

# Code Smell and Refactoring

## Freecol Server

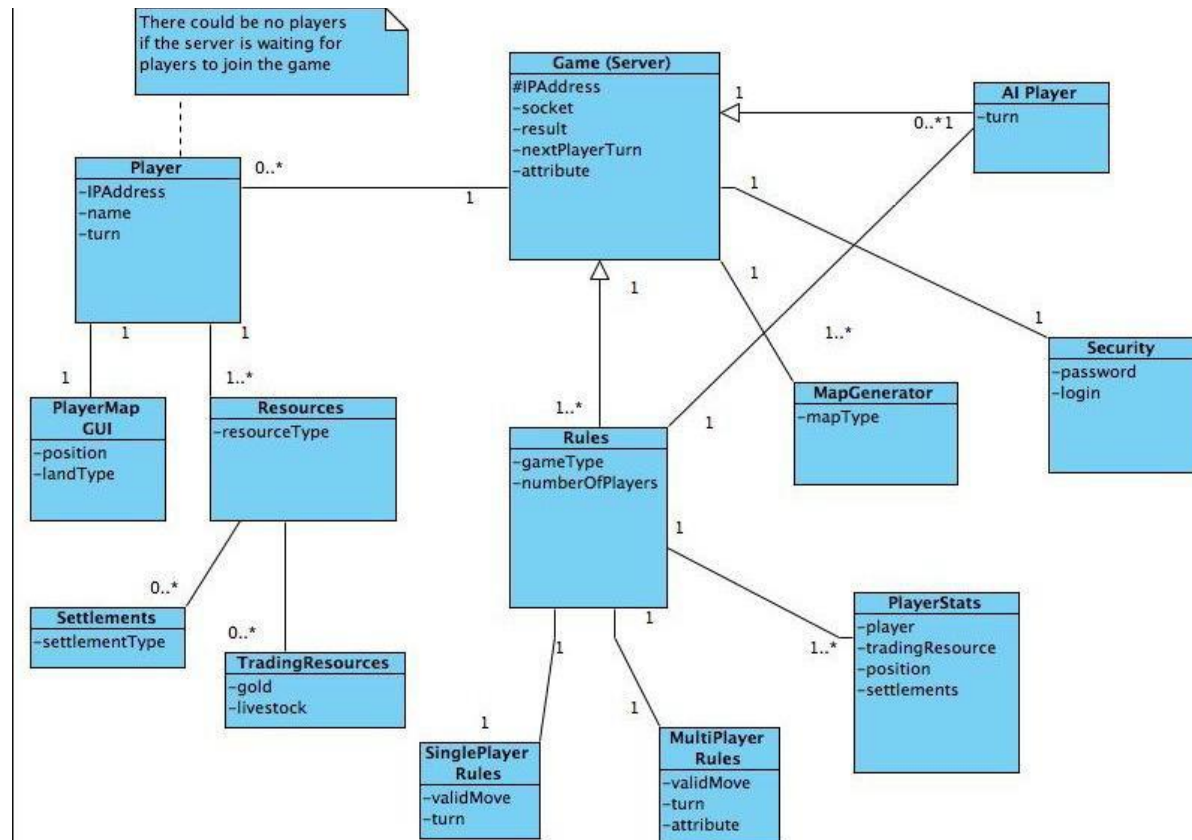
### SOEN 6471



**By:**

*Ronak Patel [6483607]  
Vishal Mittal [6425240]  
Hardev Goraya [6446019]  
Navrang Singh [6447430]*

## FreeCol server Conceptual Diagram:

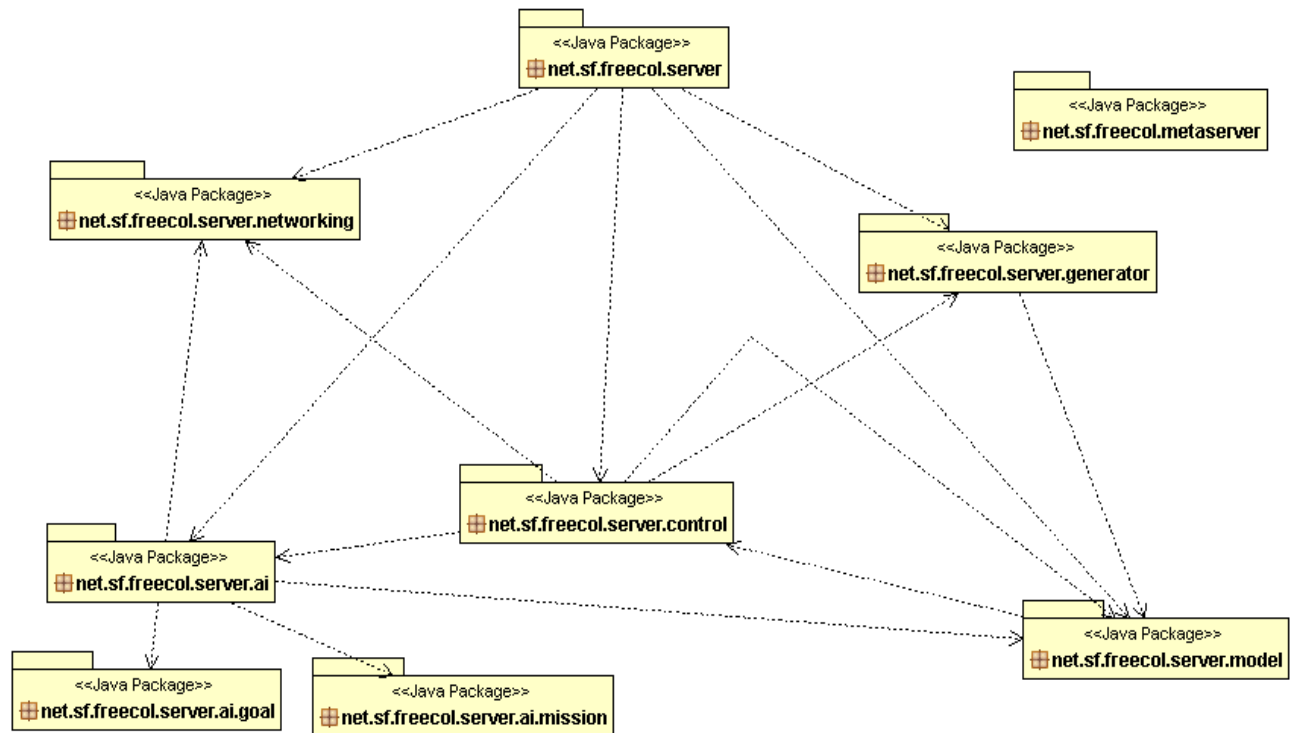


The Domain Diagram and conceptual architecture for our initial design of FreeCol was based on the client-server model. The reason for this was specifically to support multiplayer games that were to be played over the internet. The communication of the Player and the server was limited to a single control flow between the player and the server (the game), and knowledge was either contained on the server side or the client side. The designers of FreeCol however chose a different approach and incorporated an additional layer called Common.

The Common layer contains aspects of the game that are shared by both the client and the server. It doesn't contain details like GUI but the model of the game common to both client and server. The model could be described as the conceptual or imaginary aspects of the game. For example the concept of "building". A building has a type and a builder, these aspects award the player different scores depending on different aspects of the building. Both Server and Client need this information in order to play the game. On the other hand the Server needs no information about the GUI since it is the player who needs a visual representation of the game. Creating the Common model for both Client and Server obviously leads to much less code repetition, since both Client and Server utilize the same classes for common features of the game, this in turn reduces the size of the Server and Client side Classes. FreeCol is however a large program and the rest of this document will focus on

the Server Side aspects of the game. It is important however to get an overview of the whole game to see how the game runs.

### Package Diagram of server



The Server packages of FreeCol that this project focuses on are shown in the Package Diagram. We will not discuss the AI Packages, but focus on the Model, Control, Networking and Generator Packages.

The Control Package contains the classes responsible for the control of the game. There are Nine classes each responsible for different aspects of the controller. In the Conceptual Domain Model we assume there is a controller that controls the whole server side activity. These activities include making connections, checking all networking activities and validating the rules of game. In the conceptual model, validating Rules was by defined by the Rule Class but in FreeCol this work is done by the Controller class in the Server Control Package. The primary tasks of the controller are handling network messages, making changes to the internal model and communicating these messages to clients. Handling network messages before and during game play. The primary classes been the Controller, the InGameController and ChangeSet. ChangeSet is responsible for updating the Client side model.

The Model Package contains the model classes but with server specific information. As Mentioned above the model of the game is the conceptual world that makes up the game. The Model Classes are divided into Session classes, Building and Colony classes as well as Settlement and Player Classes.

The ServerGame Class contains the Server's representation of the game. Unlike the client side, the server side doesn't contain the GUI details. The ServerPlayer Class represents a Player but with additional server specific information and points to the Players Connections and Sockets. The ServerModelObject is the interface for server-side objects. It stores extra information in order to save a game and details the conceptual aspects of the game such as Building, Colony and Tile. The turns of the games in the model are described as transaction sessions and involve transactions between players such as diplomacy and trade. These classes extend the TransactionSession Interface.

The Domain model describes resources which are essential to add information to the client side of the game. The actual class diagram contains more details through the server side via "ServerPlayer" class and the "ServerModelObject" interface. The ServerPlayer contains control over most the classes for storing additional server specific information about the players of the game.

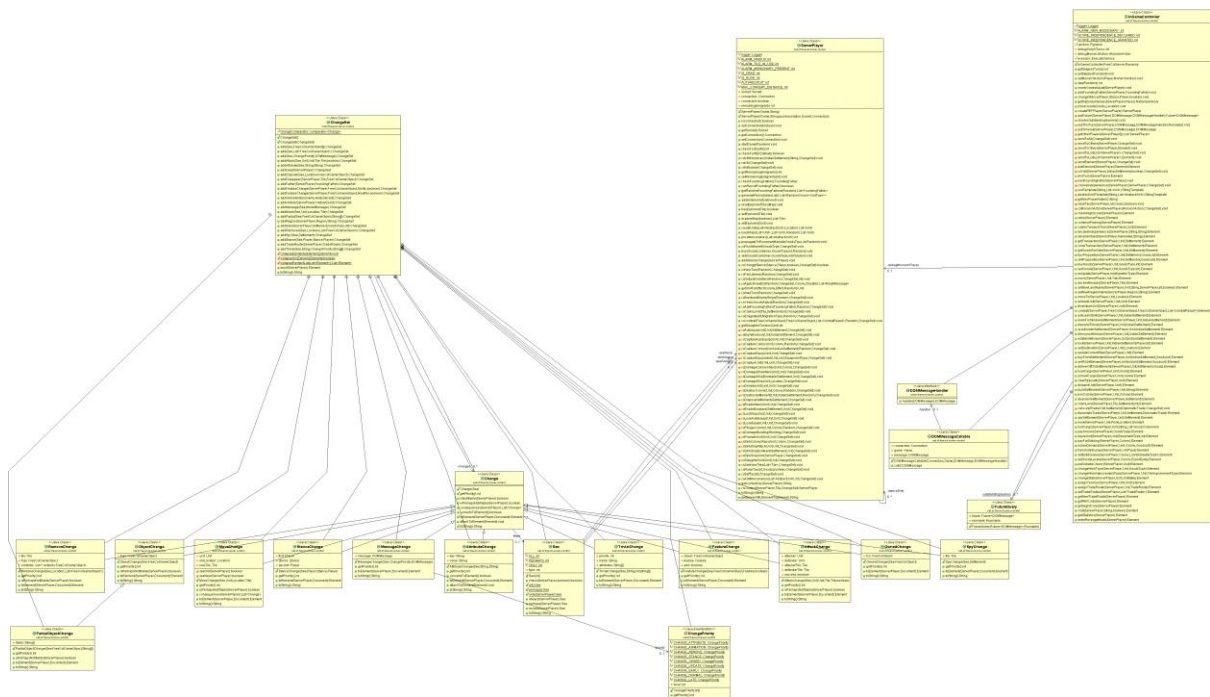
The Generator Package contains the classes that create maps and sets the starting locations for the players. There are three interfaces. The MapGenerator which creates maps and sets the starting locations for the players. The MapLayerGenerator which creates a map layer, consisting of Land, River and other types of terrain and the MapLoader which loads the map into a given game. The SimpleMapGenerator creates random maps and sets the starting locations for the players.

The MapGenerator class as described in the domain diagram is quite similar to the actual class diagram. However, it contains some specific details which are not mentioned in the domain model. The MapLayerGenerator and the MapLoader are important classes which are not mentioned in the domain diagram.

The Networking Package contains the server networking classes. The main server networking class is the Server Class which is where players make their initial connection, and server related functionality like Broadcast are implemented. As well as starting and shutting down the server. The DummyConnection Class is used to connect AI players. This package is small compared to the others and is an implementation of a skeleton Server with little functionality. The networking package, in the game contains separate functionality for establishing a connection between AI players and the client through the networking class module. This was done through the main Server (Game) in the domain diagram.

## Refactoring the server Package in the FreeCol game:

In the Freecol the server package having bunch of classes and all those classes having many code smells like god class, long method and type checking for example ChangeSet in the server.control package, inGameController class in the same package. And ServerUnit class in the server.model package respectively.



As seen in the diagram there is three big classes changeSet,ServerPlayer and InGamecontroller.Change set class and ServerPlayer class is very huge class having 1671 and 3750 number of line of code. It use large number of methods and lot of IF and switch Statements. Moreover in the change set there is 15 numbers of inner classes and all of these are public static. And the dependency between these classes is too much these all classes using a number of variable from their parent classes .And also it import an ArrayList that also make code very complicated.

Import java.util.Arraylist;

And after that it makes an object of array list called changes its attribute is private that make much dependency.

```

19
20 package net.sf.freecol.server.control;
21
22+ import java.util.ArrayList;
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57+ * Changes to be sent to the client.
58
59 public class ChangeSet {
60
61     // Convenient way to specify the relative priorities of the fixed
62     // change types in one place.
63+ public static enum ChangePriority {
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86     private ArrayList<Change> changes;
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106

```

Furthermore ServerPlayer is a Player with Additional (server specific) information and incuses some fields i.e. Socket, connection, Boolean, List<serverplayer> which can be moved to another separate class in order to split the responsibility of the class.

```

107
108     // How far to search for a colony to add an Indian convert to.
109     public static final int MAX_CONVERT_DISTANCE = 10;
110
111
112     /** The network socket to the player's client. */
113     private Socket socket;
114
115     /** The connection for this player. */
116     private Connection connection;
117
118     private boolean connected = false;
119
120     /** Remaining emigrants to select due to a fountain of youth */
121     private int remainingEmigrants = 0;
122
123     /** Players with respect to which stance has changed. */
124     private List<ServerPlayer> stanceDirty = new ArrayList<ServerPlayer>();
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

By using extract class refactoring we can reduce some dependency between the inner classes in the changeSet classes and ServerPlayer class to make the code much readable and maintainable.

### **Refactoring the changeSet class(GOD Class):**

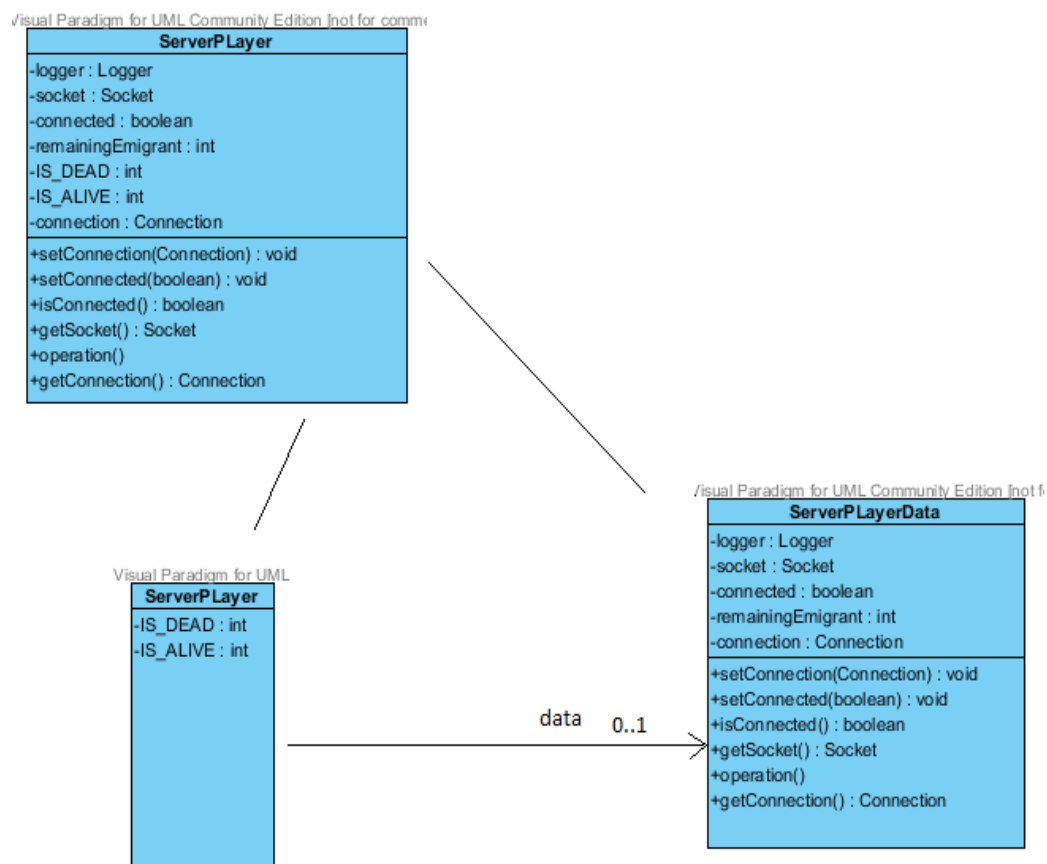
In this case we can separate "Array\_List" behaviour into its own nested class by inserting the field class in the original class to limit the responsibly to the original class. Secondly we can remove the Fields i.e. ArrayList<change> and replace by the link from old class to new class in order to make connect among them. Lastly replace a set of fields (write accesses and read accesses) with new container object. All references to the fields are updated to access the new container object.

## Refactoring the ServerPlayer(GOD Class):

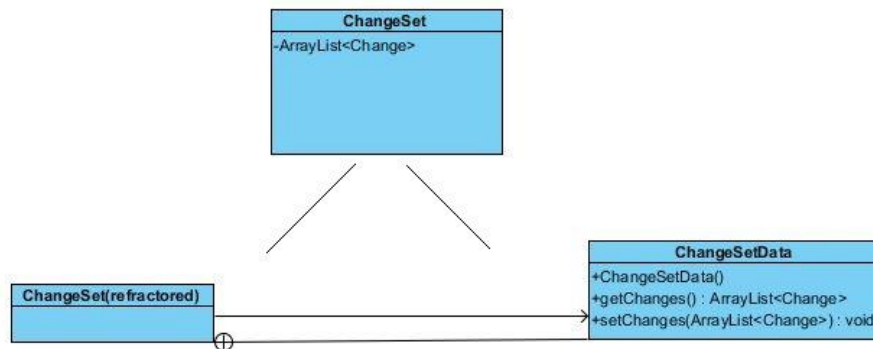
As we already know the ServerPlayer is the huge class having lot of dependency between them. To make it less coupled we extract some of the members to put into another class and then just call from the ServerPlayer class.

we can extract the body of these fields into the another class called ServerPlayerProduct in the server.model.package define their getters setters and instead of doing all these stuff in one class we just call the method and return the value to the ServerPlayer method class.

### ServerPlayer.java



## ChangeSet.java



## Refactoring for the demand tribute (long Method)

The *InGameController* class is inherently too big with more than 3000 lines of code. Moreover, it can be observed that the method named “*demandTribute*” is large enough. The *demandTribute* method has a *ServerPlayer* and *unit* that is demanding the tribute from *IndianSettlement*. Hence, we have *ServerPlayer*, *unit* and *IndianSettlement* as method parameters. We will extract the method *ModelMessage* into a new method *m*. The new method will have parameters such as *unit*, *settlement*, *gold* and the year when the settlement is made. So, we will create a new private method *m* and call this method to access its properties. The additional gold attribute added in the method is also demanded as tribute. In this way, we can achieve code comprehension by extracting a long method and the code is indeed maintainable due to this. A snapshot for the code is given below:

```
InGameController.java
Original Source
2082         Tension.TENSION_ADD_NORMAL));
2083         settlement.setLastTribute(year);
2084         ModelMessage m;
2085         if (gold > 0) {
2086             indianPlayer.modifyGold(-gold);
2087             serverPlayer.modifyGold(gold);
2088             cs.addPartial(See.only(serverPlayer), serverPlayer, "gold", "score");
2089             m = new ModelMessage(ModelMessage.MessageType.FOREIGN_DIPLOMACY,
2090                                 "scoutSettlement.tributeAgree",
2091                                 unit, settlement)
2092                 .addAmount("%amount%", gold);
2093         } else {
2094             m = new ModelMessage(ModelMessage.MessageType.FOREIGN_DIPLOMACY,
2095                                 "scoutSettlement.tributeDisagree",
2096                                 unit, settlement);
2097         }
2098         cs.addMessage(See.only(serverPlayer), m);
2099         Tile tile = settlement.getTile();
2100         tile.updatePlayerExploredTile(serverPlayer, true);
2101         cs.add(See.only(serverPlayer), tile);
2102         unit.setMovesLeft(0);
2103         cs.addPartial(See.only(serverPlayer), unit, "movesLeft");
2104
2105         // Do not update others, this is all private.
```



## Refactoring of the Ship Method (long method) in ServerPlayer class:

The *csPillageColony* method in the *ServerPlayer* class has a code smell of long method. The method works in order to damage the building or ship or steal some resources such as any goods or gold. It has method parameters such as unit, colony, random and changeset. Every unit is owned by a player and it has a location. The colony is the colony which will be pillaged by the attacker. The method consists of almost 90 lines of code. Hence, we will extract a new ship method and pass these parameters in this method. In this way, the damaged or illegally obtained ships will be dealt by this method. A snippet of the code is given below:

```
3248 private void csPillageColony(Unit attacker, Colony colony,
3249                               Random random, ChangeSet cs) {
3250     ServerPlayer attackerPlayer = (ServerPlayer) attacker.getOwner();
3251     StringTemplate attackerNation = attacker.getApparentOwnerName();
3252     ServerPlayer colonyPlayer = (ServerPlayer) colony.getOwner();
3253     StringTemplate colonyNation = colonyPlayer.getNationName();
3254
3255     // Collect the damagable buildings, ships, movable goods.
3256     List<Building> buildingList = colony.getBurnableBuildingList();
3257     List<Unit> shipList = colony.getShipList();
3258     List<Goods> goodsList = colony.getLootableGoodsList();
3259
3260     // Pick one, with one extra choice for stealing gold.
3261     int pillage = Utils.randomInt(logger, "Pillage choice", random,
3262                                  buildingList.size() + shipList.size() + goodsList.size()
3263                                  + ((colony.canBePlundered()) ? 1 : 0));
3264     if (pillage < buildingList.size()) {
3265         Building building = buildingList.get(pillage);
3266         csDamageBuilding(building, cs);
3267         cs.addMessage(See.only(colonyPlayer),
3268                      new ModelMessage(ModelMessage.MessageType.COMBAT_RESULT,
3269                                       "model.unit.buildingDamaged", colony)
3270                      .add("%building%", building.getNameKey())
3271                      .addName("%colony%", colony.getName()));
3272     }
```