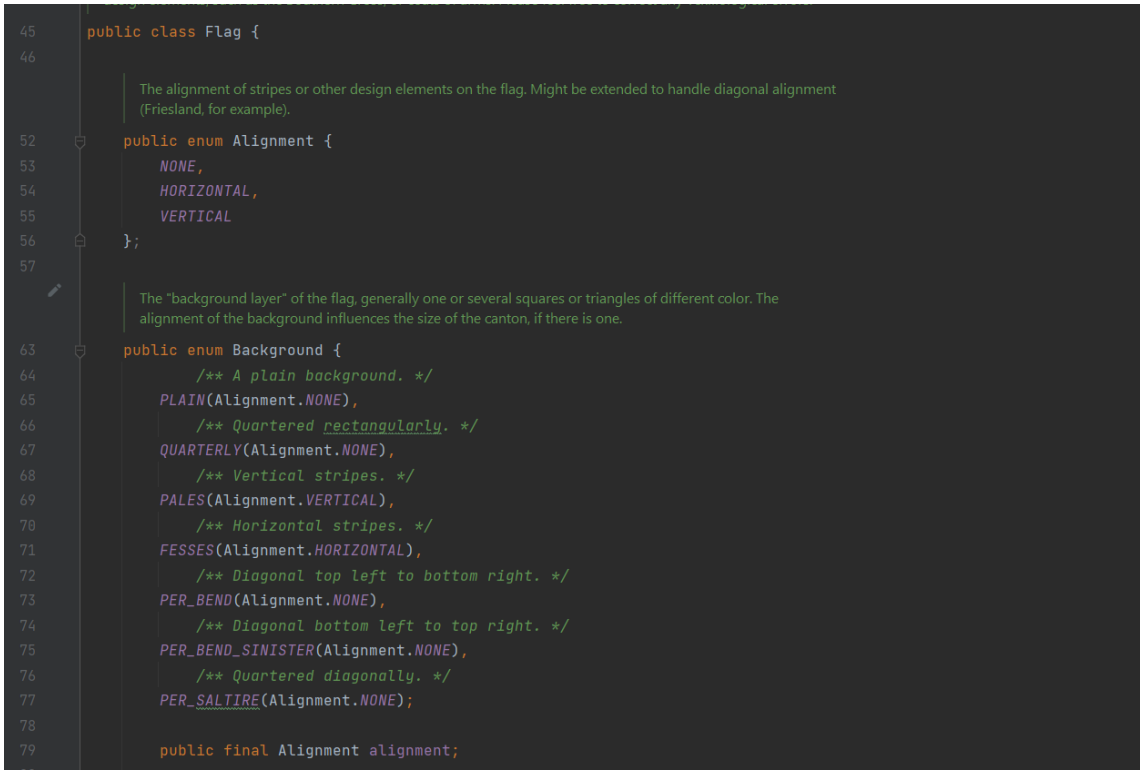
Code smells

Large Class

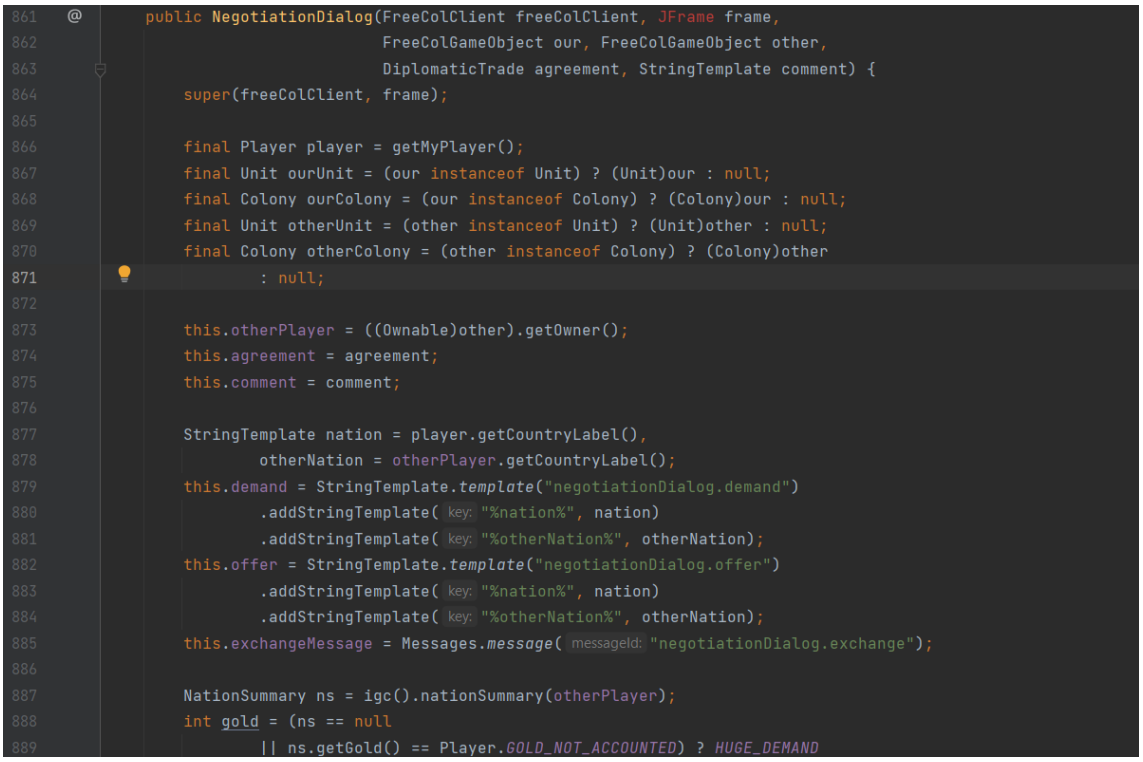


```
45  public class Flag {
46
        The alignment of stripes or other design elements on the flag. Might be extended to handle diagonal alignment
        (Friesland, for example).
52      public enum Alignment {
53          NONE,
54          HORIZONTAL,
55          VERTICAL
56      };
57
        The "background layer" of the flag, generally one or several squares or triangles of different color. The
        alignment of the background influences the size of the canton, if there is one.
63      public enum Background {
64              /** A plain background. */
65          PLAIN(Alignment.NONE),
66              /** Quartered rectangularly. */
67          QUARTERLY(Alignment.NONE),
68              /** Vertical stripes. */
69          PALES(Alignment.VERTICAL),
70              /** Horizontal stripes. */
71          FESSES(Alignment.HORIZONTAL),
72              /** Diagonal top left to bottom right. */
73          PER_BEND(Alignment.NONE),
74              /** Diagonal bottom left to top right. */
75          PER_BEND_SINISTER(Alignment.NONE),
76              /** Quartered diagonally. */
77          PER_SALTIRE(Alignment.NONE);
78
79          public final Alignment alignment;
```

Path: net/sf/freecol/client/gui/dialog/Flag.java

Aqui podemos ver a classe “Flag” com mais de 1200 linhas de código, o que é um exemplo do "large class code smell," um tipo de problema de código que se encaixa no grupo de "code smells" chamados de "bloaters." Este problema acontece quando uma única classe assume um grande número de responsabilidades. Para tentarmos resolver este problema, podemos subdividir a classe Flag em mais classes menores, cada uma encarregada de diferentes funcionalidades que estão todas presentes nesta mesma classe. Por exemplo poderíamos separar em diferentes classes associadas à classe Flag o calculo das diferentes figuras necessárias para desenhar uma “bandeira”.

Long Method



```
861  @    public NegotiationDialog(FreeColClient freeColClient, JFrame frame,
862                                FreeColGameObject our, FreeColGameObject other,
863                                DiplomaticTrade agreement, StringTemplate comment) {
864           super(freeColClient, frame);
865
866           final Player player = getMyPlayer();
867           final Unit ourUnit = (our instanceof Unit) ? (Unit)our : null;
868           final Colony ourColony = (our instanceof Colony) ? (Colony)our : null;
869           final Unit otherUnit = (other instanceof Unit) ? (Unit)other : null;
870           final Colony otherColony = (other instanceof Colony) ? (Colony)other
871                   : null;
872
873           this.otherPlayer = ((Ownable)other).getOwner();
874           this.agreement = agreement;
875           this.comment = comment;
876
877           StringTemplate nation = player.getCountryLabel(),
878                   otherNation = otherPlayer.getCountryLabel();
879           this.demand = StringTemplate.template("negotiationDialog.demand")
880                   .addStringTemplate( key: "%nation%", nation)
881                   .addStringTemplate( key: "%otherNation%", otherNation);
882           this.offer = StringTemplate.template("negotiationDialog.offer")
883                   .addStringTemplate( key: "%nation%", nation)
884                   .addStringTemplate( key: "%otherNation%", otherNation);
885           this.exchangeMessage = Messages.message( messageId: "negotiationDialog.exchange");
886
887           NationSummary ns = igc().nationSummary(otherPlayer);
888           int gold = (ns == null
889                   || ns.getGold() == Player.GOLD_NOT_ACCOUNTED) ? HUGE_DEMAND
```

Path: net/sf/freecol/client/gui/dialog/NegotiationDialog.java

Este método com quase duzentas linhas é um exemplo evidente do "code smell" denominado "Long Method." Este "code smell" aponta para uma extensão excessiva e a complexidade de um método, tornando-o difícil de compreender, modificar e fazer “debug”. A solução envolve a refatoração do código, ou seja, dividir este método em métodos menores, cada um com uma função específica e um nome descritivo.

Message Chain

Path: net/sf/freecol/client/gui/dialog/VictoryDialog.java

Line 51-53 : panel.add(Utility.localizedHeader(Messages.message("victory.text"), Utility.FONTSPEC_TITLE), "align center, wrap 20");

Como podemos ver no excerto de código acima encontramos um “code smell” da categoria “Couplers” chamado de “Message Chain”. Este code smell ocorre quando existem várias chamadas de métodos encadeados numa só linha de código. Esta cadeia de mensagens indica uma forte dependência entre objetos tornando o código menos legível e frágil. Uma forma de resolver este “code smell” seria guardar o resultado intermediário em variáveis locais dividindo assim as chamadas em varias linhas evitando assim este encadeamento de métodos numa só linha.