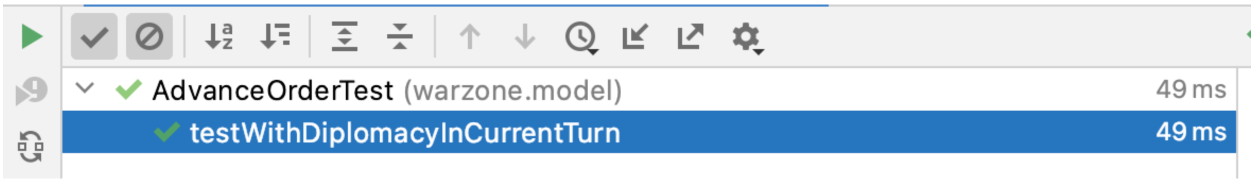
The refactoring process of this build consisted mostly of code inspections and discussions among developers. Each developer was asked to sift through the code and mark down any potential refactoring changes that could be made. Below is a list of refactoring targets that we came up with:

1) Refactor the local d_gameContext in AdvanceOrder into retrieving from GameContext singleton
2) In startup service, loadmap, extract the implementation to a new file.
3) isGameEnded, keep the simple logic for the function.
4) Rename l_protentialWinner to l_potentialWinner in GameEngine.java
5) Rename conventDeployOrder to convertDeployOrder in Player.java. We could also remove this method since it is not used. Maybe @Deprecated?
6) Rename conventOrder to convertOrder in Player.java
7) Create a service class to create the various orders. Right now there are several methods to create orders in the Player class. This logic is not super relevant to this class.
8) Added adapters to read conquest map files.
9) Refactor RouterService.java. There are several switch statements that are a bit redundant and bulky. This could be updated to provide cleaner, more understandable code.
10) GameContext class is getting bulky. Some new files could be created to reduce the amount of class variables in GameContext. For example, a MapContext class could be created to house all map related data. GameContext could have a MapContext as a data member.
11) We have several enums with values that are not used. Some enum classes could be removed or updated to get rid of unused values.
12) We have several service classes that could be combined. Ex: ContinentService, CountryService, and MapService are all very similar and could be refactored into a single class.
13) Our command router passes ControllerName enum values. This could be refactored to not use enums and instead use inheritance/polymorphism to call the correct controller.
14) Our enum fields are not consistent. Some are all capital letters, while others have lower case letters. We could rename them to all use a consistent style. Ex: GamePhase.java
15) HelpView.java is hard-coded with applicable commands a user can enter. Perhaps we could generate this based on the current game phase so that it is dynamic.

This list is quite extensive, so we narrowed it down to the most important targets to actually implement. We did this during a team meeting in which we all agreed on which ones to start with. Our ranking was based on the usefulness/importance of the target and time and energy required. Here are five of the targets that we ended up implementing:

1) Refactor the local d_gameContext in AdvanceOrder into retrieving from GameContext singleton

Tests:

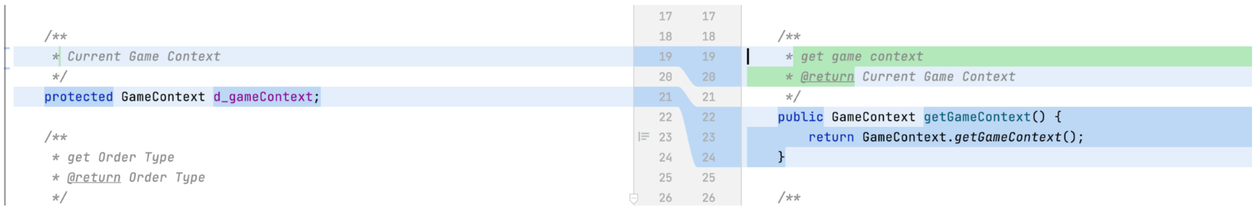


```
AdvanceOrderTest (warzone.model)                         49 ms
    testWithDiplomacyInCurrentTurn                       49 ms
```

Reasoning:

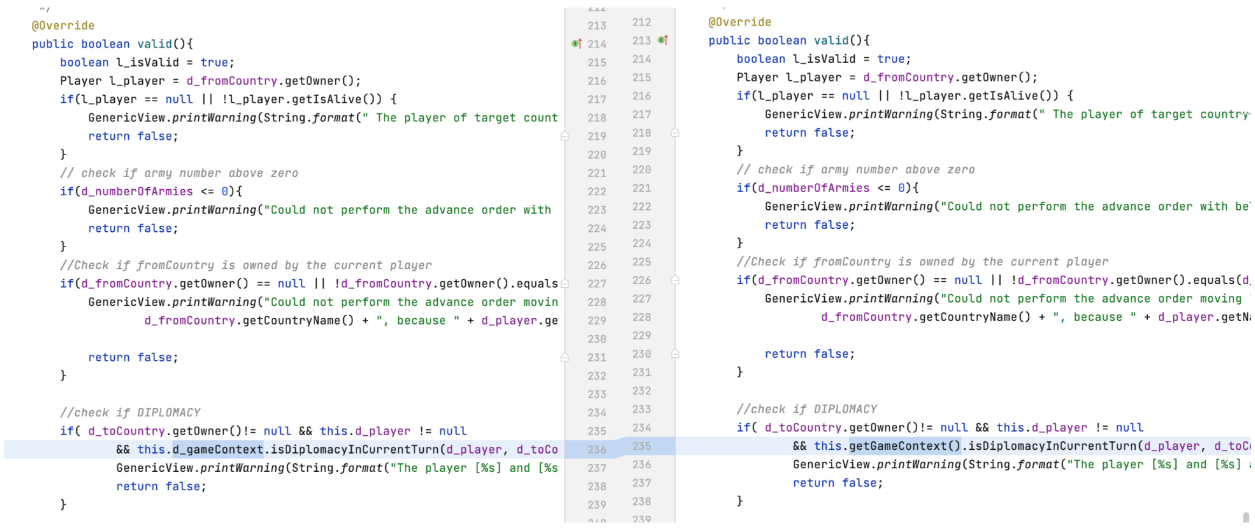
We use the singleton pattern for the GameContext in most places, so it made sense to update this occurrence. This ensures a consistent game context instance is used everywhere.

Before/After:



```
    /**
     * Current Game Context
     */
    protected GameContext d_gameContext;

    /**
     * get Order Type
     * @return Order Type
     */
```

```
    /**
     * get game context
     * @return Current Game Context
     */
    public GameContext getGameContext() {
        return GameContext.getGameContext();
    }

    /**
```



```
    @Override
    public boolean valid(){
        boolean l_isValid = true;
        Player l_player = d_fromCountry.getOwner();
        if(l_player == null || !l_player.getIsAlive()) {
            GenericView.printWarning(String.format(" The player of target count
            return false;
        }
        // check if army number above zero
        if(d_numberOfArmies <= 0){
            GenericView.printWarning("Could not perform the advance order with
            return false;
        }
        //Check if fromCountry is owned by the current player
        if(d_fromCountry.getOwner() == null || !d_fromCountry.getOwner().equals
            GenericView.printWarning("Could not perform the advance order movin
                    d_fromCountry.getCountryName() + ", because " + d_player.ge

            return false;
        }

        //check if DIPLOMACY
        if( d_toCountry.getOwner()!= null && this.d_player != null
                && this.d_gameContext.isDiplomacyInCurrentTurn(d_player, d_toCo
                GenericView.printWarning(String.format("The player [%s] and [%s
                return false;
        }
```

```
    @Override
    public boolean valid(){
        boolean l_isValid = true;
        Player l_player = d_fromCountry.getOwner();
        if(l_player == null || !l_player.getIsAlive()) {
            GenericView.printWarning(String.format(" The player of target country
            return false;
        }
        // check if army number above zero
        if(d_numberOfArmies <= 0){
            GenericView.printWarning("Could not perform the advance order with be
            return false;
        }
        //Check if fromCountry is owned by the current player
        if(d_fromCountry.getOwner() == null || !d_fromCountry.getOwner().equals(d
            GenericView.printWarning("Could not perform the advance order moving
                    d_fromCountry.getCountryName() + ", because " + d_player.getN

            return false;
        }

        //check if DIPLOMACY
        if( d_toCountry.getOwner()!= null && this.d_player != null
                && this.getGameContext().isDiplomacyInCurrentTurn(d_player, d_toC
                GenericView.printWarning(String.format("The player [%s] and [%s]
                return false;
        }
```

2) GameEngine.play() was renamed to

Tests: GameEngineServiceTest.java

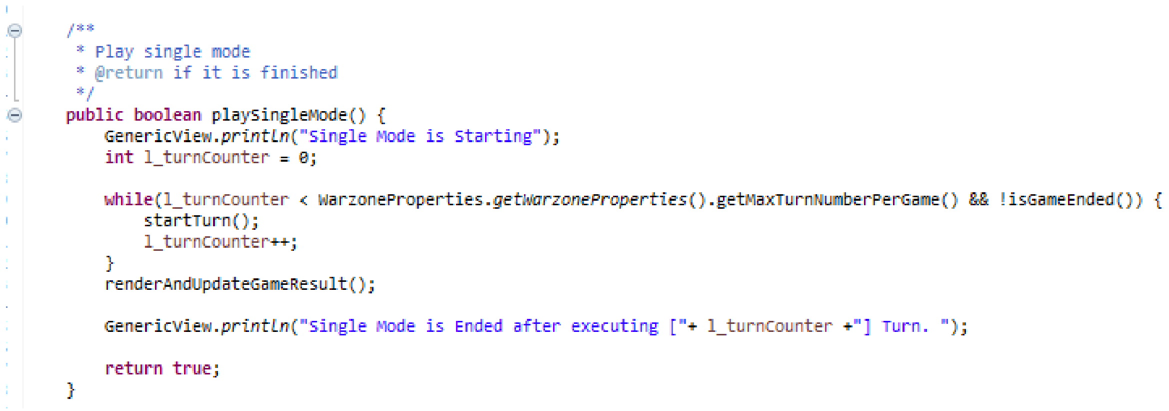
Reasoning:

The logic was changed a bit, but the renaming of the method is what we would like to highlight here. Since we needed to add tournament functionality this build, play() became ambiguous. playSingleMode() now clearly identifies what the method is doing.

Before:

```
/**
 * If the game turn is greater than 100, the game will end.
 *
 * @return true if the game can end.
 */
public boolean play() {

        if(! isReadyToStart())
                return false;
        if(this.d_gameContext.getIsDemoMode())
                startTurn();
        else {
                while(!isGameEnded())
                        startTurn();
        }
        return true;
}
```

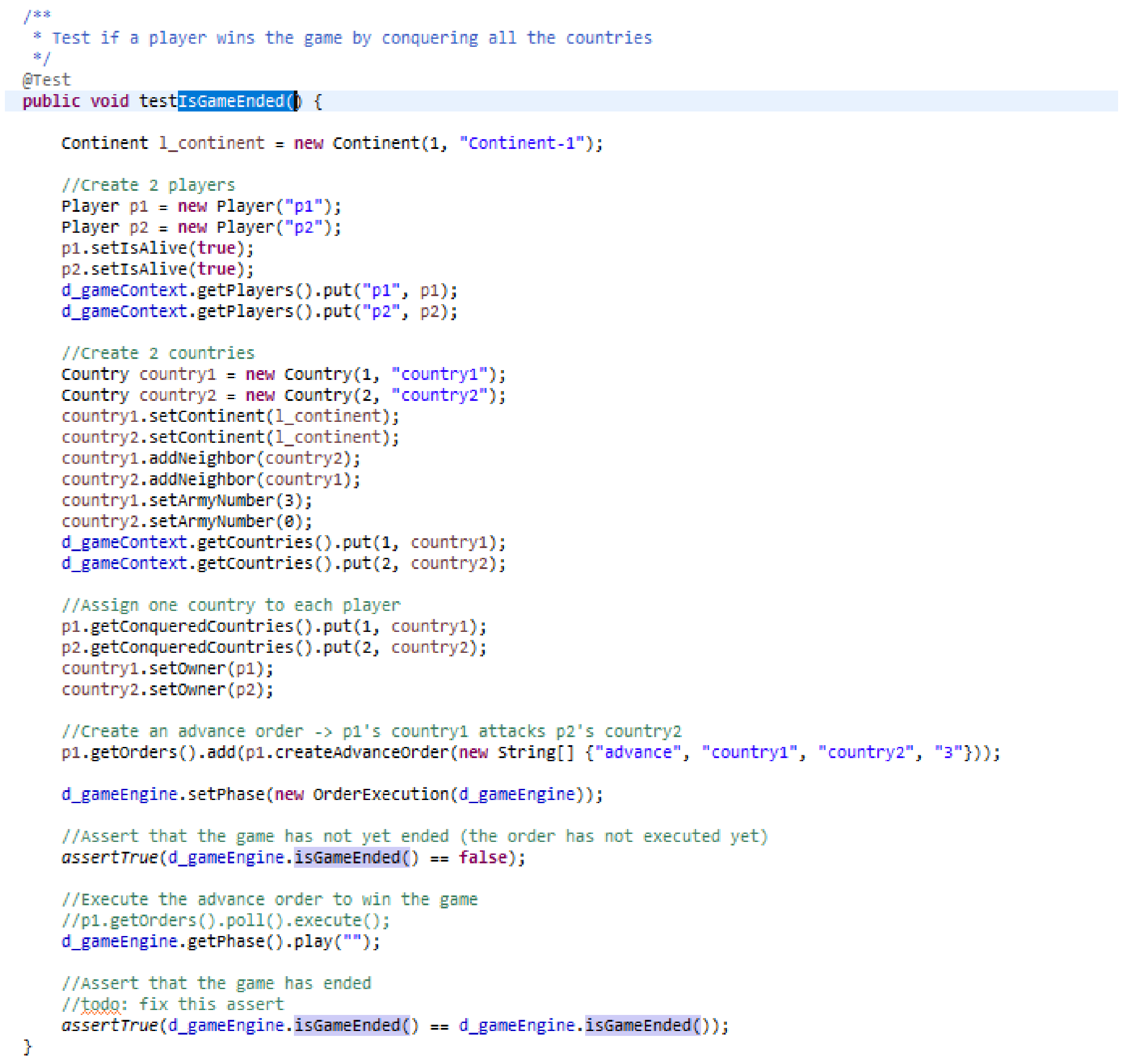
After:



```
/**
 * Play single mode
 * @return if it is finished
 */
public boolean playSingleMode() {
    GenericView.println("Single Mode is Starting");
    int l_turnCounter = 0;

    while(l_turnCounter < WarzoneProperties.getWarzoneProperties().getMaxTurnNumberPerGame() && !isGameEnded()) {
        startTurn();
        l_turnCounter++;
    }
    renderAndUpdateGameResult();

    GenericView.println("Single Mode is Ended after executing ["+ l_turnCounter +"] Turn. ");

    return true;
}
```

3) isGameEnded, keep the simple logic for the function.

Tests:



```java
/**
 * Test if a player wins the game by conquering all the countries
 */
@Test
public void testIsGameEnded() {

    Continent l_continent = new Continent(1, "Continent-1");

    //Create 2 players
    Player p1 = new Player("p1");
    Player p2 = new Player("p2");
    p1.setIsAlive(true);
    p2.setIsAlive(true);
    d_gameContext.getPlayers().put("p1", p1);
    d_gameContext.getPlayers().put("p2", p2);

    //Create 2 countries
    Country country1 = new Country(1, "country1");
    Country country2 = new Country(2, "country2");
    country1.setContinent(l_continent);
    country2.setContinent(l_continent);
    country1.addNeighbor(country2);
    country2.addNeighbor(country1);
    country1.setArmyNumber(3);
    country2.setArmyNumber(0);
    d_gameContext.getCountries().put(1, country1);
    d_gameContext.getCountries().put(2, country2);

    //Assign one country to each player
    p1.getConqueredCountries().put(1, country1);
    p2.getConqueredCountries().put(2, country2);
    country1.setOwner(p1);
    country2.setOwner(p2);

    //Create an advance order -> p1's country1 attacks p2's country2
    p1.getOrders().add(p1.createAdvanceOrder(new String[] {"advance", "country1", "country2", "3"}));

    d_gameEngine.setPhase(new OrderExecution(d_gameEngine));

    //Assert that the game has not yet ended (the order has not executed yet)
    assertTrue(d_gameEngine.isGameEnded() == false);

    //Execute the advance order to win the game
    //p1.getOrders().poll().execute();
    d_gameEngine.getPhase().play("");

    //Assert that the game has ended
    //todo: fix this assert
    assertTrue(d_gameEngine.isGameEnded() == d_gameEngine.isGameEnded());
}
```

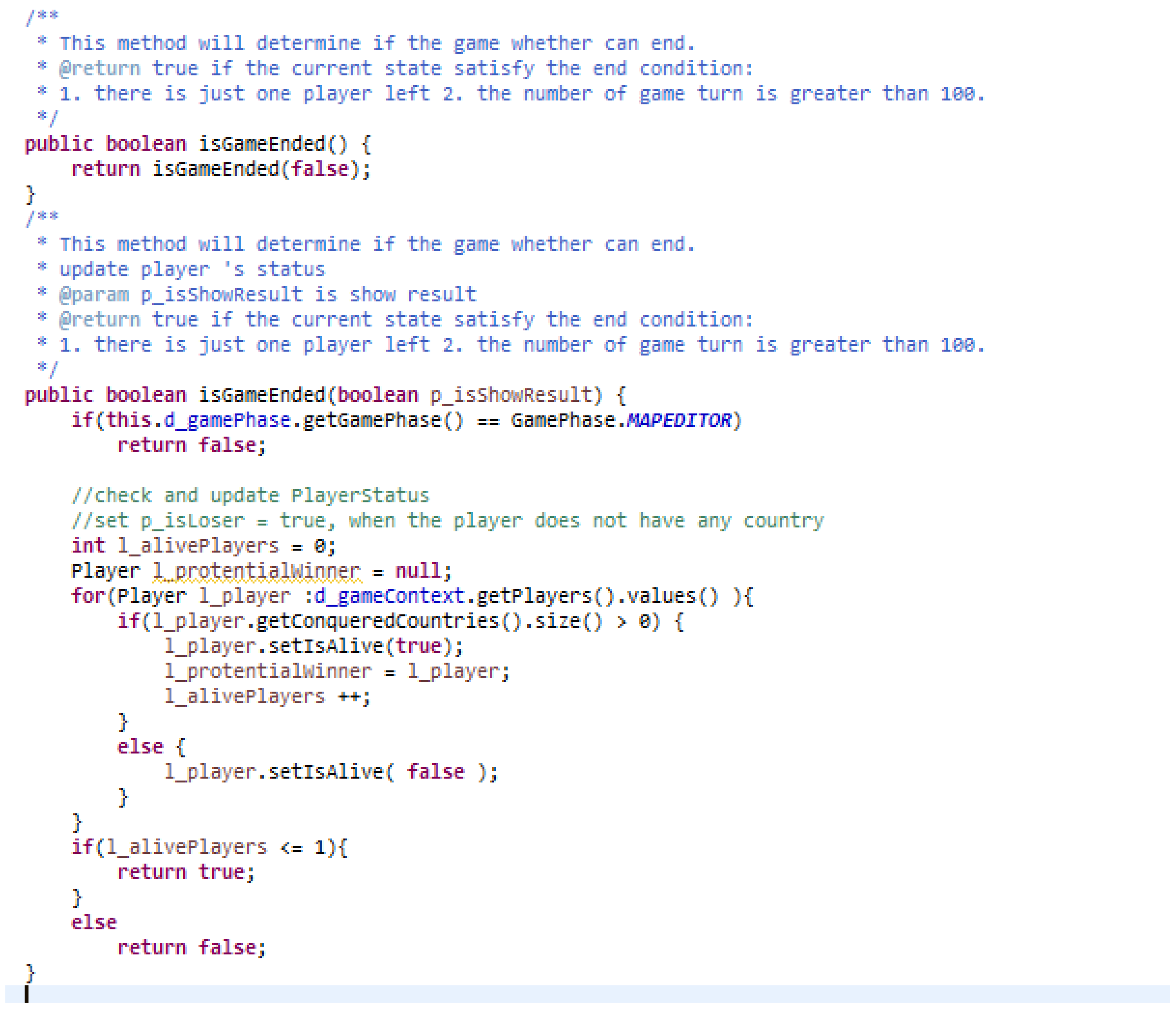
Reasoning:

The code for isGameEnded() was cumbersome and a bit inefficient, so we refactored it to run faster and improve code quality.

Before:

```java
/**
 * This method will determine if the game whether can end.
 * @return true if the current state satisfy the end condition:
 * 1. there is just one player left 2. the number of game turn is greater than 100.
 */
public boolean isGameEnded() {
        if(this.d_gamePhase.getGamePhase() == GamePhase.MAPEDITOR)
                return false;

        //check and update PlayerStatus
        //set p_isLoser = true, when the player does not have any country
        int l_alivePlayers = 0;
        Player l_protentialWinner = null;
        for(Player l_player :d_gameContext.getPlayers().values() ){
                if(l_player.getConqueredCountries().size() > 0) {
                        l_player.setIsAlive(true);
                        l_protentialWinner = l_player;
                        l_alivePlayers ++;
                }
        }
        if(l_alivePlayers <= 1){
                GenericView.println("-------------------- Game End");
                if(l_alivePlayers == 1) {

                        GenericView.printSuccess("player " + l_protentialWinner.getName() + " wins the game.");

                        if(d_gameContext.getIsTournamentMode() == true) {

                                d_tournamentContext.getResults()[d_mapIndex][d_gameIndex] = l_protentialWinner.getName();
                        }
                }
                else {
                        GenericView.printSuccess("All the player died.");

                        if(d_gameContext.getIsTournamentMode() == true) {

                                d_tournamentContext.getResults()[d_mapIndex][d_gameIndex] = "Draw";
                        }
                }
                GenericView.println("-------------------- Reboot the game");
                this.reboot();
                return true;
        }
        else
                return false;
}
```

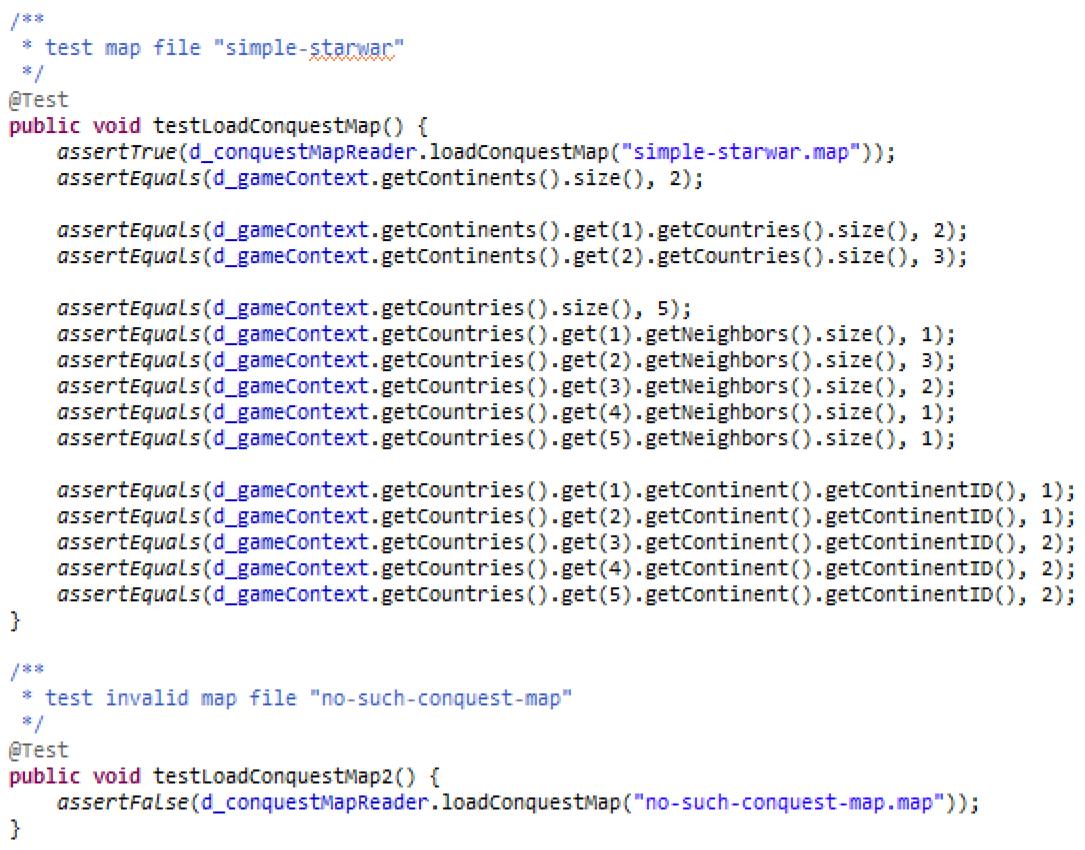
After:



```
/**
 * This method will determine if the game whether can end.
 * @return true if the current state satisfy the end condition:
 * 1. there is just one player left 2. the number of game turn is greater than 100.
 */
public boolean isGameEnded() {
    return isGameEnded(false);
}
/**
 * This method will determine if the game whether can end.
 * update player 's status
 * @param p_isShowResult is show result
 * @return true if the current state satisfy the end condition:
 * 1. there is just one player left 2. the number of game turn is greater than 100.
 */
public boolean isGameEnded(boolean p_isShowResult) {
    if(this.d_gamePhase.getGamePhase() == GamePhase.MAPEDITOR)
        return false;

    //check and update PlayerStatus
    //set p_isLoser = true, when the player does not have any country
    int l_alivePlayers = 0;
    Player l_protentialWinner = null;
    for(Player l_player :d_gameContext.getPlayers().values() ){
        if(l_player.getConqueredCountries().size() > 0) {
            l_player.setIsAlive(true);
            l_protentialWinner = l_player;
            l_alivePlayers ++;
        }
        else {
            l_player.setIsAlive( false );
        }
    }
    if(l_alivePlayers <= 1){
        return true;
    }
    else
        return false;
}
```

4) Added adapters to read conquest map files.

<u>Tests:</u>

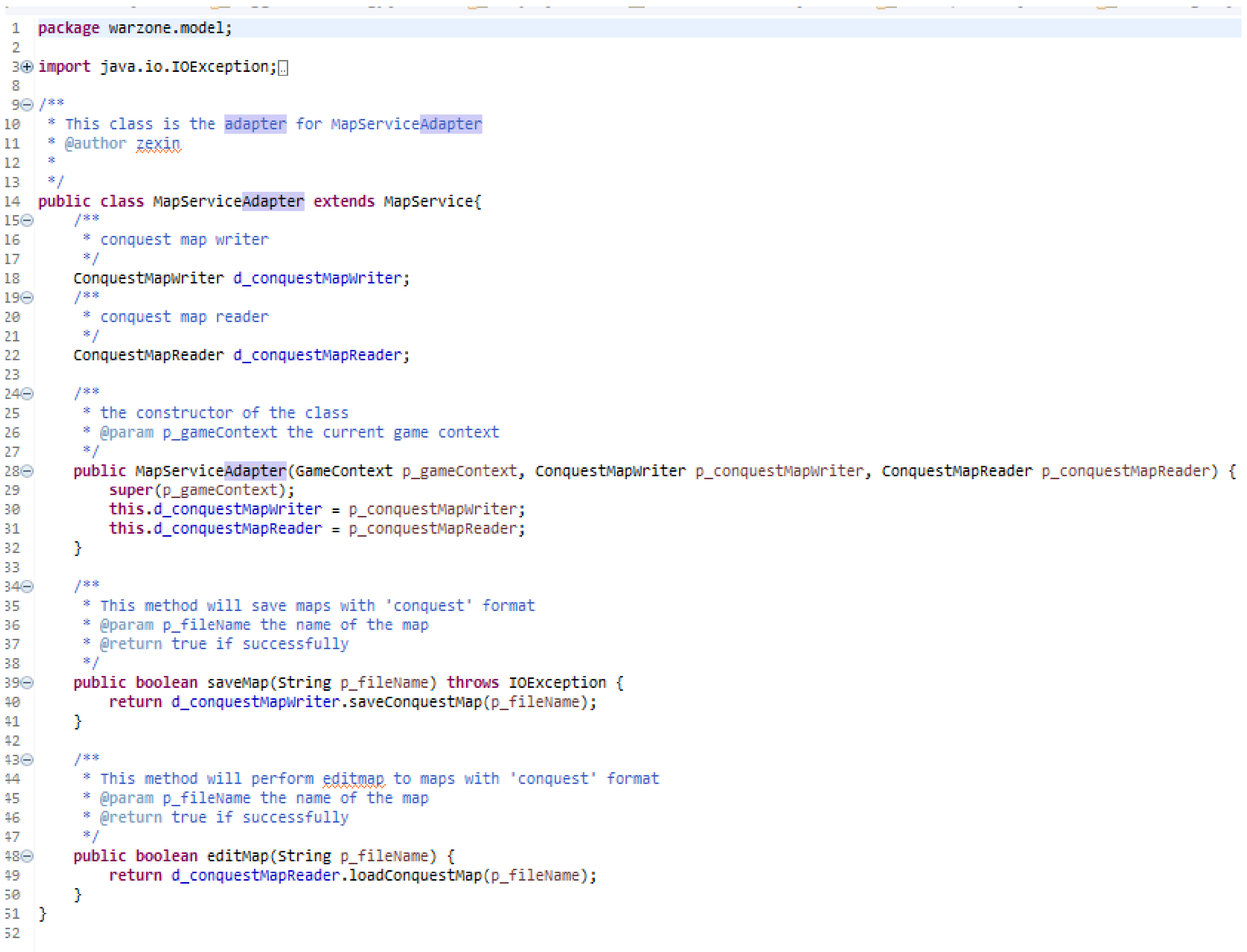


```java
/**
 * test map file "simple-starwar"
 */
@Test
public void testLoadConquestMap() {
    assertTrue(d_conquestMapReader.loadConquestMap("simple-starwar.map"));
    assertEquals(d_gameContext.getContinents().size(), 2);

    assertEquals(d_gameContext.getContinents().get(1).getCountries().size(), 2);
    assertEquals(d_gameContext.getContinents().get(2).getCountries().size(), 3);

    assertEquals(d_gameContext.getCountries().size(), 5);
    assertEquals(d_gameContext.getCountries().get(1).getNeighbors().size(), 1);
    assertEquals(d_gameContext.getCountries().get(2).getNeighbors().size(), 3);
    assertEquals(d_gameContext.getCountries().get(3).getNeighbors().size(), 2);
    assertEquals(d_gameContext.getCountries().get(4).getNeighbors().size(), 1);
    assertEquals(d_gameContext.getCountries().get(5).getNeighbors().size(), 1);

    assertEquals(d_gameContext.getCountries().get(1).getContinent().getContinentID(), 1);
    assertEquals(d_gameContext.getCountries().get(2).getContinent().getContinentID(), 1);
    assertEquals(d_gameContext.getCountries().get(3).getContinent().getContinentID(), 2);
    assertEquals(d_gameContext.getCountries().get(4).getContinent().getContinentID(), 2);
    assertEquals(d_gameContext.getCountries().get(5).getContinent().getContinentID(), 2);
}

/**
 * test invalid map file "no-such-conquest-map"
 */
@Test
public void testLoadConquestMap2() {
    assertFalse(d_conquestMapReader.loadConquestMap("no-such-conquest-map.map"));
}
```

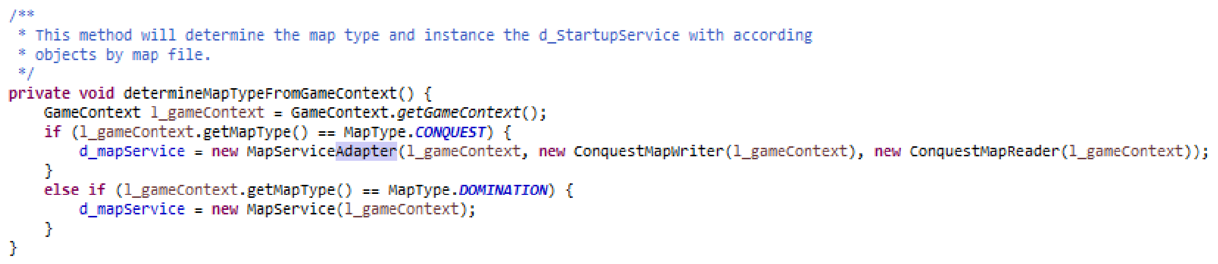
Reasoning:

This was part of the build specifications, so it was a high priority target. This refactoring allowed users to use both Domination and Conquest map files when playing the game.

Before/After:



```java
package warzone.model;

import java.io.IOException;

/**
 * This class is the adapter for MapServiceAdapter
 * @author zexin
 *
 */
public class MapServiceAdapter extends MapService{
    /**
     * conquest map writer
     */
    ConquestMapWriter d_conquestMapWriter;
    /**
     * conquest map reader
     */
    ConquestMapReader d_conquestMapReader;

    /**
     * the constructor of the class
     * @param p_gameContext the current game context
     */
    public MapServiceAdapter(GameContext p_gameContext, ConquestMapWriter p_conquestMapWriter, ConquestMapReader p_conquestMapReader) {
        super(p_gameContext);
        this.d_conquestMapWriter = p_conquestMapWriter;
        this.d_conquestMapReader = p_conquestMapReader;
    }

    /**
     * This method will save maps with 'conquest' format
     * @param p_fileName the name of the map
     * @return true if successfully
     */
    public boolean saveMap(String p_fileName) throws IOException {
        return d_conquestMapWriter.saveConquestMap(p_fileName);
    }

    /**
     * This method will perform editmap to maps with 'conquest' format
     * @param p_fileName the name of the map
     * @return true if successfully
     */
    public boolean editMap(String p_fileName) {
        return d_conquestMapReader.loadConquestMap(p_fileName);
    }
}
```



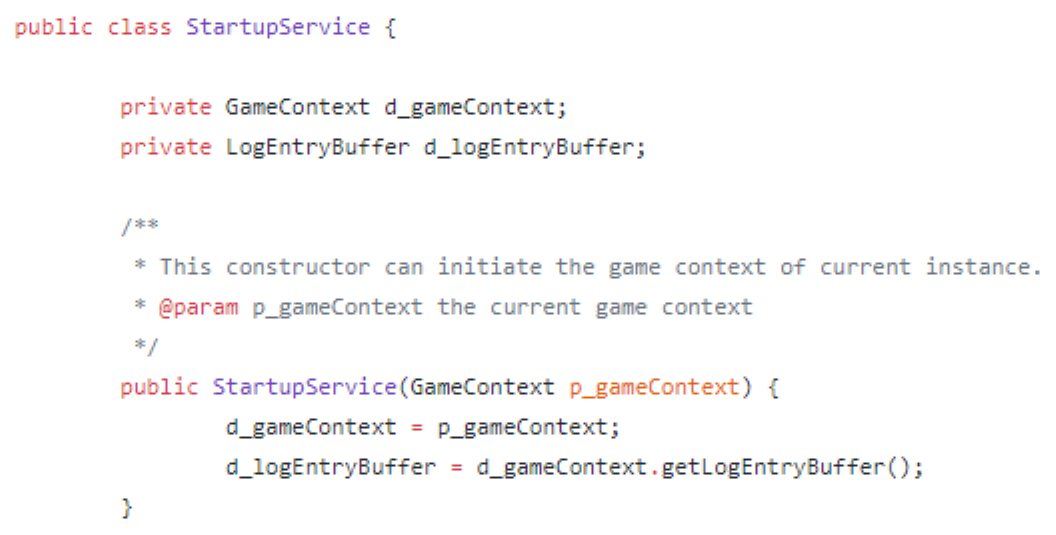
```java
/**
 * This method will determine the map type and instance the d_StartupService with according
 * objects by map file.
 */
private void determineMapTypeFromGameContext() {
    GameContext l_gameContext = GameContext.getGameContext();
    if (l_gameContext.getMapType() == MapType.CONQUEST) {
        d_mapService = new MapServiceAdapter(l_gameContext, new ConquestMapWriter(l_gameContext), new ConquestMapReader(l_gameContext));
    }
    else if (l_gameContext.getMapType() == MapType.DOMINATION) {
        d_mapService = new MapService(l_gameContext);
    }
}
```

5) New StartupService constructor.

<u>Tests:</u> There are several tests to be found in StartupServiceTest.java

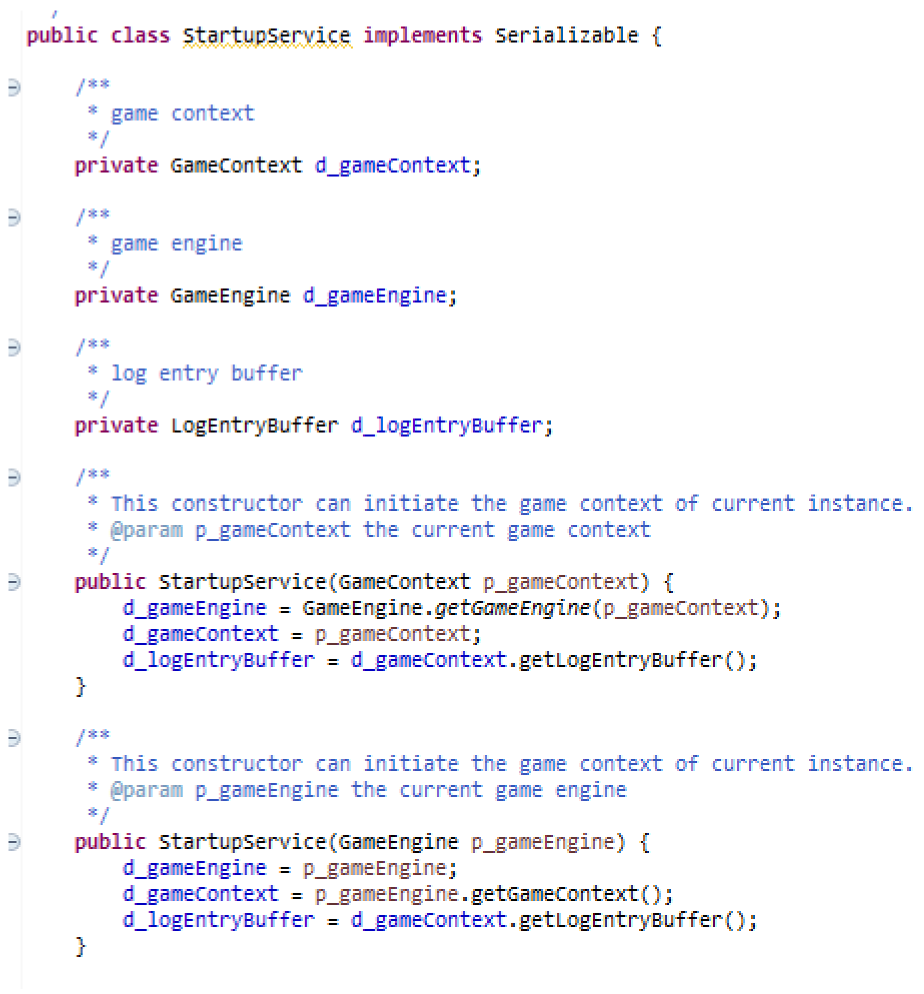
<u>Reasoning:</u> This additional constructor provides more flexibility when coding.

<u>Before:</u>



```
public class StartupService {

        private GameContext d_gameContext;
        private LogEntryBuffer d_logEntryBuffer;

        /**
         * This constructor can initiate the game context of current instance.
         * @param p_gameContext the current game context
         */
        public StartupService(GameContext p_gameContext) {
                d_gameContext = p_gameContext;
                d_logEntryBuffer = d_gameContext.getLogEntryBuffer();
        }
```

After:



```
 /
public class StartupService implements Serializable {

    /**
     * game context
     */
    private GameContext d_gameContext;

    /**
     * game engine
     */
    private GameEngine d_gameEngine;

    /**
     * log entry buffer
     */
    private LogEntryBuffer d_logEntryBuffer;

    /**
     * This constructor can initiate the game context of current instance.
     * @param p_gameContext the current game context
     */
    public StartupService(GameContext p_gameContext) {
        d_gameEngine = GameEngine.getGameEngine(p_gameContext);
        d_gameContext = p_gameContext;
        d_logEntryBuffer = d_gameContext.getLogEntryBuffer();
    }

    /**
     * This constructor can initiate the game context of current instance.
     * @param p_gameEngine the current game engine
     */
    public StartupService(GameEngine p_gameEngine) {
        d_gameEngine = p_gameEngine;
        d_gameContext = p_gameEngine.getGameContext();
        d_logEntryBuffer = d_gameContext.getLogEntryBuffer();
    }
```