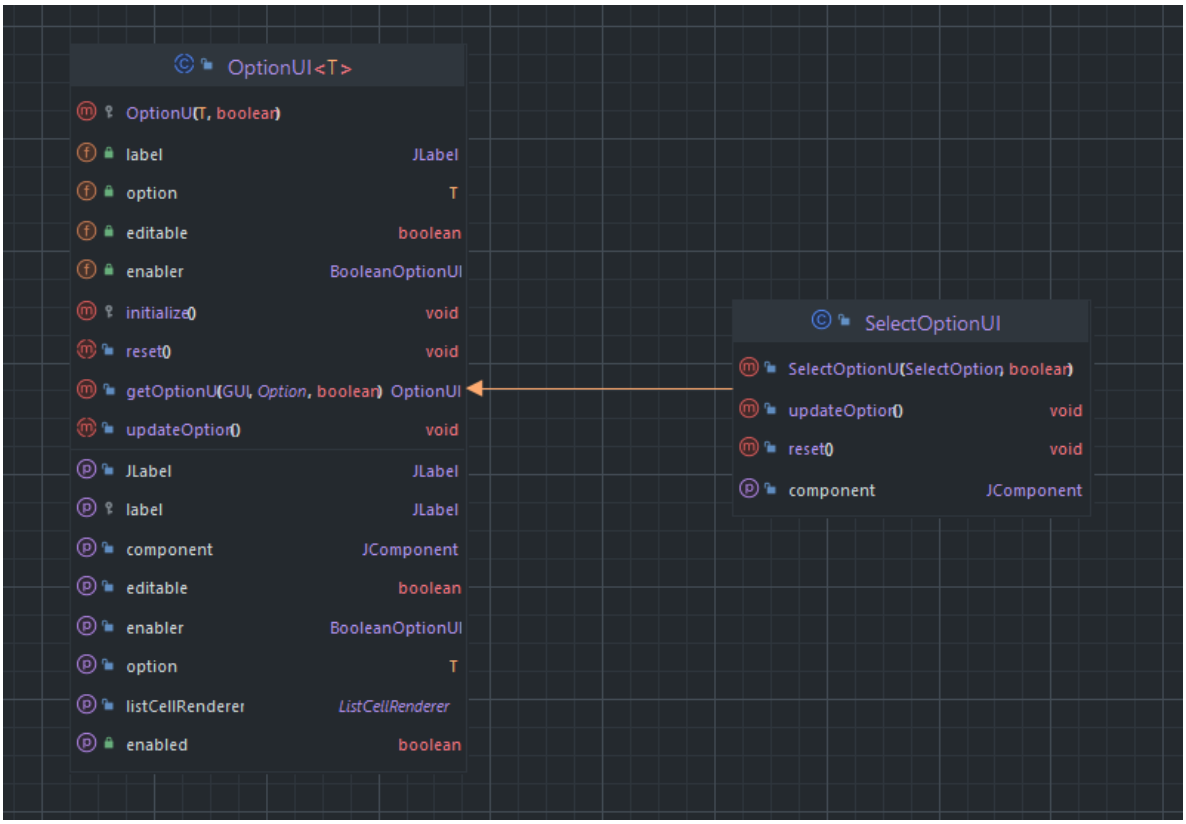
Design Patterns

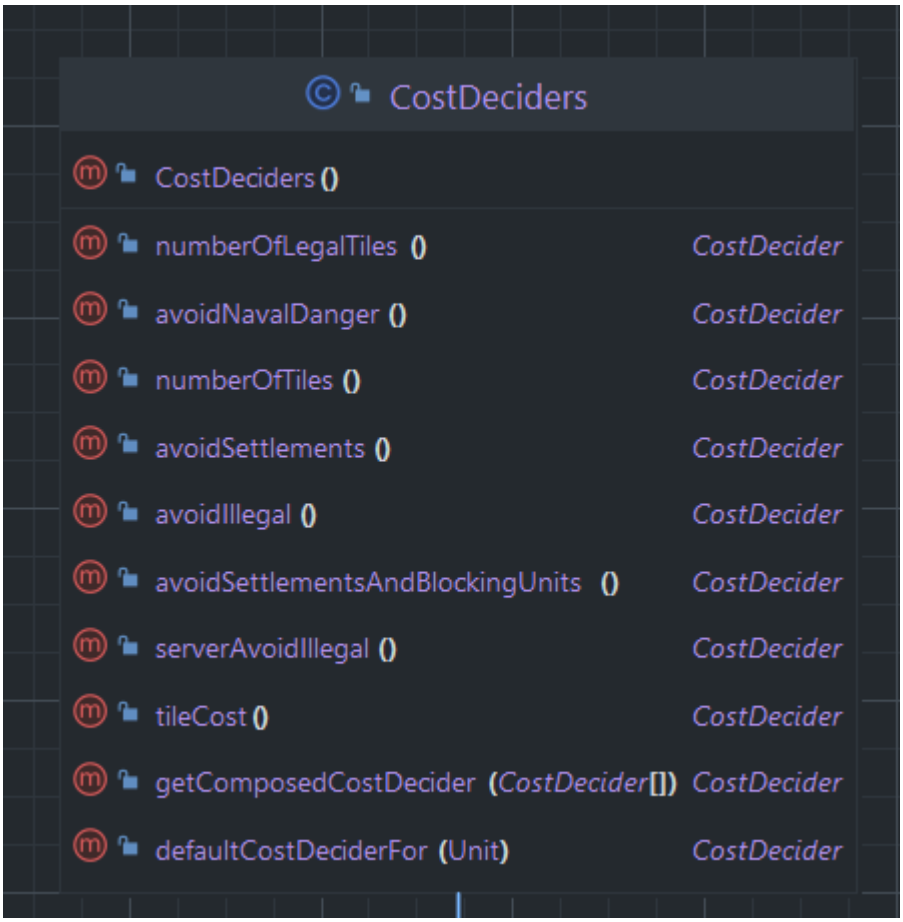
1- Strategy Pattern



Localização: “src/net/sf/freecol/cliente/gui/option”

Estas classes representam um Strategy Pattern, pois SelectOptionUI é uma classe concreta que implementa a classe OptionUI, indicando que fornece uma implementação específica para lidar com opções de seleção.

2- Composite Pattern

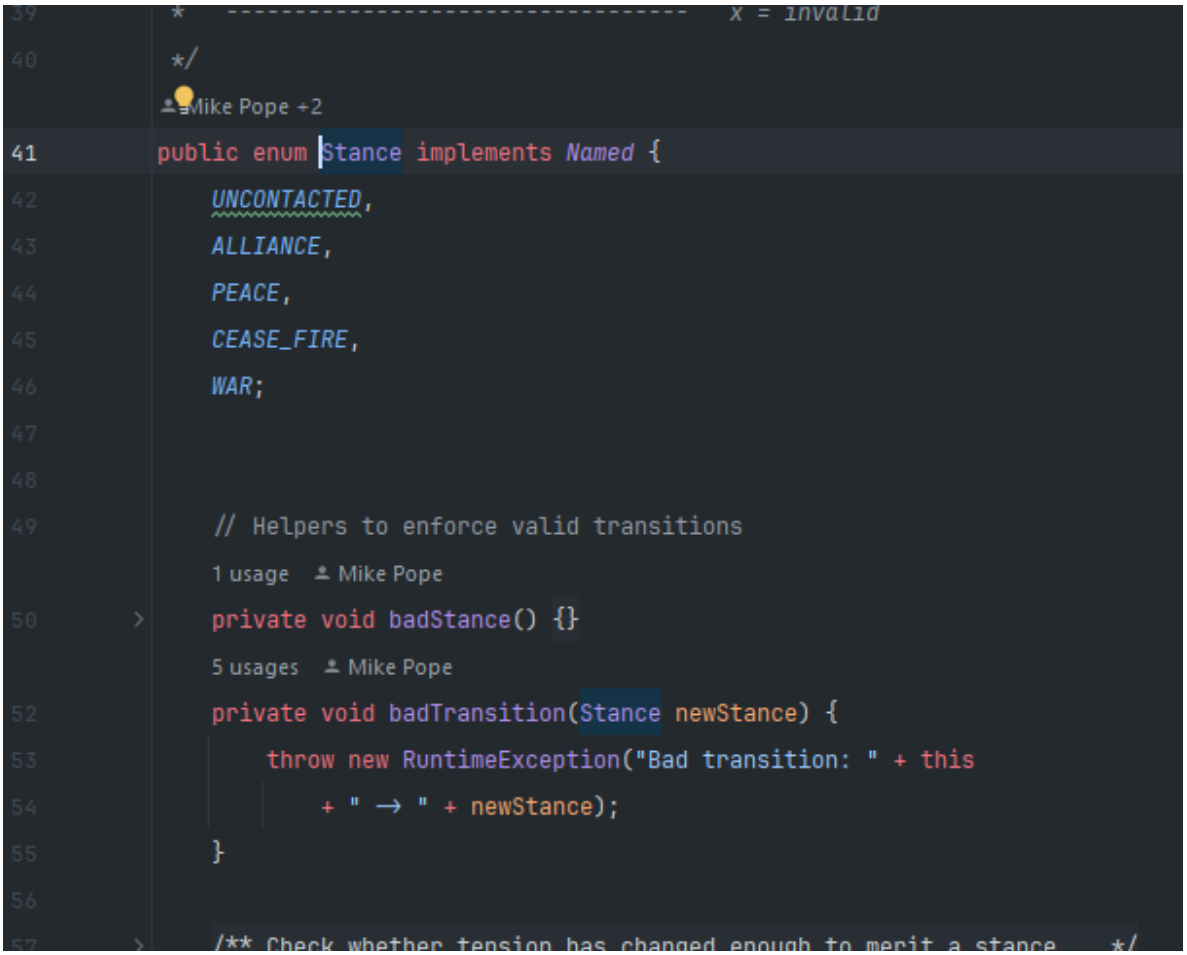


Localização: “src/net/sf/freecol/common/model/pathfinding/CostDeciders.java”

Esta classe apresenta um Composite Pattern. Podemos concluir isto pois o método getComposedCostDecider(CostDecider[]) sugere-nos que esta classe pode compor vários objetos do tipo CostDecider para criar um CostDecider mais complexo. Isto significa que CostDeciders pode criar um novo CostDecider que combina o comportamento de outro objetos CostDecider. Esta capacidade de compor objetos de forma hierárquica é uma característica do Composite Pattern.


* @return A new {@code CostDecider} composed of the argument
* cost deciders.
*/
Michael Pope +2
public static CostDecider getComposedCostDecider(final CostDecider... cds) {
if (cds.length < 2) {
throw new RuntimeException("Short CostDecider list: " + cds.length);
}
Michael Pope +2
return new CostDecider() {
4 usages
private final CostDecider[] costDeciders = cds;
3 usages
private int ret = -1;
6 usages
private int index = -1;
Michael Pope +1
@Override

3- State Pattern



Localização: src/net/sf/freecol/common/model/Stance.java

O diagrama representa uma classe que apresenta um State Pattern, pois encapsula diferentes estados para gerenciar transições de estado (badTransition(Stance)) e obter comportamentos específicos do estado (badStance()). Além disso, a presença de métodos estáticos como values() e valueOf(String) para gerenciamento de estado também suporta esse padrão.