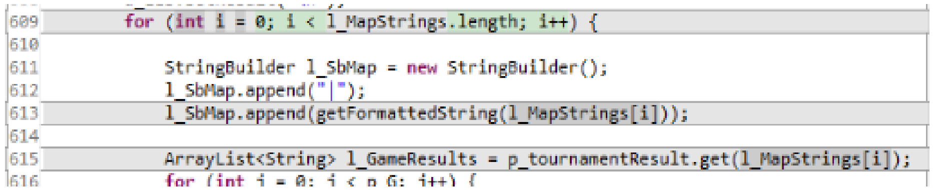
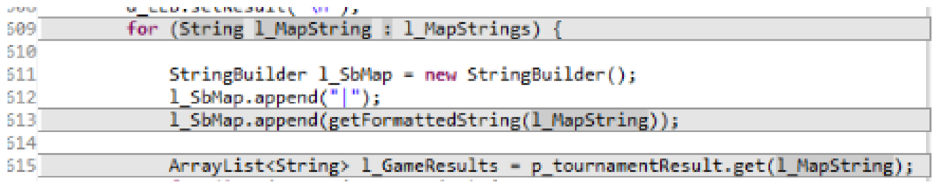
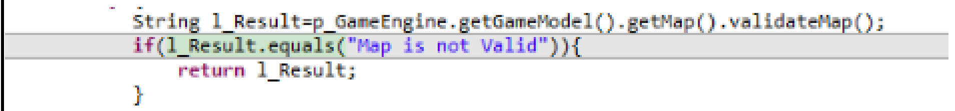
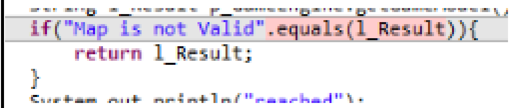
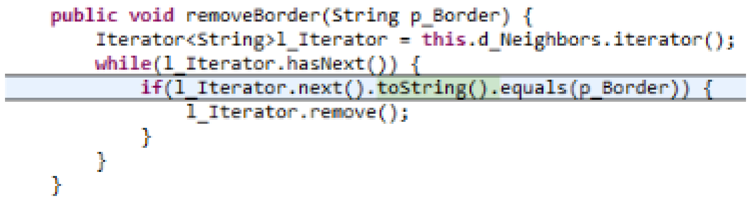
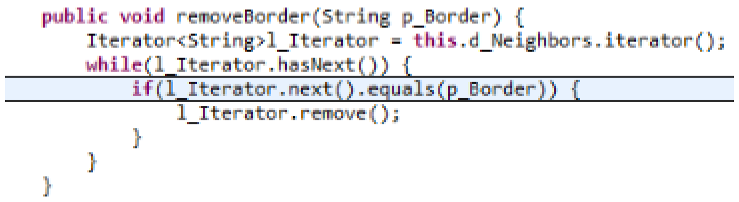
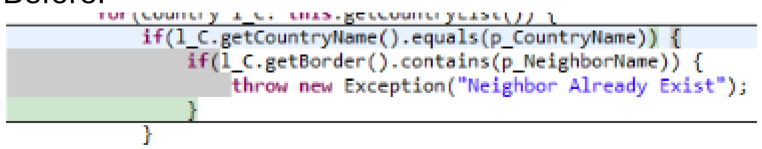
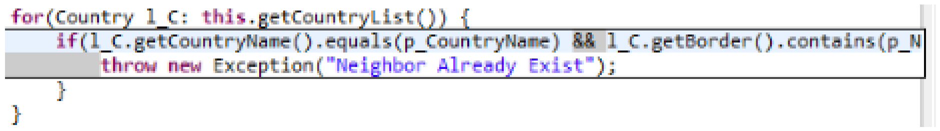
REFACTORING DOCUMENT

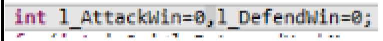
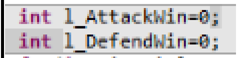
Refactoring Type	Refactoring Class	Explanation	Screenshot

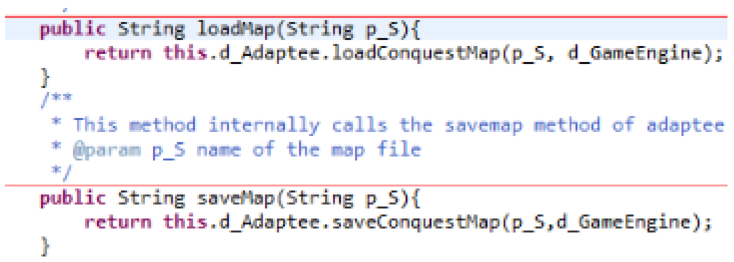
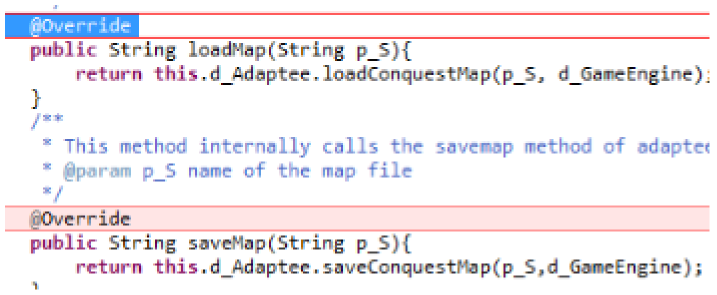
1. Replace for loop with enhanced for-loop	GameEngine	Replacing old for-loops utilizing iterators with enhanced for-loops in order to improve readability	Before:
			
			`for (int i = 0; i < l_MapStrings.length; i++) {`
			
			`StringBuilder l_SbMap = new StringBuilder();`
			`l_SbMap.append("|");`
			`l_SbMap.append(getFormattedString(l_MapStrings[i]));`
			
			`ArrayList<String> l_GameResults = p_tournamentResult.get(l_MapStrings[i]);`
			`for (int j = 0; j < p_G; j++) {`
			After:
			
			`for (String l_MapString : l_MapStrings) {`
			
			`StringBuilder l_SbMap = new StringBuilder();`
			`l_SbMap.append("|");`
			`l_SbMap.append(getFormattedString(l_MapString));`
			
			`ArrayList<String> l_GameResults = p_tournamentResult.get(l_MapString);`

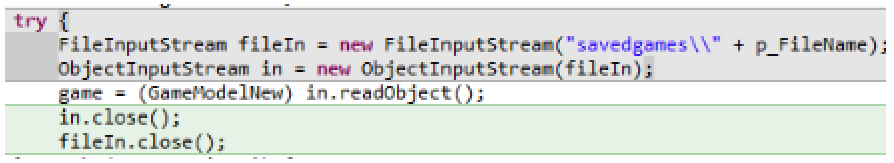
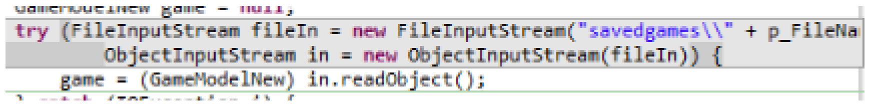
2. Reorder String Equality check	Order files	To avoid NullPointerExceptions , it is recommended to put string literals in the left-hand-side of equals() or equalsIgnoreCase() when checking for equality.	Before:
			
			`String l_Result=p_GameEngine.getGameModel().getMap().validateMap();`
			`if(l_Result.equals("Map is not Valid")){`
			`    return l_Result;`
			`}`
			After:
			
			`if("Map is not Valid".equals(l_Result)){`
			`    return l_Result;`
			`}`
			`System.out.println("reached");`

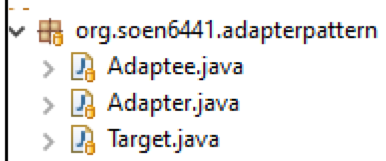
3. Remove toString() on String	Country.java, Map.java, Reinforcemet.java, Startup.java	All method invocations of toString() on a String element are not needed.	Before:
			
			`public void removeBorder(String p_Border) {`
			`    Iterator<String>l_Iterator = this.d_Neighbors.iterator();`
			`    while(l_Iterator.hasNext()) {`
			`        if(l_Iterator.next().toString().equals(p_Border)) {`
			`            l_Iterator.remove();`
			`        }`
			`    }`
			`}`
			After:
			
			`public void removeBorder(String p_Border) {`
			`    Iterator<String>l_Iterator = this.d_Neighbors.iterator();`
			`    while(l_Iterator.hasNext()) {`
			`        if(l_Iterator.next().equals(p_Border)) {`
			`            l_Iterator.remove();`
			`        }`
			`    }`
			`}`

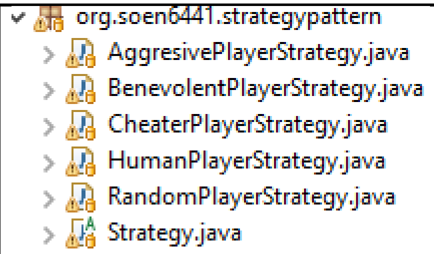
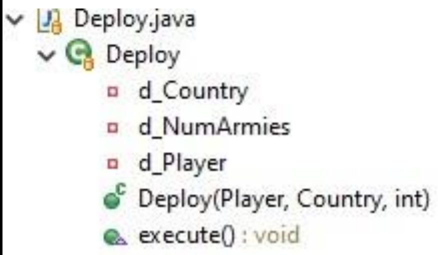
4. Collapse if statement	Map.java, PlayerController.java	Collapses nested if-statements into a single one when possible.	Before:
			
			`for(Country l_C: this.getCountryList()) {`
			`    if(l_C.getCountryName().equals(p_CountryName)) {`
			`        if(l_C.getBorder().contains(p_NeighborName)) {`
			`            throw new Exception("Neighbor Already Exist");`
			`        }`
			`    }`
			After:
			
			`for(Country l_C: this.getCountryList()) {`
			`    if(l_C.getCountryName().equals(p_CountryName) && l_C.getBorder().contains(p_N`
			`        throw new Exception("Neighbor Already Exist");`
			`    }`
			`}`

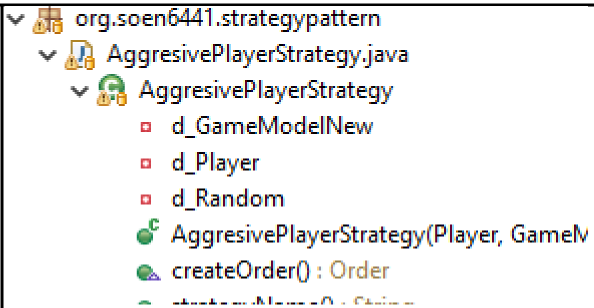
5. Split multiple variable declarations	Order files	It will split declarations occurring on the same line over multiple lines to improve readability.	Before:
			
			`int l_AttackWin=0,l_DefendWin=0;`
			After:
			
			`int l_AttackWin=0;`
			`int l_DefendWin=0;`

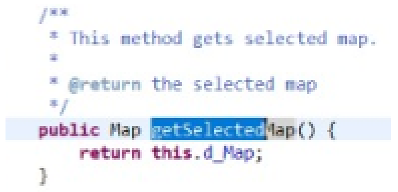
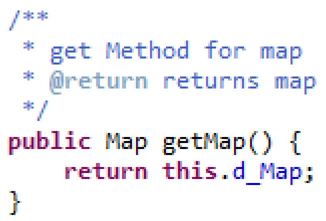
6. Use @Override Annotations	Advance.java	Adding override annotations to methods overriding or implementing another method declared in a parent class.	Before:
			
			`public String loadMap(String p_S){`
			`    return this.d_Adaptee.loadConquestMap(p_S, d_GameEngine);`
			`}`
			`/**`
			` * This method internally calls the savemap method of adaptee`
			` * @param p_S name of the map file`
			` */`
			`public String saveMap(String p_S){`
			`    return this.d_Adaptee.saveConquestMap(p_S,d_GameEngine);`
			`}`
			After:
			
			`@Override`
			`public String loadMap(String p_S){`
			`    return this.d_Adaptee.loadConquestMap(p_S, d_GameEngine);`
			`}`
			`/**`
			` * This method internally calls the savemap method of adaptee`
			` * @param p_S name of the map file`
			` */`
			`@Override`
			`public String saveMap(String p_S){`
			`    return this.d_Adaptee.saveConquestMap(p_S,d_GameEngine);`

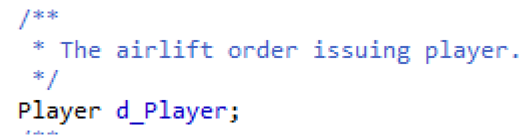
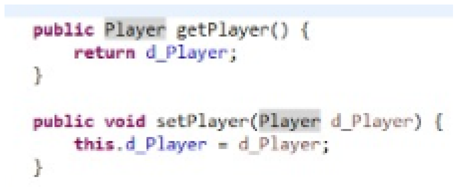
7. Use try-with-resource	GameModelNew	Closing statements are removed as the construct takes of that to make code safer and more readable.	Before:  `try {`
			`    FileInputStream fileIn = new FileInputStream("savedgames\\" + p_FileName);`
			`    ObjectInputStream in = new ObjectInputStream(fileIn);`
			`    game = (GameModelNew) in.readObject();`
			`    in.close();`
			`    fileIn.close();`
			After:  `try (FileInputStream fileIn = new FileInputStream("savedgames\\" + p_FileNa`
			`        ObjectInputStream in = new ObjectInputStream(fileIn)) {`
			`    game = (GameModelNew) in.readObject();`

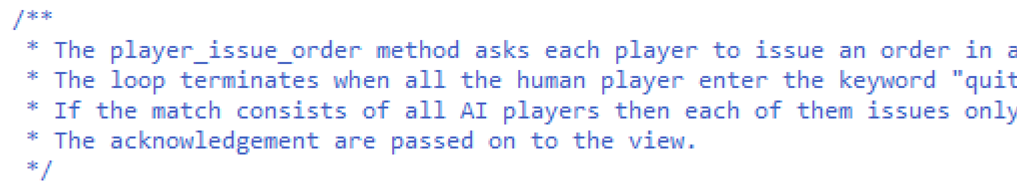
8. Adapter Pattern	Main Folder	Adapter pattern to enable the application to read/write from/to a file using the “conquest” game map format	 org.soen6441.adapterpattern Adaptee.java Adapter.java Target.java

9. Strategy pattern	Main folder	During the main development phase, implement different computer player behaviors using the Strategy pattern, where the strategies provide varying behavior that support the Player class to expose varying behavior when executing the issueOrders() method.	 org.soen6441.strategypattern AggresivePlayerStrategy.java BenevolentPlayerStrategy.java CheaterPlayerStrategy.java HumanPlayerStrategy.java RandomPlayerStrategy.java Strategy.java
10. Push Down Method	Strategy Pattern	Pushing down the create order from order class to strategy class	Before:  Deploy.java Deploy d_Country d_NumArmies d_Player Deploy(Player, Country, int) execute() : void After:

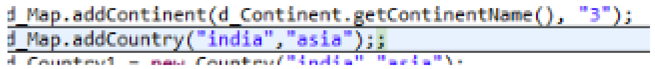
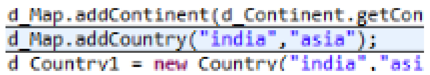
			 org.soen6441.strategypattern AggresivePlayerStrategy.java AggresivePlayerStrategy d_GameModelNew d_Player d_Random AggresivePlayerStrategy(Player, GameM createOrder() : Order

11. Remove method	Gamemodelnew	Two methods which were doing the same work	Before:
			
			`/**`
			`* This method gets selected map.`
			`*`
			`* @return the selected map`
			`*/`
			`public Map getSelectedMap() {`
			`    return this.d_Map;`
			`}`
			
			After:
			
			`/**`
			`* get Method for map`
			`* @return returns map`
			`*/`
			`public Map getMap() {`
			`    return this.d_Map;`
			`}`

12. Hide field	Airlift.java	The data member is made private and adding getter setter method	Before:
			
			`/**`
			` * The airlift order issuing player.`
			` */`
			`Player d_Player;`
			After:
			
			`public Player getPlayer() {`
			`    return d_Player;`
			`}`
			
			`public void setPlayer(Player d_Player) {`
			`    this.d_Player = d_Player;`
			`}`

13. Substitute Algorithm	PlayerController.java	Terminating condition for player issue order	
			`/**`
			` * The player_issue_order method asks each player to issue an order in a`
			` * The loop terminates when all the human player enter the keyword "quit`
			` * If the match consists of all AI players then each of them issues only`
			` * The acknowledgement are passed on to the view.`
			` */`

14. Organize imports	All classes	Removing unused imports declarations	

15. Remove unnecessary semicolons	GameEngine	Removing unnecessary semicolons	Before:
			
			`d_Map.addContinent(d_Continent.getContinentName(), "3");`
			`d_Map.addCountry("india","asia");;`
			`d_Country1 = new Country("india","asia");`
			After:
			
			`d_Map.addContinent(d_Continent.getCon`
			`d_Map.addCountry("india","asia");`
			`d Country1 = new Country("india","asi`