Milestone 3

Team 7 - 2023/04/20
Richard Kosta
Hunter Masur
Aidan Mazany
Evan Sellers

Executive Summary

As a team the goal of milestone 3 was to get the user interface updated and add secret missions - additionally, we wanted to clean up some of the tests before. Because of time constrains and issues that arouse during development we were not able to dedicate as much time as we wanted to testing. We stil do not have great coverage on the game class, which we were striving to complete on this milestone, but was not able to achieve the level we wanted. Because of this we will also be adding additional testing for game in milestone 4.

When it came to finding bad code smells for the codebase, this was not a major issue because we had implement some bad design choices while implement milestone 2. However when it came finding bad code smells in the tests, this proved to be more challenging than expecting. None of tests had setup procedures, so there were not the optunity to fix bad smells that arise from that, but we ended up having to add some setup procedures to remove some other worse smells.

Administrative

To better manage our development process we have put alot of time into our ticket management (Trello). Each task is broken up into two separate categories either a milestone for the ticket to be completed on, or Grabbag - these are not connected to a required feature and can be grabbed to complete at any time. From there the tasks are labeled into one (or multiple depending on scale) categories:

- **Feature** - Planned Feature that add additional features to the game.
- **Bad Code Smell** - These are bad code smells which must be fixed to implement.
- **Refactor Change** - These are changes which are not bad code smells, are not good.
- **QoL Improvements** - These are Quality of Life Improvements, small items like renaming unclear variable names or moving a method.
- **Administrative** - These are tasks that are not code based and are admin actions.

Plans

A more detailed plan can be viewed on our ticket management system (trello).

Milestone 2

- Add Names to each user and let them select their color.
- Ability to have different maps
- Separation of Game from other game objects using MVC
- Add “Wild Cards”
- Updated User Interface
    - Provide an explanation of the game at the beginning.
    - Provide explanation dialog during the first round of each stage.
    - Provide dialog to indicate switching to the next player's turn.

Milestone 3

- Add “Secret Mission Mode”
    - Will require adding mission cards
- Further Separation of Game and User Interface
- Update User Interface
    - Reorganization play screen to better inform users.
    - Provide information for attack, indicate selected territories, draw a line between.

Milestone 4

- Perform final testing.
- Add additional integration tests.
- Clean up documentation.
- Prepare to deliver the project.

Bad Code Smells for Codebase

(The roughly ~216 unit tests we started with are still all passing in addition to the new ones - many had to be altered to accommodate structural changes to our codebase)

(1) Switch Statement for Differentiate Game Mode Win Checking

Seeing as there are many game modes with unique win conditions, our “hasWon()” method in player would need to keep growing and growing with a switch statement to accommodate any new ones. We decided to remedy this by introducing a strategy pattern, where each player is given an abstract “WinCondition” object with a variety of implementations depending on the game mode being played. With this, the Player object doesn’t know which implementation is being used, and simply calls the “hasWon()” method of the object to see if they’ve won the game somehow.

```
public boolean hasWon() {
    switch (GameManager.getCurrentGameMode()) {
        case GameMode.CAPITAL:
            // Capital risk win condition
        case GameMode.SECRET_MISSION:
            if (MapManager.getInstance().controlsContinents(th
                return true;
            }
        case GameMode.WORLD_DOMINATION:
        default:
            return this.occupiedTerritories.size() == MapManag
    }
}
```

```
public boolean hasWon() {
    return this.winCondition.hasWon();
}
```

```
public abstract class WinCondition {
    2 usages
    Player player;

    2 usages   new *
    public WinCondition(Player playerToCheck) {
        this.player = playerToCheck;
    }

    1 usage   2 implementations   new *
    abstract boolean hasWon();
}
```

(2) Long Method for Game Setup

The main start method for the game, was originally extremely hard to read and very long. Even the methods in the main class it called to get information from the user like player names, colors, maps, and language were not very clean. It was hard to reason about what was going on and as we added more features the start up process this problem become even worse. To fix this issue we created a new class called Setup which handles this setup process, we were then able to remove all other methods, except the main method, from the main class (these methods were cleaned up as well). Each stage of the setup process was then handled by a method inside setup. Additionally, we added the ability to have a development mode which uses defaults for player names, colors, map and lanauges, which makes testing alot easier.

Example Tests: This is with the UI and is not tested by a test bench

```
18      public static void main(String[] args) {
19          System.setProperty("sun.java2d.uiScale", "1");
20          JComboBox<String> selectLanguage = new JComboBox<>();
21          selectLanguage.addItem("en");
22          selectLanguage.addItem("fr");
23          JOptionPane.showMessageDialog(null, selectLanguage,
24                  "Language", JOptionPane.INFORMATION_MESSAGE);
25          String bundleName = "messages_" + selectLanguage.getSelectedItem().toString();
26
27          JComboBox<String> numberOfPlayers = new JComboBox<>();
28          for (int i = 2; i < 7; i++) {
29              numberOfPlayers.addItem(String.valueOf(i));
30          }
31          JOptionPane.showMessageDialog(null, numberOfPlayers,
32                  "Number of Players", JOptionPane.INFORMATION_MESSAGE);
33          ArrayList<Player> players = fillPlayerArray(numberOfPlayers.getSelectedIndex() + 2);
34          JComboBox<String> mapOptions = new JComboBox<>();
35          for (String mapName : World.getMapFiles().keySet())
36              mapOptions.addItem(mapName);
37          JOptionPane.showMessageDialog(null, mapOptions,
38                  "Which map would you like to play?", JOptionPane.INFORMATION_MESSAGE);
39          File mapFile = (File) World.getMapFiles().values().toArray()[mapOptions.getSelectedIndex()];
40          World map = new World(mapFile);
41
42          Game gameController = new Game(numberOfPlayers.getSelectedIndex() + 2, map, players);
43          gameController.setLanguageBundle(bundleName);
44          gameController.initWindow();
45      }
```

```
17      public static void main(String[] args) {
18          if (SETUP_MODE_PRODUCTION) {
19              setupProduction();
20          } else {
21              setDevelopment();
22          }
23      }
24
25
26      private static void setupProduction() {
27          Setup.setupProperties();
28          ResourceBundle bundle = Setup.selectLanguage();
29          World world           = Setup.selectWorld(bundle);
30          int numberOfPlayers   = Setup.selectNumberOfPlayers(bundle);
31          List<Player> players  = Setup.setupPlayers(bundle, numberOfPlayers);
32
33          Game game = new Game(world, players);
34          game.setupUI(bundle, GameView.class);
35          game.begin();
36      }
```

(3) Coordinate Data Class for Territory

The coordinate class was originally a subclass of the world class. This class was returned by the world class as a set of coordinates for each territory button. The issue is that that the coordinate class is just a data class, it has no important logic to it. So this logic was added to the territory class. So when the world generates instead of getting 3 arrays of territories, territory coordinates, and continents; you just get territories and continents - and the map just attaches the coordinates for the button to the territory class. This is especially helpful because we were returning two array territories and coordinates and it was just assumed the coordinates and territories lined up in order, but that is confusing. So, pulling up the coordinates into territory made it less confusing and removed a data class bad code smell.
Example Tests: WorldTest.LoadedCoordinatesBrazil(), WorldTest.LoadedCoordinatesWAustralia(), WorldTest.LoadedCoordinatesScandinavia()

```
21      public static class Coordinate {
22          private final double x;
23          private final double y;
24
25          Coordinate(double x, double y) {
26              this.x = x;
27              this.y = y;
28          }
29
30          public double getX() {
31              return x;
32          }
33
34          public double getY() {
35              return y;
36          }
37      }
```

```
11  public class Territory {
```

```
20      private double posX;
21      private double posY;
```

```
153     public void setPosY(double posY) {
154         this.posY = posY;
155     }
156
157
158     public double getPosX() {
159         return this.posX;
160     }
161
162     public double getPosY() {
163         return this.posY;
164     }
```

(4) Shotgun Surgery Player Setup

The main bad code smell has is a small change but could easily lead to issues down the line. The minimum and maximum player count was not shared acrossed the codebase and was hardcoded in each spot it was required. This was fixed by making these constants public for everyone to access. But there was additional refactoring changes when it comes to player setup - originally player setup is done throughout varieties of different place (4 different classes). This just happened over time because of technical debt as features were added, to

fix this issue all methods related to setting up the player were rolled up into player controller. This way all the code for player setup was all in one place.

Example Tests: GameTest.testDefaultPlayers(), GameTest.testSetPlayer()

```java
public class GameSetup {
    private int armiesPerPlayer;
    private static final int MIN_PLAYERS = 2;
    private static final int MAX_PLAYERS = 6;
    private static final int BASE_ARMIES = 40;
    private static final int EXTRA_PLAYER_MULTIPLIER = 5;

    protected int numberOfPlayers;

    public GameSetup(int numberOfPlayers) {
        this.numberOfPlayers = numberOfPlayers;
    }

    public void setInitialArmies() {
        if (numberOfPlayers >= MIN_PLAYERS && numberOfPlayers <= MAX_PLAYERS) {
            this.armiesPerPlayer = (BASE_ARMIES - ((numberOfPlayers - 2) * EXTRA_PLAYER_MULTIPLIER));
        } else {
            throw new IllegalArgumentException("playerCount must be between 2 and 6");
        }
    }
```

```java
public class Main {
    public static void main(String[] args) {
        System.setProperty("sun.java2d.uiScale", "1");
        JComboBox<String> selectLanguage = new JComboBox<>();
        selectLanguage.addItem("en");
        selectLanguage.addItem("fr");
        JOptionPane.showMessageDialog(null, selectLanguage,
                "Language", JOptionPane.INFORMATION_MESSAGE);
        String bundleName = "messages_" + selectLanguage.getSelectedItem().toString();

        JComboBox<String> numberOfPlayers = new JComboBox<>();
        for (int i = 2; i < 7; i++) {
            numberOfPlayers.addItem(String.valueOf(i));
        }
```

```java
public class Game {
    protected Phase currentPhase;
    private GameView gameView;
    private World world;
    protected TerritoryButton selectedButton;

    protected PlayerController playerController;
    protected TerritoryController territoryController;
    protected ContinentController continentController;

    protected GameSetup gameSetup;
    protected ResourceBundle messages;

    public Game(int numberOfPlayers, World world, ArrayList<Player> players) {
        gameSetup = new GameSetup(numberOfPlayers);
        playerController = new PlayerController(numberOfPlayers, players);
        setupWorld(world);
    }


    public Game(int numberOfPlayers, World map) {
        gameSetup = new GameSetup(numberOfPlayers);
        playerController = new PlayerController(numberOfPlayers, gameSetup.fillPlayerArray(numberOfPlayers));
        setupWorld(map);
    }


    public void setupWorld(World world) {
        gameSetup.setInitialArmies();
        for (Player p: playerController.getPlayerArray()) {
            p.giveArmies(gameSetup.getArmiesPerPlayer());
        }
        this.world          = world;
        territoryController = new TerritoryController(this.world.getTerritories());
        continentController = new ContinentController(this.world.getContinents());
        setFirstPlayer(new Random());
        currentPhase = Phase.territoryClaim;
    }
```

```java
    public void setFirstPlayer(Random r) {
        int startingPlayer = r.nextInt(playerController.getNumberOfPlayers()) + 1;
        playerController.setCurrentPlayer(gameSetup.getPlayerWhoGoesFirst(startingPlayer));
    }
```

```java
public class PlayerController {

    public static final int  MIN_PLAYERS = 2;
    public static final int  MAX_PLAYERS = 6;
    private static final int BASE_ARMIES = 40;
    private static final int EXTRA_PLAYER_MULTIPLIER = 5;

    protected List<Player> players;
    protected int currentPlayer = 0;


    public PlayerController(List<Player> players) {
        if (players.size() >= MIN_PLAYERS && players.size() <= MAX_PLAYERS) {
            this.players = players;
            this.shufflePlayers();
            this.setupArmies();
        } else {
            throw new IllegalArgumentException("Must have between 2-6 Player");
        }
    }


    private void shufflePlayers() {
        Collections.shuffle(this.players);
    }


    private void setupArmies() {
        int amountOfPlayers = this.players.size();
        int armiesPerPlayer = (BASE_ARMIES - ((amountOfPlayers - 2) * EXTRA_PLAYER_MULTIPLIER));
        for (Player player: this.players)
            player.giveArmies(armiesPerPlayer);
    }
```

```java
    public static int selectNumberOfPlayers(ResourceBundle bundle) {
        String unitLabel    = bundle.getString("numberOfPlayersUnit");
        String promptLabel  = bundle.getString("numberOfPlayersPrompt");
        int    minPlayers   = PlayerController.MIN_PLAYERS;
        int    maxPlayers   = PlayerController.MAX_PLAYERS;
        JComboBox<String> promptPlayerCnt = new JComboBox<>();
        for (int i = minPlayers; i <= maxPlayers; i++) {
            promptPlayerCnt.addItem(String.format("%d %s", i, unitLabel));
        }
        JOptionPane.showMessageDialog(null, promptPlayerCnt,
                promptLabel, JOptionPane.INFORMATION_MESSAGE);
        return promptPlayerCnt.getSelectedIndex() + minPlayers;
    }
```

Bad Code Smells for Testing

(1) External dependency to Java Random - Introduce Seam

As we were implementing “Secret Mission Mode” we realized we’d have no way to test much of the code seeing as the target continents given to each player are randomly generated. To solve this, we used dependency injection to introduce a seam in the SecretMissionWin class. With this, we could mock and calls to Java random and guarantee it will return the values we want for our tests.

```
private void generateSecretMission(List<Continent> targetContinents) {
    Random random = new Random();
    while (this.secretTargets.size() < 2) {
        this.secretTargets.add(targetContinents.get(random.nextInt(tar
    }
}
```

```
private final Set<Continent> secretTargets;
public SecretMissionWin(Player playerToCheck, List<Continent> mapContinents, Random random) {
    super(playerToCheck);
    this.secretTargets = new HashSet<>();
    this.generateSecretMission(mapContinents, random);
}
```

```
private void generateSecretMission(List<Continent> targetContinents, Random random) {
    while (this.secretTargets.size() < 2) {
        this.secretTargets.add(targetContinents.get(random.nextInt(targetContinents.si
    }
}
```

(2) Test Chaining for World Setup Singleton

This was an issue happening in the player test. If you ran all the tests it passed however if you ran just one (the second test displayed below) then it failed. This was because of test chaining, which while test chaining was not intended here it’s effects caused errors. The issue is caused because there is a singleton that shared map data with isolated parts of our system. If the map is not declared then it can cause an error. Some tests did not declare the map, but this error was not detected because of the chaining. So the 2nd test would only pass because the test before it passed. This was fixed by before ever test setting the default world map.

```
528         @Test
529         void testPlayerHasWon1() {
530                 new World(World.getMapFiles().get("Earth"));
531                 Player player = new Player(PlayerColor.RED, null, null);
532                 occupyTerritoriesSetup(player, 42);
533                 assertTrue(player.hasWon());
534         }
535
536         @Test
537         void testPlayerHasWon1Integration() {
538                 Player player = new Player(PlayerColor.RED, null, null);
539                 player.giveArmies(42);
540                 Continent continent = EasyMock.mock(Continent.class);
541                 ArrayList<Territory> territories = new ArrayList<>();
542                 for(int i = 0; i < 42; i++) {
543                         Territory t = new Territory("Test", continent);
544                         player.occupyTerritory(t);
545                         territories.add(t);
546                 }
547                 MapManager.getInstance().setTerritories(territories);
548                 assertTrue(player.hasWon());
549         }
```

```
18          @BeforeEach
19          void setupDefaultWorld() {
20                  Setup.defaultWorld();
21          }
```

(3) Test-only Code in Production AttackData Constructor

The attack data class orignally had two constructors. But one of the instructors was only used by the testing system so had no point, and the constructor used in product was never tested by the code. So we removed one of the constructors and updated the code to use the same construtor as the tests. Super easy fix, but now the class is much cleaner and our testing make sense.

```java
public AttackData(Territory attacker, Territory defender, int attackerDice, int defenderDice) {
    this.attacker = attacker;
    this.defender = defender;
    this.attackerDice = attacker.getOccupant().rollDice(attackerDice);
    this.defenderDice = defender.getOccupant().rollDice(defenderDice);
}
public AttackData(Territory attacker, Territory defender, int[] attackerDice, int[] defenderDice) {
    this.attacker = attacker;
    this.defender = defender;
    this.attackerDice = attackerDice;
    this.defenderDice = defenderDice;
}
```

```java
public AttackData(Territory attacker, Territory defender, int[] attackerDice, int[] defenderDice) {
    this.attacker = attacker;
    this.defender = defender;
    this.attackerDice = attackerDice;
    this.defenderDice = defenderDice;
}
```

(4) Too Many Asserts

The test testTradeInNinthAndTenthSets() is trying to test two separate tests being added in and uses a large number of asserts to do so. It has been broken down into two different tests, one for the ninth set and another for the tenth to still verify that the later sets work.

```java
Set<Card> tradeInSet1 = new HashSet<>();
tradeInSet1.add(card1);
tradeInSet1.add(card2);
tradeInSet1.add(card5);

// One-of-each set with a Territory bonus
Set<Card> tradeInSet2 = new HashSet<>();
tradeInSet2.add(card3);
tradeInSet2.add(card4);
tradeInSet2.add(card6);

// Make this the ninth set traded in of the game
cardTrader.numSetsTurnedIn = 8;

// Test turning in the first set
assertTrue(player.tradeInCards(tradeInSet1));
assertEquals( expected: 32, player.getArmiesAvailable());
assertEquals( expected: 3, player.getCards().size());
assertEquals( expected: 9, cardTrader.numSetsTurnedIn);

// Test turning in the second set
// (32 armies from previous for a total of 67)
assertTrue(player.tradeInCards(tradeInSet2));
assertEquals( expected: 67, player.getArmiesAvailable());
assertEquals( expected: 0, player.getCards().size());
assertEquals( expected: 10, cardTrader.numSetsTurnedIn);
EasyMock.verify(territory2, territory3);
}
```

```java
// Make this the ninth set traded in of the game
cardTrader.numSetsTurnedIn = 9;

// Test turning in the second set
// (32 armies from previous for a total of 67)
assertTrue(player.tradeInCards(tradeInSet2));
assertEquals( expected: 37, player.getArmiesAvailable());
assertEquals( expected: 0, player.getCards().size());
assertEquals( expected: 10, cardTrader.numSetsTurnedIn);
EasyMock.verify(territory2, territory3);
}
```

```java
// One-of-each set with a Territory bonus
Set<Card> tradeInSet1 = new HashSet<>();
tradeInSet1.add(card1);
tradeInSet1.add(card2);
tradeInSet1.add(card5);

// Make this the ninth set traded in of the game
cardTrader.numSetsTurnedIn = 8;

// Test turning in the ninth set
assertTrue(player.tradeInCards(tradeInSet1));
assertEquals( expected: 32, player.getArmiesAvailable());
assertEquals( expected: 3, player.getCards().size());
assertEquals( expected: 9, cardTrader.numSetsTurnedIn);
EasyMock.verify(territory2, territory3);
}
```

(5) Duplicate code in tests

The test class TerritoryTest does the same set up for different chunks of code. This makes tests longer and harder to read and unnecessarily adds more lines to code. Now it has been broken down into testing nets (groups) and their respective setup is called before each test using the @BeforeEach function.

```java
@Test
public void testAttackTerritoryAttackerLoses1OrderMatters(){
    Continent continent = EasyMock.mock(Continent.class);
    Territory attacker = new Territory("Test", continent);
    attacker.addArmies(4);
    Territory defender = new Territory("Test", continent);
    defender.addArmies(4);
    int[] attackerRolls = new int[]{5};
    int[] defenderRolls = new int[]{6,3};
    AttackData data = new AttackData(attacker, defender, attackerRolls, defenderRolls);
```

```java
@BeforeEach
public void setupAttackVariables(){
    attacker = new Territory("Test", mockContinent);
    defender = new Territory("Test", mockContinent);
}
```

```java
@Test
public void testAttackTerritoryAttackerLoses1DefenderLoses1OrderMatters() {
    attacker.addArmies(4);
    defender.addArmies(4);
    int[] attackerRolls = new int[]{6, 2, 2};
    int[] defenderRolls = new int[]{3, 4};
    AttackData data = new AttackData(attacker, defender, attackerRolls, defenderRolls);
```

```
@Test
public void testAttackTerritoryDefenderLoses1OrderMatters(){
    Continent continent = EasyMock.mock(Continent.class);
    Territory attacker = new Territory("Test", continent);
    attacker.addArmies(4);
    Territory defender = new Territory("Test", continent);
    defender.addArmies(4);
    int[] attackerRolls = new int[]{6,3};
    int[] defenderRolls = new int[]{4};
    AttackData data = new AttackData(attacker, defender, attackerRolls, defenderRolls);
```

```
@Test
public void testFortifyTerritoryInvalidNumberOfUnits(){
    Continent continent = EasyMock.mock(Continent.class);
    Player red = new Player(PlayerColor.RED, null, null);
    Territory moveFrom = new Territory("Test", continent);
    moveFrom.setOccupant(red);
    Territory moveTo = new Territory("Test", continent);
    moveTo.setOccupant(red);
    moveFrom.addArmies(1);
    moveTo.addArmies(5);
```

```
@Test
public void testFortifyTerritoryInvalidNumberOfUnits2(){
    Continent continent = EasyMock.mock(Continent.class);
    Player red = new Player(PlayerColor.RED, null, null);
    Territory moveFrom = new Territory("Test", continent);
    moveFrom.setOccupant(red);
    Territory moveTo = new Territory("Test", continent);
    moveTo.setOccupant(red);
    moveFrom.addArmies(1);
```

```
@BeforeEach
public void setupFortifyVariables(){
    red = new Player(PlayerColor.RED, null, null);
    moveFrom = new Territory("Test", mockContinent);
    moveFrom.setOccupant(red);
    moveTo = new Territory("Test", mockContinent);
    moveTo.setOccupant(red);
}
```

```
@Test
public void testFortifyTerritoryInvalidNumberOfUnits() {
    moveFrom.addArmies(1);
    moveTo.addArmies(5);

    assertFalse(moveFrom.fortifyTerritory(moveTo, -1));
    assertEquals(1, moveFrom.getArmies());
    assertEquals(5, moveTo.getArmies());
}
```

(6) Long Test Methods

The test methods in TakingCardsIntegrationTest are all really long. Moving some of the setup for them into a single method to run BeforeEach should help to make them more concise. This doesn't look shorter but the code at the top is used in three different test cases.

```
@Test
void testPlayerDefeatsAndTakesTenCards() throws InvalidAttackException {
    Random randomMock = EasyMock.strictMock(Random.class);
    Continent asia = EasyMock.mock(Continent.class);

    Territory attackingTerritory = new Territory( name: "attackingTerritory", asia);
    Territory attackedTerritory = new Territory( name: "attackedTerritory", asia);
    attackingTerritory.addAdjacentTerritory(attackedTerritory);

    CardTrader cardTrader = new CardTrader();

    Player aggressor = new Player(PlayerColor.GREEN, randomMock, cardTrader);
    Player defender = new Player(PlayerColor.BLUE, randomMock, cardTrader);
    aggressor.giveArmies( numArmies: 4);
    defender.giveArmies( numArmies: 1);
    aggressor.occupyTerritory(attackingTerritory);
    aggressor.addArmiesToTerritory(attackingTerritory,  numArmies: 3);
    defender.occupyTerritory(attackedTerritory);

    for (int i = 0; i < 10; i++) {
        EasyMock.expect(randomMock.nextInt( bound: 22)).andReturn( value: 21);
        EasyMock.expect(randomMock.nextInt( bound: 3)).andReturn( value: i % 3);
        EasyMock.expect(randomMock.nextInt(MapManager.getInstance().getTerritories().size())).andReturn( value: i % 2);
    }

    // Attacker rolls
    EasyMock.expect(randomMock.nextInt( bound: 6)).andReturn( value: 1);
    EasyMock.expect(randomMock.nextInt( bound: 6)).andReturn( value: 6);
    EasyMock.expect(randomMock.nextInt( bound: 6)).andReturn( value: 1);
```

```
Territory attackedTerritory;
23 usages
Player aggressor;
15 usages
Player defender;
3 usages
CardTrader cardTrader;
Evan Sellers *
@BeforeEach
void doSetup(){
    Random randomMock = EasyMock.strictMock(Random.class);

    Continent asia = EasyMock.mock(Continent.class);
    attackingTerritory = new Territory( name: "attackingTerritory", asia);
    attackedTerritory = new Territory( name: "attackedTerritory", asia);
    attackingTerritory.addAdjacentTerritory(attackedTerritory);

    cardTrader = new CardTrader();

    aggressor = new Player(PlayerColor.GREEN, randomMock, cardTrader);
    defender = new Player(PlayerColor.BLUE, randomMock, cardTrader);
    aggressor.giveArmies( numArmies: 5);
    defender.giveArmies( numArmies: 2);
    aggressor.occupyTerritory(attackingTerritory);
```

```
    // Defender roll
    EasyMock.expect(randomMock.nextInt( bound: 6)).andReturn( value: 5);

    EasyMock.replay(randomMock);

    // Defender holds 10 cards
    for (int i = 0; i < 10; i++) {
        defender.drawCard();
    }

    int [] attackerRolls = aggressor.rollDice( numberOfDice: 3);
    int [] defenderRolls = defender.rollDice( numberOfDice: 1);
    AttackData data = new AttackData(attackingTerritory, attackedTerritory, attackerRolls, defenderRolls);

    assertEquals( expected: 2, aggressor.attackTerritory(data));
    assertTrue(aggressor.getOccupiedTerritories().contains(attackedTerritory));
    assertEquals( expected: 2, aggressor.getOccupiedTerritories().size());
    assertEquals( expected: 0, defender.getOccupiedTerritories().size());
    assertTrue(defender.hasLost());
    assertEquals( expected: 3, attackingTerritory.getArmies());
    assertEquals( expected: 1, attackedTerritory.getArmies());

    // Attacker took all 10 cards
    assertEquals( expected: 10, aggressor.getCards().size());

    EasyMock.verify(randomMock);
}
```

```
void testPlayerDefeatsAndTakesTenCards() throws InvalidAttackException {
    for (int i = 0; i < 10; i++) {
        EasyMock.expect(randomMock.nextInt( bound: 22)).andReturn( value: 21);
        EasyMock.expect(randomMock.nextInt( bound: 3)).andReturn( value: i % 3);
        EasyMock.expect(randomMock.nextInt(MapManager.getInstance().getTerritories().size())).andReturn( value: i % 2);
    }

    // Attacker rolls
    EasyMock.expect(randomMock.nextInt( bound: 6)).andReturn( value: 1);
    EasyMock.expect(randomMock.nextInt( bound: 6)).andReturn( value: 6);
    EasyMock.expect(randomMock.nextInt( bound: 6)).andReturn( value: 1);

    // Defender roll
    EasyMock.expect(randomMock.nextInt( bound: 6)).andReturn( value: 5);

    EasyMock.replay(randomMock);

    // Defender holds 10 cards
    for (int i = 0; i < 10; i++) {
        defender.drawCard();
    }

    int [] attackerRolls = aggressor.rollDice( numberOfDice: 3);
    int [] defenderRolls = defender.rollDice( numberOfDice: 1);
    AttackData data = new AttackData(attackingTerritory, attackedTerritory, attackerRolls, defenderRolls);
```

```
    assertEquals( expected: 2, aggressor.attackTerritory(data));
    assertTrue(aggressor.getOccupiedTerritories().contains(attackedTerritory));
    assertEquals( expected: 2, aggressor.getOccupiedTerritories().size());
    assertEquals( expected: 0, defender.getOccupiedTerritories().size());
    assertTrue(defender.hasLost());
    assertEquals( expected: 3, attackingTerritory.getArmies());
    assertEquals( expected: 1, attackedTerritory.getArmies());

    // Attacker took all 10 cards
    assertEquals( expected: 10, aggressor.getCards().size());

    EasyMock.verify(randomMock);
```

(7) Randomly Failing Tests - Null Pointer Exceptions in Integration Tests

This is related to our test chaining issue discussed earlier. Out of nowhere, our TakingCardsIntegrationTest test suite started failing multiple tests that were previously not a problem. It turns out that once the order in which tests ran changed randomly, there was an issue with the setup of the earth map happening in the wrong order that could mean the setup wasn't ready for tests which expected it. To fix this, we used the @BeforeAll JUnit annotation to ensure no tests were run in classes that expected world setup before the map was definitely set up.

```
Tests failed: 3, passed: 203 of 206 tests – 1 sec 851 ms

java.lang.NullPointerException Create bre
    at model.TakingCardsIntegrationTes
    at java.base/java.util.ArrayList.
    at java.base/java.util.ArrayList.
```

```
no usages   new *
@BeforeAll
static void setupMap() {
    Setup.defaultWorld();
}
```

```
Tests passed: 205 of 205 tests – 1 sec 904 ms

> Task :compileJava UP-TO-DATE
> Task :processResources UP-TO-D
> Task :classes UP-TO-DATE
> Task :compileTestJava
> Task :processTestResources NO-
```

(8) Sprout Class Used to Make Testing Secret Mission Mode Easier

There were probably more straightforward ways to implement Secret Missions and get it to work, but we chose to abstract out the concept of a "win condition" to make it simpler for our

new tests to target only the new code. Because the Player class always checks if they've won no matter what type of "WinCondition" they're given, we created specific tests for the new class that had no chance of interfering with anything old. With very clearly separate tests for different types of wins, it's extremely clear why tests are failing. The old tests checking for the default gamemode wins didn't need to be changed at all, so the fact that they continued to all pass meant we could refactor safely without needing to worry.

(No "before" picture because we preemptively avoided creating confusing tests with 'Sprout Class')

```
@Test
void testPlayerDoesWinSecretMission1() {
    Player player = new Player(PlayerColor.RED,  random: null,  cardTrader: null);
    player.giveArmies( numArmies: 10);
    List<Continent> testContinents = this.setupContinentsForSecretMissions();
    Random randomMock = EasyMock.strictMock(Random.class);
    EasyMock.expect(randomMock.nextInt(testContinents.size())).andReturn( value: 0);
    EasyMock.expect(randomMock.nextInt(testContinents.size())).andReturn( value: 1);
    EasyMock.replay(randomMock);
    player.setWinCondition(new SecretMissionWin(player, testContinents, randomMock));

    // Player controls both target continents
    player.occupyTerritory((Territory) testContinents.get(0).territories.toArray()[0]);
    player.occupyTerritory((Territory) testContinents.get(0).territories.toArray()[1]);

    player.occupyTerritory((Territory) testContinents.get(1).territories.toArray()[0]);
    player.occupyTerritory((Territory) testContinents.get(1).territories.toArray()[1]);

    assertTrue(player.hasWon());
    EasyMock.verify(randomMock);
}
```

Updated Artifacts

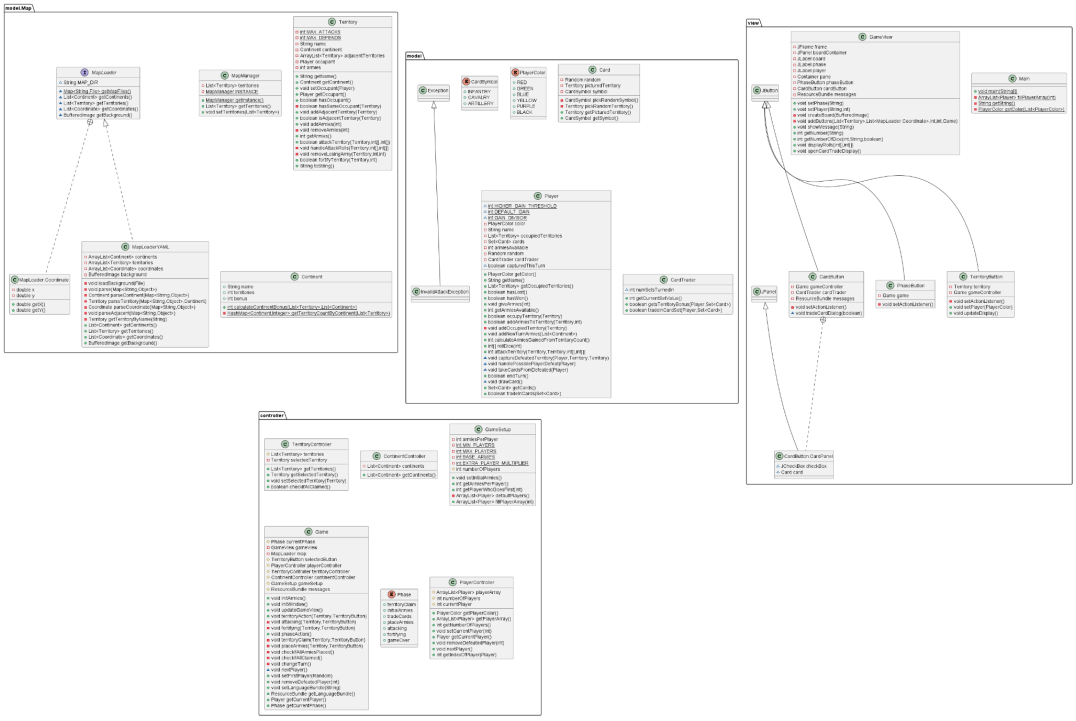
Features (newly implemented features bolded)

F1. Ability to set Number of Players From 2 - 6
F2. Detect when a player wins the game:
- When a player has control of all territories, they win the game (World Domination)
- **When a player captures their secret target continents, they win the game (Secret Mission Mode)**

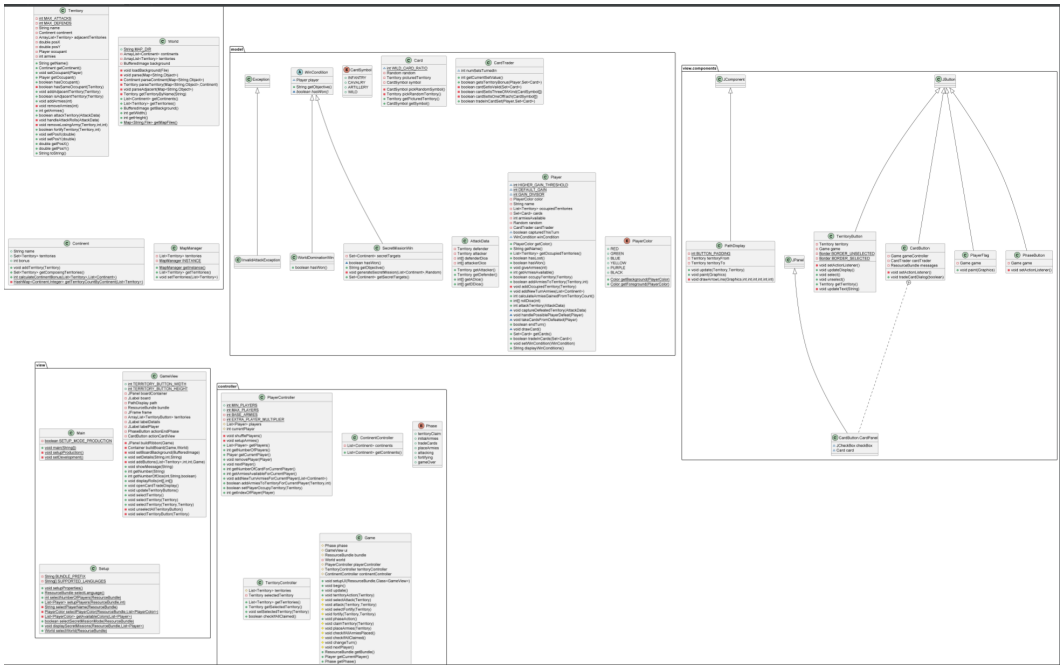
F3. Ability to place armies on territories during start of game
F4. Ability to attack adjacent territories during turn
F5. Ability to move armies to adjacent territories
F6. Ability to draw a card after capturing territory
F7. Ability to trade in cards on turn
F8. Ability to view the state of the map
F9. Detect when a player has lost the game
When a player loses control of all of their territories they lost the game
F10. Ability to gain and place armies at the start of your turn
F11. Set number of armies given at start of game depending on number of players
F12. Ability to take cards from another player when you capture their final territory
F13. Ability to have a choice of multiple, mechanically different maps
- Normal World Map
- **Nonsense World**

F14. Ability to choose your color and name
F15. Ability to select alternative Risk Gamemode - Secret Mission Mode
F16. Ability to view a graphical representation of attacks and fortifications
F17. Ability to view a summary of the Player whose turn it is

OLD M2 UML:



NEW M3 UML:



Key Differences:

- Added WinCondition and SecretMission classes for additional support
- Game Setup Class was removed and each task was allocated to it's appropriate class
- Methods for setup like player name, color, language, and map selection were extracted from the main class and moved into their own class.
- Redesigned UI by adding a DisplayPath and replacing the button system

- UI setup methods and display methods for secret mission
- Added an additional world map