SOEN 6441

Advanced Programming Practices

Build 2

Refactoring Document

Team 20

Konstantin Chemodanov…. 40049285
Farzaneh Jamshidi…..…...…40075234
Saman Safaei……. 40119399
Ali Molla Mohammadi …….40042597
Negar Ashouri……40071530

March 2019

Department of Computer Science and Software Engineering

Refactoring:

After receiving the feedbacks of the first build and new requirements for the second build, we found out that some specific parts of our code needs refactoring in order to meet new requirements. Beside all the new parts and functionalities that we had to add to the code, we came up with a list of potential refactoring targets, as we had to apply and add observer pattern to our MVC design and we needed to move all of the reinforcement, attack and fortification phases as methods of the Player class.

We faced challenges during the refactoring operation such as finding out where to apply changes for new requirements, moving all of the methods into the player class and choosing the best signatures for methods in favor of having the least possible changes in the code, finding a new protocol which can enable fast communication between server and client in order to implement the observer pattern for “phase view”, “players world domination view” and “card exchange view”.

We found "WebSocket" as the best solution for our situation, because it provides new capability for web applications like full-duplex and two-way communication. So in the system where the client and server need to exchange data at high frequency and with low latency, WebSocket is the best solution. We used WebSocket in both client side and server side.

Regarding to update the UI when there is a change in state of the observer, we first decided to compel the frontend to check backend in specified intervals, which leaded to a bad performance and high overhead. For solving this problem we tried "Push" instead, and backend will push its changes to the frontend. This approach results a higher performance.

Based on our design the player status shows the stage which player is on it, so we consider the set status as the point which we need to notify the observers as well as updating the logger for player view phase, so whenever the player status has been changed we push the new data from backend to UI and notify the observers for domination view and player phase view.

Also after adding the attack-all-out phase both code for UI and Backend has been updated accordingly and test cases has been added.

```
    connectAndSubscribeForViewPhase() {
        const socket = new SockJS(RiskgameBackendServer.url + 'gs-guide-websocket');
        this.stompClient = Stomp.over(socket);
        const _this = this;

        _this.stompClient.connect({}, function (frame) {
            console.log('Connected: ' + socket);
            _this.stompClient.subscribe('/topic/viewPhase', function (data) {
                console.log('viewPhase FROM WEBSOCKET PUSHED', data);
                _this.viewPhasePlayer = JSON.parse(data['body']);
            });
        });
    }
}
```

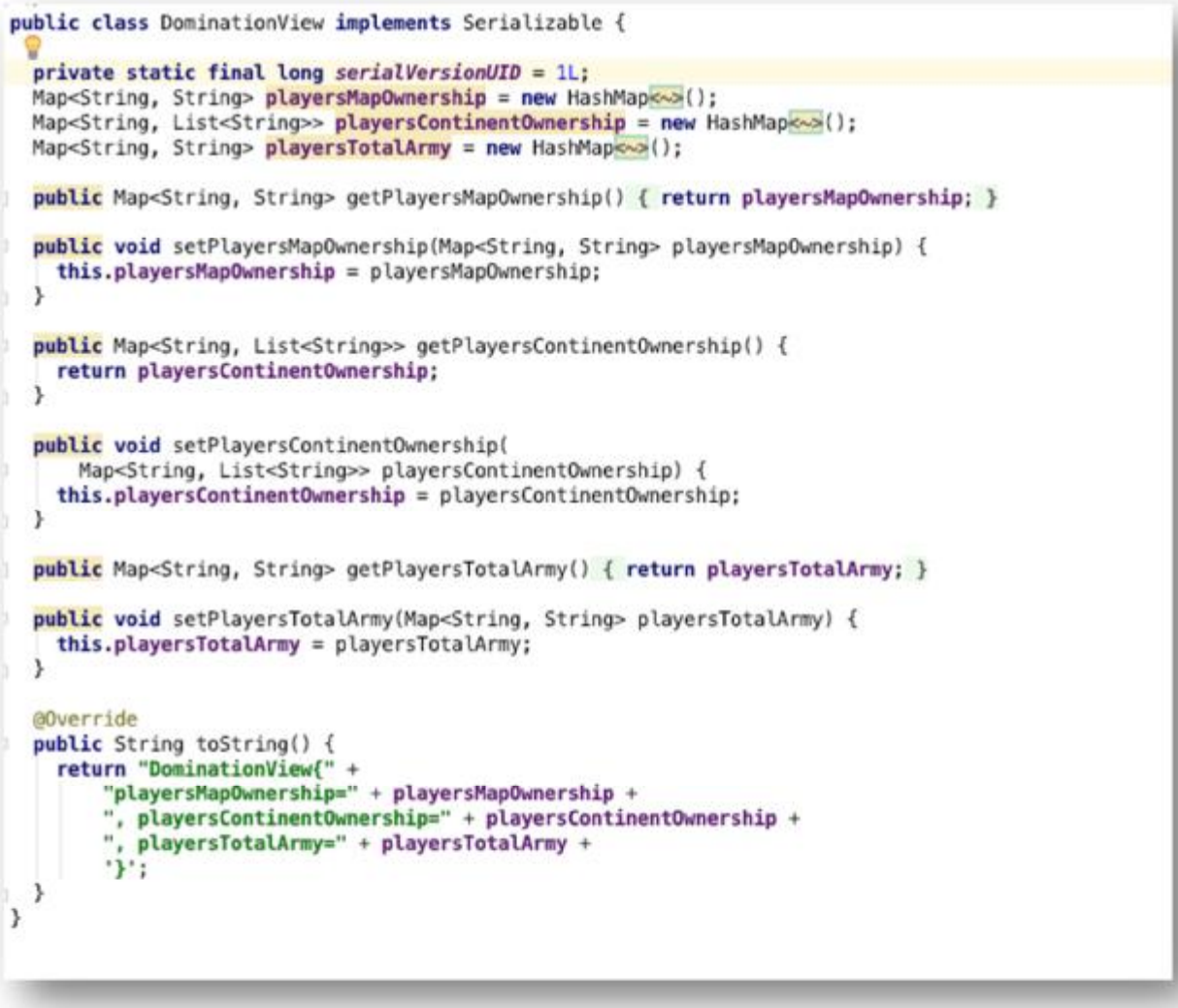
```
    connectAndSubscribeForDominationView() {
        const socket = new SockJS(RiskgameBackendServer.url + 'gs-guide-websocket');
        this.stompClient = Stomp.over(socket);
        const _this = this;

        _this.stompClient.connect({}, function (frame) {
            console.log('Connected: ' + socket);

            _this.stompClient.subscribe('/topic/dominationView', function (data) {

                _this.playersMapOwnershipList = [];
                _this.playersContinentOwnershipList = [];
                _this.playersTotalArmyList = [];

                console.log('Domination View FROM WEBSOCKET PUSHED', data);
                const dominationView = JSON.parse(data['body']);
                for (let key in dominationView['playersMapOwnership']) {
                    // console.log('key: ', key, ', _this.cominationView[key]: ', dominationView['playersMapOwnership'][key]);
                    let tmp = new PlayersMapOwnership();
                    tmp.playerName = key;
                    tmp.percentage = dominationView['playersMapOwnership'][key];
                    _this.playersMapOwnershipList.push(tmp);
                }
                for (let key in dominationView['playersContinentOwnership']) {
                    // console.log('key: ', key, ', _this.cominationView[key]: ', dominationView['playersContinentOwnership'][key]);
                    let tmp = new PlayersContinentOwnership();
                    tmp.playerName = key;
                    tmp.continents = dominationView['playersContinentOwnership'][key];
                    _this.playersContinentOwnershipList.push(tmp);
                }
                for (let key in dominationView['playersTotalArmy']) {
                    // c let tmp: PlayersTotalArmy  _this.cominationView[key]: ', dominationView['playersTotalArmy'][key]);
                    let tmp = new PlayersTotalArmy();
                    tmp.playerName = key;
                    tmp.armies = dominationView['playersTotalArmy'][key];
                    _this.playersTotalArmyList.push(tmp);
                }
            });
        });

    }
```

```java
public class ViewPhase implements Observer {

  private final static Logger LOGGER = LoggerFactory.getLogger(ViewPhase.class);

  @Autowired
  private SimpMessagingTemplate template;

  /**
   * This method is called whenever the observed object is changed. An application calls an
   * <tt>Observable</tt> object's
   * <code>notifyObservers</code> method to have all the object's
   * observers notified of the change.
   *
   * @param o the observable object.
   * @param arg an argument passed to the <code>notifyObservers</code>
   */
  @Override
  public void update(Observable o, Object arg) {
    Player player = (Player) o;
    DominationView dominationView = GameMap.getDominationView();
    LOGGER.info(dominationView.toString() + "\n" + player.getPlayerLog().toString());
    try {
      pushViewPhaseData(player);
      pushDominationView(dominationView);
    } catch (Exception e) {
      final String stackTrace = ExceptionUtils.getStackTrace(e);
      LOGGER.error(stackTrace);
    }

  }

  /**
   * send the data to web socket endpoint which register for player view changes
   *
   * @param player player which its status has been changed
   * @throws Exception if fail to push notification or network face run time issues.
   */
  public void pushViewPhaseData(Player player) throws Exception {
    template.convertAndSend( destination: "/topic/viewPhase", player);
  }


  /**
   * send the data to web socket endpoint which register for domination view changes
   *
   * @param dominationView player which its status has been changed
   * @throws Exception if fail to push notification or network face run time issues.
   */

  public void pushDominationView(DominationView dominationView) throws Exception {
    template.convertAndSend( destination: "/topic/dominationView", dominationView);
  }
}
```

Below tests confirm that after changing the player status in the game, domination view and player view has been updated and update the player view phase accordingly.

```
/**
 * Method to test valid attack allout
 */
@Test
public void testSuccessfulAttackAllOut() {
  ResponseEntity<Object> responseEntity = attackPhaseEndpoint.attackAllOut( playerNumber: "1",  attackingCountry: "usa",  defendingCountry: "mexico");
  assertEquals(HttpStatus.OK, responseEntity.getStatusCode());
  assertTrue( condition: GameMap.getCountries().get("usa").getArmies() > 0);
  assertTrue( condition: GameMap.getCountries().get("mexico").getArmies() > 0);
}

/**
 * Method to test invalid attack all out, player attacks itself
 */
@Test
public void testSamePlayerAllOut() {
  ResponseEntity<Object> responseEntity = attackPhaseEndpoint.attackAllOut( playerNumber: "1",  attackingCountry: "usa",  defendingCountry: "bermuda");
  assertEquals(responseEntity.getStatusCode(), HttpStatus.BAD_REQUEST);
}

/**
 * Method to test invalid attack all out, attacking country is not connected to defending country
 */
@Test
public void testNotConnectedCountriesAllOut() {
  ResponseEntity<Object> responseEntity = attackPhaseEndpoint
      .attackAllOut( playerNumber: "2",  attackingCountry: "mexico",  defendingCountry: "canada");
  assertEquals(responseEntity.getStatusCode(), HttpStatus.BAD_REQUEST);
}

/**
 * Method to test invalid attack all out, player is not in attacking phase
 */
@Test
public void testNotInAttackingPhaseAllOut() {
  ResponseEntity<Object> responseEntity = attackPhaseEndpoint.attackAllOut( playerNumber: "3",  attackingCountry: "mexico",  defendingCountry: "usa");
  assertEquals(responseEntity.getStatusCode(), HttpStatus.BAD_REQUEST);
}
}
```

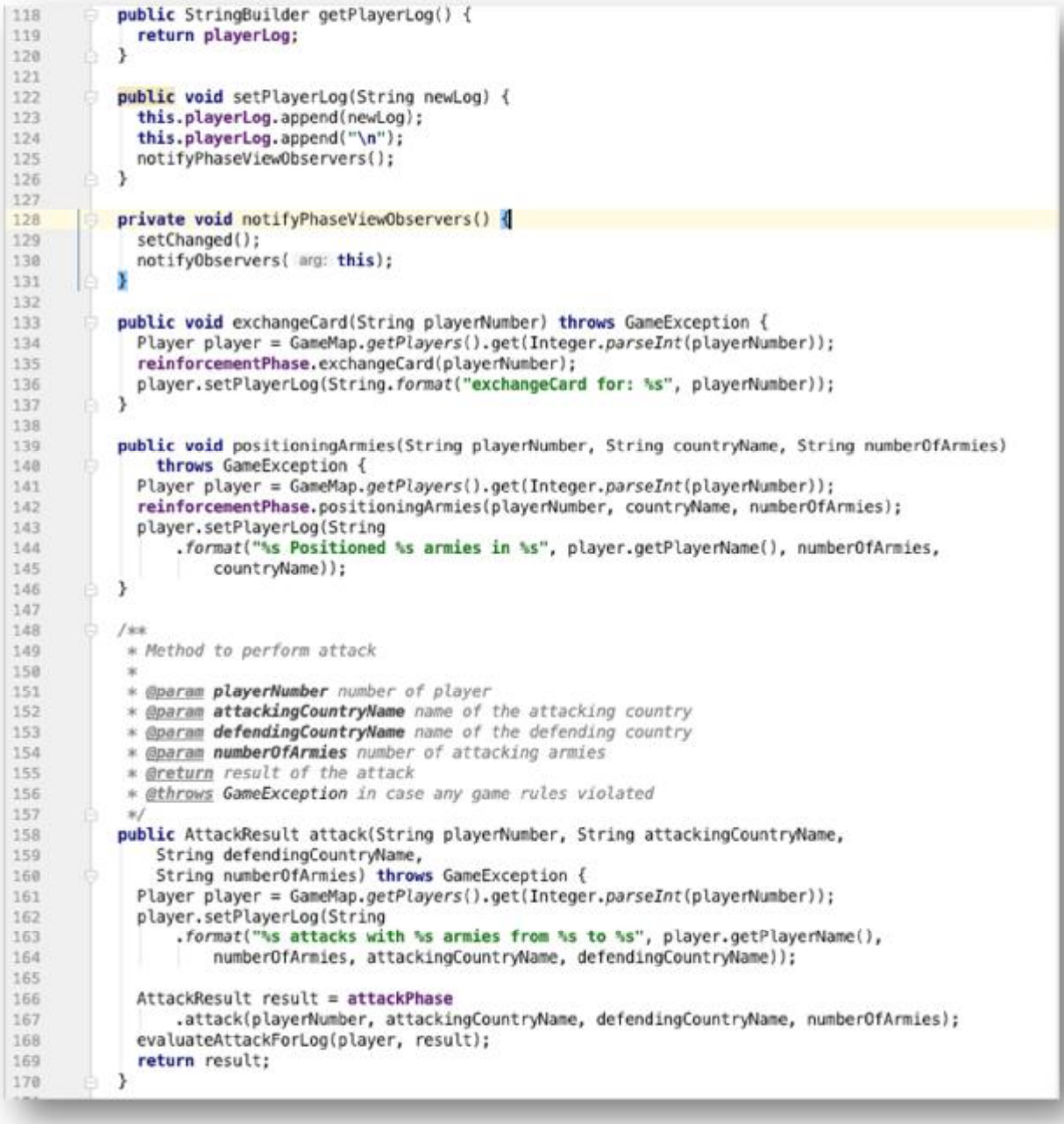
```java
public class ViewPhaseTest {

  @Autowired
  private ViewPhase viewPhase;

  private Player player;

  @Before
  public void before() {

    Continent asia = new Continent();
    Country ir = new Country( name: "IRAN");
    ir.setCountryId(1);

    List<Country> countryList = new ArrayList<>();
    countryList.add(ir);
    asia.setCountries(countryList);

    player = new PlayerRegular( playerName: "Test-Player");
    player.setPlayerStatus(PlayerStatus.Waiting);
    player.setCard(10);
    player.setArmy(20);
    player.setPlayerId(1);
    player.setPlayerName("TestName");
    player.setCountries(countryList);
  }

  @After
  public void after() {

  }

  @Test
  public void testPhaseView() {
    player.setPlayerStatus(PlayerStatus.Reinforcement);
    viewPhase.update(player,  arg: null);
    assertNotNull(GameMap.getDominationView());

  }

}
```



```java
public class DominationView implements Serializable {
  private static final long serialVersionUID = 1L;
  Map<String, String> playersMapOwnership = new HashMap<>();
  Map<String, List<String>> playersContinentOwnership = new HashMap<>();
  Map<String, String> playersTotalArmy = new HashMap<>();

  public Map<String, String> getPlayersMapOwnership() { return playersMapOwnership; }

  public void setPlayersMapOwnership(Map<String, String> playersMapOwnership) {
    this.playersMapOwnership = playersMapOwnership;
  }

  public Map<String, List<String>> getPlayersContinentOwnership() {
    return playersContinentOwnership;
  }

  public void setPlayersContinentOwnership(
      Map<String, List<String>> playersContinentOwnership) {
    this.playersContinentOwnership = playersContinentOwnership;
  }

  public Map<String, String> getPlayersTotalArmy() { return playersTotalArmy; }

  public void setPlayersTotalArmy(Map<String, String> playersTotalArmy) {
    this.playersTotalArmy = playersTotalArmy;
  }

  @Override
  public String toString() {
    return "DominationView{" +
        "playersMapOwnership=" + playersMapOwnership +
        ", playersContinentOwnership=" + playersContinentOwnership +
        ", playersTotalArmy=" + playersTotalArmy +
        '}';
  }
}
```

```java
public class ViewPhase implements Observer {

  private final static Logger LOGGER = LoggerFactory.getLogger(ViewPhase.class);

  @Autowired
  private SimpMessagingTemplate template;

  /**
   * This method is called whenever the observed object is changed. An application calls an
   * <tt>Observable</tt> object's
   * <code>notifyObservers</code> method to have all the object's
   * observers notified of the change.
   *
   * @param o the observable object.
   * @param arg an argument passed to the <code>notifyObservers</code>
   */
  @Override
  public void update(Observable o, Object arg) {
    Player player = (Player) o;
    DominationView dominationView = GameMap.getDominationView();
    LOGGER.info(dominationView.toString() + "\n" + player.getPlayerLog().toString());
    try {
      pushViewPhaseData(player);
      pushDominationView(dominationView);
    } catch (Exception e) {
      final String stackTrace = ExceptionUtils.getStackTrace(e);
      LOGGER.error(stackTrace);
    }

  }

  /**
   * send the data to web socket endpoint which register for player view changes
   *
   * @param player player which its status has been changed
   * @throws Exception if fail to push notification or network face run time issues.
   */
  public void pushViewPhaseData(Player player) throws Exception {
    template.convertAndSend( destination: "/topic/viewPhase", player);
  }


  /**
   * send the data to web socket endpoint which register for domination view changes
   *
   * @param dominationView player which its status has been changed
   * @throws Exception if fail to push notification or network face run time issues.
   */

  public void pushDominationView(DominationView dominationView) throws Exception {
    template.convertAndSend( destination: "/topic/dominationView", dominationView);
  }
}
```

```java
/**
 * Method to perform attack all-out
 * @param playerNumber number of player
 * @param attackingCountryName name of the attacking country
 * @param defendingCountryName name of the defending country
 * @return result of the attack
 * @throws GameException in case any game rules violated
 */
public AttackResult attackAllOut(String playerNumber, String attackingCountryName,
    String defendingCountryName) throws GameException {
  Player player = GameMap.getPlayers().get(Integer.parseInt(playerNumber));
  player.setPlayerLog(String
      .format("%s attacks all-out from %s to %s", player.getPlayerName(),
          attackingCountryName, defendingCountryName));

  AttackResult result = attackPhase
      .attackAllOut(playerNumber, attackingCountryName, defendingCountryName);
  evaluateAttackForLog(player, result);
  return result;
}

public void fortification(String playerNumber, String sourceCountry, String destinationCountry,
    String numberOfArmies) throws GameException {
  Player player = GameMap.getPlayers().get(Integer.parseInt(playerNumber));
  fortificationPhase
      .fortification(playerNumber, sourceCountry, destinationCountry, numberOfArmies);
  player.setPlayerLog(String
      .format("%s fortified with %s armies from %s to %s", player.getPlayerName(), numberOfArmies,
          sourceCountry, destinationCountry));
}

private void evaluateAttackForLog(Player player, AttackResult result) {
  if (StringUtils.isBlank(result.getConqueredCountry())) {
    player.setPlayerLog("Attack failed");
  } else {
    player.setPlayerLog(
        "Attack is successful, country " + result.getConqueredCountry() + " is conquered");
  }
}

/**
 * Method to add card in case of first victory
 */
public void addCard() {
  if (firstVictory) {
    firstVictory = false;
    card++;
  }
}
```



```java
public StringBuilder getPlayerLog() {
  return playerLog;
}

public void setPlayerLog(String newLog) {
  this.playerLog.append(newLog);
  this.playerLog.append("\n");
  notifyPhaseViewObservers();
}

private void notifyPhaseViewObservers() {
  setChanged();
  notifyObservers( arg: this);
}

public void exchangeCard(String playerNumber) throws GameException {
  Player player = GameMap.getPlayers().get(Integer.parseInt(playerNumber));
  reinforcementPhase.exchangeCard(playerNumber);
  player.setPlayerLog(String.format("exchangeCard for: %s", playerNumber));
}

public void positioningArmies(String playerNumber, String countryName, String numberOfArmies)
    throws GameException {
  Player player = GameMap.getPlayers().get(Integer.parseInt(playerNumber));
  reinforcementPhase.positioningArmies(playerNumber, countryName, numberOfArmies);
  player.setPlayerLog(String
      .format("%s Positioned %s armies in %s", player.getPlayerName(), numberOfArmies,
          countryName));
}

/**
 * Method to perform attack
 *
 * @param playerNumber number of player
 * @param attackingCountryName name of the attacking country
 * @param defendingCountryName name of the defending country
 * @param numberOfArmies number of attacking armies
 * @return result of the attack
 * @throws GameException in case any game rules violated
 */
public AttackResult attack(String playerNumber, String attackingCountryName,
    String defendingCountryName,
    String numberOfArmies) throws GameException {
  Player player = GameMap.getPlayers().get(Integer.parseInt(playerNumber));
  player.setPlayerLog(String
      .format("%s attacks with %s armies from %s to %s", player.getPlayerName(),
          numberOfArmies, attackingCountryName, defendingCountryName));

  AttackResult result = attackPhase
      .attack(playerNumber, attackingCountryName, defendingCountryName, numberOfArmies);
  evaluateAttackForLog(player, result);
  return result;
}
```