrer do jogo.



Princípios de Projeto utilizados:

Common Reuse Principle: Embora este componente seja muito simples, sua independência em relação ao resto do programa fez com que ele seja mantido separado.

Padrões de Projeto utilizados:

Singleton: Garante que haja apenas uma instância de vulto.

Interfaces:

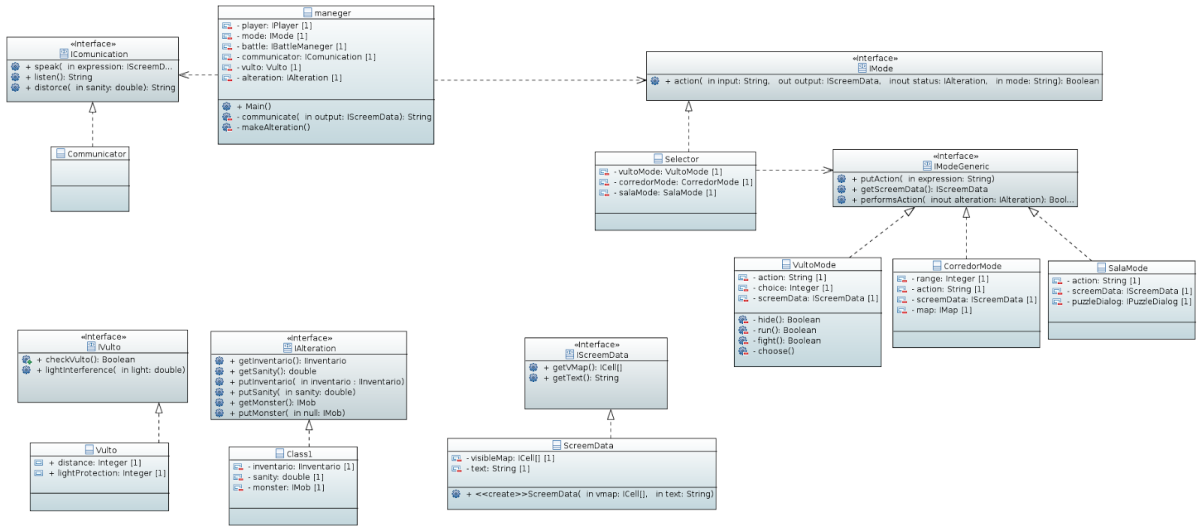
IVulto: Interface para controle do vulto e ações relacionadas a ele.
Métodos:

- checkVulto: verifica se o vulto alcançou o jogador.
- lightInterference: aplica a interferência da luz originada de determinados itens do jogador na posição do vulto.
- chooseAction: chamado quando o vulto alcança o jogador, tem como parâmetro a ação escolhida pelo usuário.

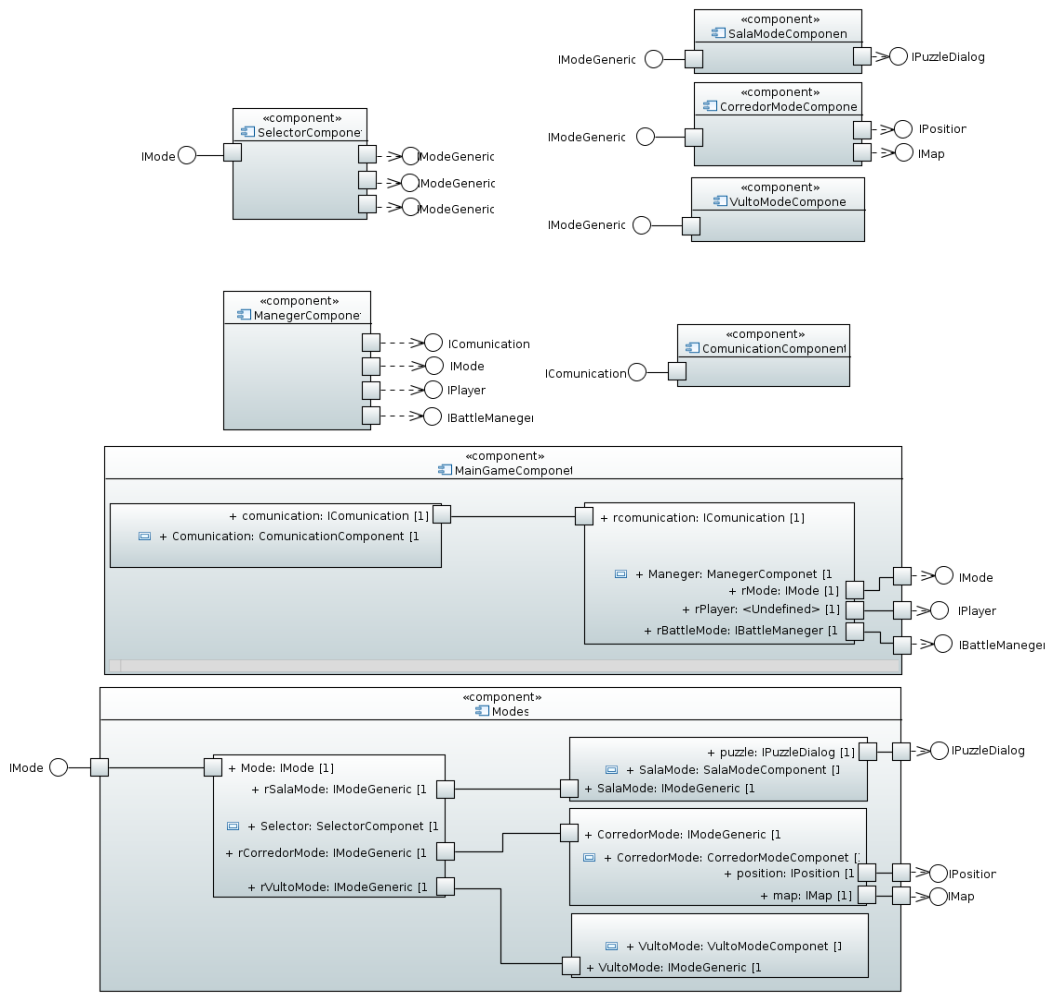
Implementação interna do component:
VultoSingleton: Classe responsável por instanciar Vulto.

Evolução Main Game e Vulto

Antigo diagrama de classes:



Antigo diagrama de componentes:



Principais alterações:

Adoção do Padrão State:

Na versão anterior uma mudança de modo do jogo consistia apenas em alterar a classe responsável por se comunicar com os outros componentes, deste modo ficava a cargo do selector e do manager checar constantemente o modo atual e então realizar as ações relativas a ele.

Na nova versão, cada modo é um estado diferente do selector, desta forma para alterar o modo do jogo basta mudar o modo instanciado em selector. Assim todo o comportamento do jogo se adéqua ao novo modo.

Separação do componente Vulto:

Na versão anterior, o VultoModeComponente era responsável integralmente pelo vulto, diferente dos outros Modes, que apenas se encarregavam de gerenciar os componentes que implementam os demais objetos do jogo. Na nova versão existe um componente separado para simular o vulto.

Criação do BattleModeComponent:

Na versão anterior não havia um componente interno do Main Game para conectar o Componete Battle com o restante do jogo, desta forma o Componete Battle ficava com responsabilidades que deveriam pertencer ao MainGame, como por exemplo se comunicar com o usuário.

Na nova versão, criou-se o componente BattleModeComponent, responsável por gerenciar o componente Battle.

Alteração na abordagem do Comunicator:

Na versão anterior, a comunicação com o usuário consistia em reunir os dados gerados por cada modo em um único objeto e os levar ao Comunicator, onde eles eram processados e a comunicação com o usuário era efetuada.

Na nova versão, os objetos que implementam IComunicator são capazes de se comunicar com o usuário e os modos possuem uma instância própria para realizar a comunicação.