

Refactoring List Build 3-Group 7

REFACTOR TESTS AND ADD MORE TESTS BEFORE ACTUALLY ADDING NEW SPECIFICATIONS AND CODE

Build 1 & 2: Tests cases were independent of each other and many stuff were hardcoded. Consequently, we could not test our game for all maps.

Build 3: We were required to load domination map as well. We wanted to ensure our game works for both domination and conquest maps as well for all maps of these 2 types. Hence we refactored all our tests (especially map tests) to make them independent of each other and independent of map files. The tests can be run on all maps now. This is important and reassures us that our game works on all files.

This also makes adding more tests (especially for conquest) easier.

The code is more readable and understandable as well.

(Build 1 & 2: Code hard to read & understand & very difficult to add new tests)

```
/**
 * MapEditorTest class tests cases relevant with the controller components of the map editor.
 */
@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class MapEditorControllerTest {

    static private CommandPromptView testCmdView;
    static private MapService testMapService;
    static private MapLoaderController testMapLoader;
    static private GameController testGameController;
    String test_map;

    //define test variables
    URI uri;
    String mapname, file, savename, inputcommand, continentcommand1, continentcommand2, continentcommand3, continentcommand4, continentcommand5, continentcommand6, countrycommand1, countrycommand2, countrycommand3, countrycommand4, countrycommand5, countrycommand6, neighborcommand1, neighborcommand2, neighborcommand3, neighborcommand4, neighborcommand5, neighborcommand6, newcontinentstr1, newcontinentstr2a, newcontinentstr2b, newcontinentstr3, newcontinentstr4a, newcontinentstr4b, delcontinentstr1, delcontinentstr2a, delcontinentstr2b, newcontinentstr3, delcontinentstr3, newcontinentstr4a, delcontinentstr4a, delcontinentstr4b, newcountrystr1, countrycontinentstr1, newcountrystr2a, countrycontinentstr2a, newcountrystr2b, countrycontinentstr2b, delcountrystr1, delcountrystr2a, delcountrystr2b, newcountrystr3, countrycontinentstr3, delcountrystr3, newcountrystr4a, newcountrystr4b, countrycontinentstr4a, countrycontinentstr4b, delcountrystr4a, delcountrystr4b, countrystr1, neighborstr1, countrystr2a, neighborstr2a, countrystr2b, neighborstr2b, countrystr3, neighborstr3, countrystr4a, neighborstr4a, countrystr4b, neighborstr4b, countrystr5a, neighborstr5a, countrystr5b, neighborstr5b, countrystr6a, neighborstr6a, countrystr6b, neighborstr6b, countrystr6c, neighborstr6c, countrystr6d, neighborstr6d;
    String[] continentcommands1, continentcommands2, continentcommands3, continentcommands4, continentcommands5, continentcommands6, countrycommands1, countrycommands2, countrycommands3, countrycommands4, countrycommands5, countrycommands6, neighborcommands1, neighborcommands2, neighborcommands3, neighborcommands4, neighborcommands5, neighborcommands6;
    int initcontinentsize, initcountrysize, expectedcontinentsize1, expectedcontinentsize2, expectedcontinentsize3, expectedcontinentsize4, expectedcontinentsize5, expectedcontinentsize6, expectedcountrysize1, expectedcountrysize2, expectedcountrysize3, expectedcountrysize4, expectedcountrysize5;
    static int testCounter;
    boolean mapIsRead;
    Optional<String> inputmap;
    Optional<Integer> country1, neighbor1, country2a, neighbor2a, country2b, neighbor2b, country3, neighbor3, country4a, neighbor4a, country4b, neighbor4b, country5a, neighbor5a, country5b, neighbor5b, country6a, neighbor6a, country6b, neighbor6b, country6c, neighbor6c, neighbor6d;
}
```

(Build 3: Very little to No hardcoding at all, Tests are independent and run on all maps and on both domination and conquest map types.)

```
/**
 * This method is executed by {@link #test008_addAndRemoveCountry()}
 * @param name1
 * @param continentName1
 * @param name2
 * @throws IOException on invalid values
 */
public void addAndRemoveCountry(String name1, String continentName1, String name2) throws IOException {
    testMapLoader.readCommand("editcountry -add "+name1+" "+continentName1+" "+" -remove "+name2);
}

/**
 * This method is executed by {@link #test009_addNeighbor()}
 * @param origin
 * @param neighborCountry
 * @throws IOException on invalid values
 * country1 is the origin country retrieved by the testMapLoader
 * neighbor1 is the neighboring country retrieved by the testMapLoader
 * borders1 is the map that stores countries and their adjacent neighbors
 * part1 is the adjacency list for country1
 */
public void addNeighbor(String origin, String neighborCountry) throws IOException {
    country1 = testMapLoader.getMapService().findCorrespondingIdByCountryName(origin);
    neighbor1 = testMapLoader.getMapService().findCorrespondingIdByCountryName(neighborCountry);
    testMapLoader.readCommand("editneighbor -add "+origin+" "+neighborCountry);
    borders1 = testMapLoader.getMapService().getAdjacencyCountriesMap();
    pair1 = borders1.get(country1.get());
}
```

```
 * @param neighborCountry
 * @throws IOException on invalid values
 * country1 is the origin country retrieved by the testMapLoader
 * neighbor1 is the neighboring country retrieved by the testMapLoader
 * borders1 is the map that stores countries and their adjacent neighbors
 * part1 is the adjacency list of country1
 */
public void removeNeighbor(String origin, String neighborCountry) throws IOException {
    country1 = testMapLoader.getMapService().findCorrespondingIdByCountryName(origin);
    neighbor1 = testMapLoader.getMapService().findCorrespondingIdByCountryName(neighborCountry);
    testMapLoader.readCommand("editneighbor -remove "+origin+" "+neighborCountry);
    borders1 = testMapLoader.getMapService().getAdjacencyCountriesMap();
    pair1 = borders1.get(country1.get());
}

/**
 * This method is executed by {@link #test011_addAndRemoveNeighbor()}
 * @param origin1
 * @param neighborCountry1
 * @param origin2
 * @param neighborCountry2
 * @throws IOException
 * country1 is the first origin country retrieved by the testMapLoader
 * neighbor1 is the to-be-added neighboring country retrieved by the testMapLoader
 * borders1 is the map that stores countries and their adjacent neighbors
 * part1 is the adjacency list of country1
 * country2 is the second origin country retrieved by the testMapLoader
 * neighbor2 is the to-be-removed neighboring country retrieved by the testMapLoader
 * pair2 is the adjacency list of country2
 */
public void addAndRemoveNeighbor(String origin1, String neighborCountry1, String origin2, String neighborCountry2) throws IOException {
    country1 = testMapLoader.getMapService().findCorrespondingIdByCountryName(origin1);
    neighbor1 = testMapLoader.getMapService().findCorrespondingIdByCountryName(neighborCountry1);
    country2 = testMapLoader.getMapService().findCorrespondingIdByCountryName(origin2);
    neighbor2 = testMapLoader.getMapService().findCorrespondingIdByCountryName(neighborCountry2);

    testMapLoader.readCommand("editneighbor -add "+origin1+" "+neighborCountry1+" -remove "+origin2+" "+neighborCountry2);
    //create map object from adjacency list
    borders1 = testMapLoader.getMapService().getAdjacencyCountriesMap();
    //get pair of country and neighbor
    pair1 = borders1.get(country1.get());
    pair2 = borders1.get(country2.get());
}
```

CHANGED CODE STRUCTURE FOR VALIDATION IN CONTROLLERS AND GAME LOGIC (MORE MODULAR, READABLE, UNDERSTANDABLE, INDEPENDENT CRITERIAS, MORE EFFICIENT)

(Build 1&2: Lots of Nested Ifs and Elses for Validation + need to go through all validations even if some may have failed earlier)

```
if(!boolStartupPhaseOver.get()) {

    if(!boolStartupPhaseSet) {
        this.mapService.setState(GameState.START_UP);
        this.boolStartupPhaseSet=true;
    }

    startupPhaseController.readCommand(command, this.boolStartupPhaseOver);
}

else {

    if(this.mapService.getGameState()==GameState.REINFORCE) {

        //reinforcementGameController.readCommand(command)

        this.currentPlayer=players.get(currentPlayerIndex);

        reinforcementGameController=new ReinforceGameController(this.currentPlayer,
            this.mapService,
            startupPhaseController,
            command,
            view);

    }

    else if(this.mapService.getGameState()==GameState.FORTIFY) {

        fortificationGameController=new FortifyGameController(this.currentPlayer,
            this.mapService,
            this.startupPhaseController,
            command,
            this.boolFortificationPhaseOver,
            view);

        switchNextPlayer();
    }
}
```

(Fortification Inefficient Validation)

```
try {
    assert(fortifyState.getName().equals(GameState.FORTIFY.toString()));

} catch (Exception e) {
    // TODO: handle exception
    System.out.println("Not right Game state");
}

try {
    Map<Integer, Set<Integer>> adjacentCountriesList = mapService.getAdjacencyCountriesMap();
    Optional<Integer> toId = mapService.findCorrespondingIdByCountryName(toCountry.toString());
    Optional<Integer> fromId = mapService.findCorrespondingIdByCountryName(fromCountry.toString());
    neighbouringCountries = adjacentCountriesList.get(fromId);
    assert(neighbouringCountries.contains(toId));
} catch (Exception e) {
    System.out.println("Countries not adjacent to each other");
}

try {
    assert(fromCountry.getPlayer().equals(toCountry.getPlayer()));
} catch (Exception e) {
    System.out.println("Check if both are same player's countries");
}

try {
    assert(fromCountry.getSoldiers() > 1);
} catch (Exception e) {
    System.out.println("Not enough soldiers in source country");
}
}

/*
```

(Build 3: Consolidate Conditional Expressions)

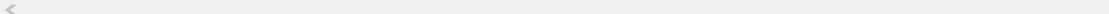
```
/**
 * Checks User Command for correct map format, player strategies. numGames and numTurns
 * @param command User command
 * @return true if all validations are met
 */
public boolean validateTournamentConditions(String command) {

    if(!(command.contains("-p")&&command.contains("-m")&&command.contains("-g")&&command.contains("-d"))) {
        phaseView.displayMessage("Tournament Command lacks 1 or more params");
        return false;
    }

    if(!validateMapCommand(command)) return false;
    if(!validatePlayerCommand(command)) return false;
    if(!validateNumGames(command)) return false;
    if(!validateNumTurns(command)) return false;
}
```

(Efficient, More Modular, Readable and Understandable Fortification Validation)

```
public boolean validateFortifyConditions(PlayerService playerService) {  
    this.boolFortifyValidationMet=true;  
    checkCountryAdjacencyForFortification(playerService.getMapService());  
    if(boolFortifyValidationMet) {  
        checkCountriesBelongToCurrentPlayer(playerService);  
    }  
    if(boolFortifyValidationMet) {  
        checkCountryOwnership();  
    }  
    if(boolFortifyValidationMet) {  
        checkNumSoldiers();  
    }  
    return this.boolFortifyValidationMet;  
}
```



REPLACED LISTS BY SETS & ITERATORS EVERYWHERE CONCURRENT EDITS WERE BEING MADE

When simple Lists are used, there were intermittent bugs and there was ConcurrentEdit Exception everywhere a list that was being traversed had an element removed. It was crucial to fix this issue and make the code more robust. Hence we replaced lists and use of for loops in such cases by using iterators that allowed concurrent additions and deletions while a list was being traversed. We then thoroughly tested the game and no longer encountered such errors.

(Concurrent Edit Exception and Consequent Bugs)

```
Phase View:
Defender has no cards to be transferred.
Phase View: Player Removed: b
Domination View: a controls 60.0 % of the map and has 33 total number of armies.
a owns: anea

Domination View: c controls 40.0 % of the map and has 29 total number of armies.
c owns: osea

Domination View: a controls 60.0 % of the map and has 33 total number of armies.
a owns: anea

Domination View: c controls 40.0 % of the map and has 29 total number of armies.
c owns: osea

Phase View: Index: 2, Size: 2
showplayer
Phase View: Index: 2, Size: 2
attack
Phase View: Index: 2, Size: 2

Phase View: 0

Phase View: 0
```

(Sets and Iterators instead of Lists)

```
//get first country from the country list
//Get set of countries using service
Set<Country> countriesFromService = mapLoaderController.getMapService().getCountries();
//Get the first country in the set using iterator
Iterator<Country> countryIterator = countriesFromService.iterator();
//Set the country
Country originCountry = countryIterator.next();
```

Other Potential Refactoring (Not done)

Use streams everywhere and uniformly throughout the project instead of traditional for loops.

Streams improve performance and are usually shorter and more readable and hence understandable.

We think using streams would be beneficial to this project as many controllers and model classes implementing the main game logic are very long and hence tedious to understand.

(Overlooked in Build 2 because only 1 member had profound knowledge of streams which has a pretty steep learning curve and requires time to fully master)

We again considered using streams in Build 3. However, we decided against it as We did not implement a GUI for which it could have been more useful and efficient.

Also, the very tight deadline we had (essentially only 1 week) to deliver build 3 also played a part and we set it low on our priority list.

Make all Model classes Observable.

We only have 2 Observable classes in our project: MapService.class and PlayerService.class.

These classes maintain the Map State and Player State which is essentially what need to be made notifyable to users for them to obtain useful information. We triggered notifications manually everytime a method was fully executed based on (Gamma et al., 1994.) instead of making state-setting methods automatically call notify everytime a state was changed.

We considered making state-setting methods trigger notification themselves as making controllers and other entities trigger updates after a series of state-setting operations was manually tedious and there could have been a tendency to forget triggering updates sometimes.

However, making state-setting methods trigger updates would have been more useful if we would implement a GUI. Since we did not do that, it made no sense to change our entire code when everything was working perfectly fine before and considering that we did not have many more notification mechanisms to implement for build 3. Domination and phase view had already been implemented for all key methods.

Also, if every class was observable, we would get too much needless intermediate updates after every operation which would be inefficient and an overhead on observers as well.

We also had a time constraint. Hence, we opted against this proposal.

REFERENCES

Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1994). *Design patterns: Elements of Reusable Object-Oriented Software*. p.331.