Test Document

Team PI-b

07 April 2019

Name	ID Number
Simon Huang	27067380
Jonathan Massabni	26337430
Alexia Soucy	40014822
Anthony Funiciello	40054110
David Gray	40055149
Mair Elbaz	40004558
Rani Rafid	26975852

1 Introduction

The purpose of this project is to use software engineering practices to develop a desktop application of the Codenames game using intelligent computer players. The software was designed using MVC architecture and several common design patterns. In this last iteration of the project, we will present how we planned to test and have tested the system so far in the development process.

This document provides a rationale for the testing methods used to verify that the requirements were met. It will include a Test Plan that describes what forms of testing we choose to use to test the system. The Test Results section will provide an overview of the tests and outcomes of the tests that were done including each requirement and the percentage of tests passed for it. The References section will include a list of any input and output files' used as test data.

2 Test Plan

The system was developed using TDD or "Test Driven Development" meaning that throughout the development process unit tests were written to test pieces of code before they were written. As a result we test most units of the system using JUnit tests. Because the Codenames game is algorithmically simple, most of the units were tested with black box testing to verify output given certain input. Some of the TDD tests also involve multiple units within a subsystem working together, serving as subsystem level integration tests. We plan to test the system as a whole through other more wholistic JUnit tests, as well as function testing, manually playing the game in order to verify that the use cases outlined in the requirements phase were met.

Because the Codenames game system is a desktop application with very little computation involved, we feel it is unnecessary to do any sort of load or performance testing. We also chose not to designate resources toward user interface, installation, or configuration testing because user-friendliness is not one of the requirements of our project.

Testing will be a joint effort among the team. The Unit and Subsystem level testing is to be done throughout the development process by the developers for each iteration. System level function testing will be done by the whole team after each iteration manually verifying that the use cases implemented in that iteration are met.

The use cases which we tested were:

- Operative chooses a codename
- Operative continues to guess

- Spymaster give a hint
- System places a card on the board based on operative's choice
- User starts game
- User clicks next to trigger next turn
- Game ends when a team wins

We did not explicitly test the following system requirements, as they are inherently tested in the above system level tests.

- System set up board
- System chooses keycard
- Stop guessing (operatives turn ends on wrong choice)
- Stop game

2.1 System Level Test Cases

The first 3 system level test cases were tested with JUnit. The remaining 7 test cases were tested manually with function testing.

Test Case 1: GameManager executes next turn

Purpose
The purpose of this test is to verify that the GameManager in the model executes turns commanded to.

Input Specification

- Create a board and a GameManager to manage the board
- Make two calls to doNextTurn, one for spy, one for operative.

Expected Output

- The board should have 24 (1 less than 25) cards left, after the first operatives turn.

Traces to Use Cases
This test case is intended to test that the function which will be triggered by the user pressing enter does its job. It also tests the use cases for the spymaster and operatives turns.

Test Case 2: Game ends when all of a teams cards have been chosen

Purpose
The purpose of this test is verify that the game ends when all red cards are chosen.
Input Specification

- A list of Blue cards and 1 assassin is created.
- A Board is created with the list of cards
- A GameManager is created with the Board.

Expected Output

- GameManager.isGameOver() returns true as there are no more Red cards to chose

Traces to Use Cases
This test case is intended to test the "game ends when team wins" use case.

Test Case 3: Game ends when assassin is chosen

Purpose
The purpose of this test is verify that the game ends when the assassin card is chosen
Input Specification

- A list of Blue cards and 1 assassin is created.
- A Board is created with the list of cards
- A GameManager is created with the Board.
- A blue card is replaced with a red card
- Assassin is removed

Expected Output

- GameManager.isGameOver() returns true because the assassin was chosen

Traces to Use Cases
This test case is intended to test the "game ends when team wins" use case, in particular that when a team choses the assassin the other team wins.

Test Case 4: Operatives chooses a codename

Purpose
The purpose of this test is to verify that the simulated operative players choose a codename card from the board on their turn.

Input Specification

- Start a game
- Choose a difficulty
- Press enter to have a Spymaster generate a hint.
- Press enter so that an operative takes their turn.

Expected Output

- An operative should choose a card (visible in Verbose mode logs)
- A card on the board should change color

Traces to Use Cases
This test case is intended to test the Operative choose card use case, but also tests that the first spymaster gives a hint, and it tests the system scenarios for start game, and pick keycard.

Test Case 5: Operative continues to guess

Purpose
This test is to verify that the simulated operative players continue to choose a codename card on their turn if their previous guess was valid.

Input Specification

- Start a game
- Choose a difficulty
- Press enter until an operative chooses a card not of the opposite team or assassin

Expected Output

- Operative's turn is still active

- A card on the board should change color
- Pressing enter again results in the operative making another guess

Traces to Use Cases
This test case tests that the first spymaster gives a hint, and it tests the system scenarios for start game, and pick keycard. Primarily it tests that the operatives make multiple guesses when they are able to.

Test Case 6: Spymaster gives a hint

Purpose
This test will verify that Spymaster players give hints.

Input Specification

- Start a game
- Choose a difficulty
- Press enter to have a Spymaster generate a hint.

Expected Output

- The hint is created with a word and number of related words.

Traces to Use Cases
Tests the system start up scenarios, and the spymaster give hint use case.

Test Case 7: System places a card on the board based on operative's choice

Purpose
The purpose of this test is to verify that system properly updates the board based on the card the operative chooses.

Input Specification

- Start a game
- Choose a difficulty
- Press enter to have a Spymaster generate a hint.
- Press enter so that an operative takes their turn.

Expected Output

- The card on the board with the word the operative choose should change color to either red, blue, black, or bystander beige.

Traces to Use Cases
This test case will test if the right card is placed based on the keycard map and operative's choice of the card that was picked. It also tests the use cases for operatives choosing codenames, and spymasters giving a clue.

Test Case 8: User starts game

Purpose
The purpose of this test case is to verify the functionality of the system. Given the proper game settings chosen, it should start a new game with a new set of codenames.

Input Specification

- Launch software
- Choose a difficulty

Expected Output

- Graphical user interface of game board with generated codenames is displayed

Traces to Use Cases
This test case is intended to make sure the core of the game is working. It requires that the software initializes with no errors and that the word lists can be loaded. It also verifies the system picks a keycard.

Test Case 9: User clicks next to trigger next turn

Purpose
This test will check the functionality of the core gameplay by advancing the progress of the game.

Input Specification

- Press enter to have a Spymaster generate a hint.
- Press enter so that an operative takes their turn.

Expected Output

- Next turn will be passed to the other team

Traces to Use Cases
This test case tests the "user clicks next to trigger next turn." It also tests that operatives "stop guessing" and that "spymaster gives a hint."

Test Case 10: Game ends when a team wins

Purpose
This test will check if our game will end the moment a team has been declared the winner.

Input Specification

- Press enter until a victor has been decided

Expected Output

- Either blue or red team wins and game ends

Traces to Use Cases
This test verifies that the requirements of the operatives and spymasters are met. Mainly, it verifies the "stop game" use case.

2.2 Subsystem Level Test Cases

Some of the JUnit tests written during TDD tested multiple units within a subsystem working correctly together. Those tests are documented here, organized by subsystem. As we were instructed to test the model thoroughly, but not the view or controller parts of our system, the subsystems represented here are model.player, and model.board.

Subsystem model.player

- **Test Case 1:** Operative player with random OperativeStrategy makes legal move with function makeMove()
  - Input:
    - The board is initialized
    - Blue OperativePlayer is created with the random strategy.
  - Expected Output: The Operative's makeMove() returns a card from boards.getCards().
- **Test Case 2:** Spymaster player with random strategy makes a non-null clue with a number part between 0 and 9.
  - Input:

    * The board is initialized
    * Blue Spymaster instance is created with random strategy.
  - Expected Output:
    * The Spymasters makeMove() returns a clue that is not null
    * The Spymasters makeMove() returns a clue with number between 0 and 9.

Subsystem model.board

- **Test Case 1:** CardBuilder.buildAll creates a valid board
  - Input:
    * A list of Cards called cards is created with CardBuilder.buildAll()
  - Expected Output:
    * cards has 25 elements
    * no item in cards is null
    * no words are duplicated among the 25 Cards.
- **Test Case 2:** A newly created board has 25 cards.
  - Input:
    * A list of Cards is created with CardBuilder.buildAll()
    * A Board is created with Board(cards)
  - Expected Output: board.getCards() returns a list of 25 Cards.
- **Test Case 3:** Game does not end when there are blue, red, and assassin cards still unchosen.
  - Input:
    * A list of Blue cards and 1 assassin is created.
    * A Board is created with the list of cards
    * A GameManager is created with the Board.
    * A blue card is replaced with a red card
  - Expected Output: GameManager.isGameOver() returns false as there are still red and blue cards remaining on the board to be guessed.
- **Test Case 4:** Red wins after ending game on choice of red card.
  - Input:
    * Create a board with cardBuilder.buildAll()
    * Create a GameManager to manage the board.

    * Create Red Player
  - Expected Output: GameManager.declareWinner() returns CardType.Red when game is over after red player chooses red card.
  - Comment: A similar case was tested, for blue players choosing blue cards

- **Test Case 5:** Red wins after blue chooses assassin or last red card
  - Input:
    * Create a board with cardBuilder.buildAll()
    * Create a GameManager to manage the board.
    * Create Blue Player
  - Expected Output: GameManager.declareWinner() returns CardType.Red when game is over and blue operative choose a red card or assassin card.
  - Comment: A similar case was tested, for blue winning when a red player ends the game after choosing blue or the assassin.

2.3 Unit Test cases

Our unit tests were written throughout the TDD process. Most of the units are tested using black box testing, verifying certain output given an input. Others include white box testing for functions which are more complicated and prone to bugs. We wrote many unit tests including several for methods like getters and setters. As a result not all unit tests we used for testing are included in this document.

RandomSpyStrategy: Function giveClue

Tests will be conducted on giveClue function of RandomSpyStrategy which is supposed to return a valid clue. Because the random Spymasters's functionality does not depend on the input (random) we use a standard board set up.

- **Black Box Testing:**
  - **Test Case 1:** Spymaster doesn't give a null clue.
    * Input:
      · cards=board.getCards()
      · bipartite=Bipartite(Board(cards))
    * Expected Output: Spymaster gives non null Clue object
  - **Test Case 2:** RandomSpyStrategy returns a clue with a valid number of cards (0-9)
    * Input:

        - cards=board.getCards()
        - bipartite=Bipartite(Board(cards))
      - Expected Output: The Spymaster returns a clue number that is within the inclusive range of 0 and 9.

RandomSpyStrategy: Function giveClueRandomly

Tests will be conducted on giveClueRandomly function of RandomSpyStrategy which ensures that the returned clue object is selected at random from the list of clues.

- **Black Box Testing:**
  - **Test Case 1:** RandomSpyStrategy doesn't return the same clue 10 times in a row.
    - Input:
      - cards=board.getCards()
      - bipartite=Bipartite(Board(cards))
    - Expected Output: The 10 calls to giveClue don't return all the same Clue.

SimpleSpyStrategy: Function giveClue

Tests will be conducted on giveClue function of SimpleSpyStrategy which is supposed to return a valid clue object based on the selected word of the Spymaster's team.

- **Black Box Testing:**
  - **Test Case 1:** Blue Spymaster gives a clue associated with one of their words.
    - Input:
      - cards=[Card(BELL, CardType.Blue), Card(TORCH, CardType.Red)]
      - bipartite=Bipartite(Board(cards))
    - Expected Output: Blue Spymaster should return one of: CHIME, GONG, BUZZER, DOOR
  - **Test Case 2:** Red Spymaster gives a clue associated with one of their words.
    - Input:
      - cards=[Card(BELL, CardType.Blue), Card(TORCH, CardType.Red)]
      - bipartite=Bipartite(Board(cards))
    - Expected Output: Red Spymaster should return one of: FLASHLIGHT, LIGHT, LANTERN, BLOWLAMP
  - **Test Case 3:** Spymaster's clue is not null

        * Input:
            · cards=board.getCards()
            · bipartite=Bipartite(Board(cards))
        * Expected Output: The returned clue is not null.

SmartSpyStrategy: Function giveClue

Tests will be conducted on giveClue function of SmartSpyStrategy which is supposed to return the most optimal clue object. The optimal clue should be associated with more than one word if possible and it will return the clue with the most occurrence.

- **Black Box Testing:**
    - **Test Case 1:** SmartSpyStrategy give the clue associated with both of their cards.
        * Input:
            · cards=[Card(LONDON, CardType.Blue), Card(ATLANTIS, CardType.Blue)]
            · bipartite=Bipartite(Board(cards))
        * Expected Output: Blue Spymaster should return one of: CITY

RandomOperativeStrategy: Function pickCard

Tests the pickCard function of RandomOperativeStrategy to ensure that the operative picks a card randomly.

- **Black Box Testing:**
    - **Test Case 1:** Random operative does not pick the same card 10 times in a row.
        * Input: Generate a random board as in a regular game.
            · cards=board.getCards()
            · bipartite=Bipartite(Board(cards))
        * Expected Output: The Operative does not pick the same card 10 times in a row.

BotOperativeStrategy: Function pickCard

Tests will be conducted on pickCard function of BotOperativeStrategy to ensure that when set to perfect accuracy, the operative picks their teams card.

- **Black Box Testing:**

  - **Test Case 1:** Blue bot with accuracy=1 picks blue card.
    - Input:
      - cards=board.getCards()
      - bipartite=Bipartite(Board(cards))
    - Expected Output: Blue operative picks blue card.
  - **Test Case 2:** Red bot with accuracy=1 picks blue card.
    - Input:
      - cards=board.getCards()
      - bipartite=Bipartite(Board(cards))
    - Expected Output: Red operative picks blue card.

HumanOperativeStrategy: Function pickCard

Tests will be conducted on pickCard function of HumanOperativeStrategy to ensure that the human player can pick a card.

- **Black Box Testing:**
  - **Test Case 1:** Human player picks the correct card
    - Input:
      - A game board containing 1 card
    - Expected Output: Human player picks the single card

CardType: Function charOf

Tests will be conducted on charOf function of CardType to ensure that a character corresponds to the correct enum value.

- **Black Box Testing:**
  - **Test Case 1:** The character 'R' returns Red
    - Input:
      - CardType.charOf('R');
    - Expected Output: returns CardType.Red
  - **Test Case 2:** The character 'B' returns Blue
    - Input:
      - CardType.charOf('B');
    - Expected Output: returns CardType.Blue
  - **Test Case 3:** The character 'A' returns Assassin

    - Input:
        - CardType.charOf('A');
    - Expected Output: returns CardType.Assassin
- **Test Case 4:** The character 'Y' returns Bystander
    - Input:
        - CardType.charOf('Y');
    - Expected Output: returns CardType.Bystander

Board: Function removeCard

The removeCard function will be tested to ensure that a removed card is no longer available on the board.

- **Black Box Testing:**
    - **Test Case 1:** Removing the first Card results in the card not being available for selection again.
        - Setup: A board is initialized with cards made from CardBuilder.buildAll().
        - Input: Card=board.getCards().get(0);
        - Expected Output: The board does not contain the removed card anymore

Board: Function getNumCardsOfType

Tests will be conducted on getNumCardsOfType function of class Board to ensure that the board contains the right number of cards for each player type and each team. For all of these test cases the setup is the same. A board is initialized with cards made from CardBuilder.buildAll().

- **Black Box Testing:**
    - **Test Case 1:** Number of red cards in initialized board
        - Input: type=CardType.Red
        - Expected Output: 9 or 8.
    - **Test Case 2:** Number of blue cards in initialized board
        - Input: type=CardType.Blue
        - Expected Output: 8 if there are 9 red cards, 9 if there are 8 blue cards.
    - **Test Case 3:** Number of Assassin cards in initialized board
        - Input: type=CardType.Assassin
        - Expected Output: 1

  - **Test Case 4:** Number of Bystander cards in initialized board
    * Input: type=CardType.Bystander
    * Expected Output: 7

Board: Function get

Tests will be conducted on the boards get(int index) function to verify that an IndexOutOfBoundsException is thrown when the function is called with an invalid index.

- **Black Box Testing:**
  - **Test Case 1:** Get card at index 25 throws exception
    * Input: 25
    * Expected Output: IndexOutOfBoundsException thrown

Card: Function toString

Tests will be conducted on toString function of Card to ensure that the card's text will be able to cast to a string.

- **Black Box Testing:**
  - **Test Case 1:** toString returns the cards codename word
    * Input: toString called on (new Card("word", cardTypeBystander))
    * Expected Output: "word"

Clue: Function getClueWord

Tests will be conducted on getClueWord function of Clue to ensure that the word used to create the object is returned. erult.s

- **Black Box Testing:**
  - **Test Case 1:** getClueWord returns word
    * Input: getClueWord called on (new Clue("Clue", 3))
    * Expected Output: "Clue"

Clue: Function getClueNum

Tests will be conducted on getClueNum function of Clue to ensure that the returned clue number is correct.

- **Black Box Testing:**
  - **Test Case 1:** getClueNum returns number associated with clue
    - Input: getClueNum called on (new Clue("Clue", 3))
    - Expected Output: 3

Extractor: Function build

Tests will be conducted on build function of Extractor to ensure that the word bank can be successfully created.

- **Black Box Testing:**
  - **Test Case 1:** Words file read successfully
    - Input: path="resources/words100_1550871908_SYN_1550898480.json"
    - Expected Output: returns list without throwing FileNotFoundException
  - **Test Case 2:** List of string words created
    - Input: path="resources/words100_1550871908_SYN_1550898480.json"
    - Expected Output: returns non null list

GameManager: Function isTurnOver

Tests will be conducted on isTurnOver function of GameManager to check if the turn has ended with the correct conditions.

- **Black Box Testing:**
  - **Test Case 1:** Red chooses a blue card
    - Input:
      - player=Operative(CardType.Red, ...)
      - card=Card("card", CardType.Blue)
      - clueNum=0
    - Expected Output: True
  - **Test Case 2:** Red chooses a red card
    - Input:
      - player=Operative(CardType.Red, ...)
      - card=Card("card", CardType.Red)
      - clueNum=0
    - Expected Output: False

- **Test Case 3:** Red chooses an assassin
  - Input:
    - player=Operative(CardType.Red, ...)
    - card=new Card("card", CardType.Assassin)
    - clueNum=0
  - Expected Output: True
- **Test Case 4:** Red chooses a bystander
  - Input:
    - player=Operative(CardType.Red, ...)
    - card=Card("card", CardType.Bystander)
    - clueNum=0
  - Expected Output: True
- **Test Case 5:** Blue chooses a red card
  - Input:
    - player=Operative(CardType.Red, ...)
    - card=new Card("card", CardType.Red)
    - clueNum=0
  - Expected Output: True
- **Test Case 6:** Blue chooses a blue card
  - Input:
    - player=new Operative(CardType.Blues, ...)
    - card=new Card("card", CardType.Blue)
    - clueNum=0
  - Expected Output: False
- **Test Case 7:** Blue chooses a bystander
  - Input:
    - player=Operative(CardType.Blue, ...)
    - card=new Card("card", CardType.ByStander)
    - clueNum=0
  - Expected Output: True
- **Test Case 8:** Blue chooses an assassin
  - Input:
    - player=Operative(CardType.Blue, ...)
    - card=new Card("card", CardType.Assassin)
    - clueNum=0
  - Expected Output: True

GameManager: Function gameIsOver

Tests will be conducted on gameIsOver function of GameManager to ensure that the function can determine whether the game is over or not.

- **Black Box Testing:**
  - **Test Case 1:** Game is over because all reds cards are chosen
    - Input:
      - A board containing exactly 1 assassin card and 9 blue cards
    - Expected Output: The game is over
  - **Test Case 2:** Game is not over because the board contains the assassin card, at least one blue card, and at least one red card.
    - Input:
      - A board containing exactly 1 assassin card, 8 blue cards, and 1 red card.
    - Expected Output: The game is not over
  - **Test Case 3:** Game is over because the assassin card is selected
    - Input:
      - A board containing exactly 2 red cards and 8 blue cards but no assassin card.
    - Expected Output: The game is over

GameManager: Function declareWinner

Tests will be conducted on declareWinner function of GameManager to ensure that the function can determine the winner.

- **Black Box Testing:**
  - **Test Case 1:** red is determined the winner for having selected his last required card
    - Input:
      - The current player that selected a card
      - Card object that player selected
    - Expected Output: Red player has been declared the winner
  - **Test Case 2:** Blue is determined the winner for having selected his last required card
    - Input:

        - The current player that selected a card
        - Card object that player selected
      - Expected Output: Blue player has been declared the winner
  - **Test Case 3:** red is determined the winner because Blue player selected Red teams last card
    - Input:
      - Blue player that selected a card
      - Card object that player selected
    - Expected Output: Red player has been declared the winner
  - **Test Case 4:** Blue is determined the winner because Red player selected Blue teams last card
    - Input:
      - player that selected a card
      - Card object that player selected
    - Expected Output: Blue player has been declared the winner
  - **Test Case 5:** red is determined the winner because Blue team selected Assassin Card
    - Input:
      - The current player that selected a card
      - Assassin card object selected
    - Expected Output: Red player has been declared the winner
  - **Test Case 6:** blue is determined the winner because Red team selected Assassin Card
    - Input:
      - The current player that selected a card
      - Assassin card object selected
    - Expected Output: Blue player has been declared the winner

GameManager: Function doNextTurn

Tests will be conducted on doNextTurn function of GameManager to ensure that the number of cards on the board updates are a turn is made.

- **Black Box Testing:**
  - **Test Case 1:** There is no next turn due to the game being over
    - Input:
      -

        * Expected Output: System outputs the Message "Game Over"
    - **Test Case 2:** It is the spymaster's turn
        * Expected Output: The Spymaster performs his turn
    - **Test Case 3:** It is the Operative's turn
        * Expected Output: The Operative performs his turn

GameManager: Function endHumanTurn

Tests will be conducted on endHumanTurn function of GameManager to ensure that the human's turn ends.

- **Black Box Testing:**
    - **Test Case 1:** The return value of endHumanTurn() is false.
        * Input:
            · Create a board with CardBuilder.buildAll();
            · Create a GameManager to manage the board
            · Call endHumanTurn() method to GameManager
        * Expected Output: endHumanTurn() returns false

GameManager: Function getBlueScore

Tests will be conducted on getBlueScore function of GameManager to ensure that the function returns the correct blue score.

- **Black Box Testing:**
    - **Test Case 1:** Returns the score for Blue Team
        * Expected Output: The score of Blue team

GameManager: Function getRedScore

Tests will be conducted on getRedScore function of GameManager to ensure that the function returns the correct red score.

- **Black Box Testing:**
    - **Test Case 1:** Retrieve the score for Red Team
        * Expected Output: The score for red team

GameManager: Function getWinner

Tests will be conducted on getWinner function of GameManager to ensure that the function returns the winner.

- **Black Box Testing:**
  - **Test Case 1:** Red team is the winner
    - Expected Output: returns Red CardType.
  - **Test Case 2:** Blue team is the winner
    - Expected Output: returns Blue CardType.

KeyCard: Function parse

Tests will be conducted on parse function of KeyCard to ensure that the generated keycards are valid for the codenames game rules.

- **Black Box Testing:**
  - **Test Case 1:** KeyCard.parse() returns valid keyCard.
    - Input: parse() takes no parameters
    - Expected Output:
      - Test Case 1: List returned is not null
      - Test Case 2: List contains 25 keycards
      - Test Case 3: Number of assassins is 1
      - Test Case 4: Number of bystanders is 7
      - Test Case 5: Number of red cards is 8 or 9
      - Test Case 6: Number of blue cards is 8 or 9

3 Test Results

3.1 System Level Test Results

The percentage of tests passing for the requirements which we tested are:

- **100%**: GameManager executes next turn
- **100%**: Game ends when all of a teams cards have been chosen
- **100%**: Game ends when assassin is chosen

- **100%**: Operative chooses a codename
- **100%**: Operative continues to guess
- **100%**: Spymaster give a hint
- **100%**: System places a card on the board based on operative's choice
- **100%**: User starts game
- **100%**: User clicks next to trigger next turn
- **100%**: Game ends when a team wins

3.2 Subsystem Level Test Results

- model.player subsystem
  - **PASS** Test Case 1: Operative player with random OperativeStrategy makes legal move with function makeMove()
  - **PASS** Test Case 2: Spymaster player with random strategy makes a non-null clue with a number part between 0 and 9.
- model.board subsystem
  - **PASS** Test Case 1: CardBuilder.buildAll creates a valid board
  - **PASS** Test Case 2: A newly created board has 25 cards
  - **PASS** Test Case 3: Game does not end when there are blue, red, and assassin cards still unchosen
  - **PASS** Test Case 4: Red wins after ending game on choice of red card.
  - **PASS** Test Case 5: Red wins after blue chooses assassin or last red card

3.3 Unit Test Results

- RandomSpyStrategy: Function giveClue
  - **PASS** Test Case 1: Spymaster doesnt give a null clue.
  - **PASS** Test Case 2: RandomSpyStrategy returns a clue with a valid number of cards.
- RandomSpyStrategy: Function giveClueRandomly
  - **PASS** Test Case 1: RandomSpyStrategy doesnt return the same clue 10 times in a row.

- SimpleSpyStrategy: Function giveClue
  - **PASS** Test Case 1: Blue Spymaster gives a clue associated with one of their words.
  - **PASS** Test Case 2: Red Spymaster gives a clue associated with one of their words.
  - **PASS** Test Case 3: Spymasters clue is not null.
- SmartSpyStrategy: Function giveClue
  - **PASS** Test Case 1: SmartSpyStrategy give the clue associated with both of their cards.
- RandomOperativeStrategy: Function pickCard
  - **PASS** Test Case 1: Random operative does not pick the same card 10 times in a row.
- BotOperativeStrategy: Function pickCard
  - **PASS** Test Case 1: Blue bot with accuracy=1 picks blue card.
  - **PASS** Test Case 2: Red bot with accuracy=1 picks blue card.
- HumanOperativeStrategy: Function pickCard
  - **PASS** Test Case 1: Human player picks the correct card
- CardType: Function charOf
  - **PASS** Test Case 1: The character 'R' returns Red
  - **PASS** Test Case 2: The character 'B' returns Blue
  - **PASS** Test Case 3: The character 'A' returns Assassin
  - **PASS** Test Case 4: The character 'Y' returns Bystander
- Board: Function removeCard
  - **PASS** Test Case 1: Removing the first Card results in the card not being available for selection again.
- Board: Function getNumCardsOfType
  - **PASS** Test Case 1: Number of red cards in initialized board
  - **PASS** Test Case 2: Number of blue cards in initialized board

  - **PASS** Test Case 3: Number of assassin cards in initialized board
  - **PASS** Test Case 4: Number of bystander cards in initialized board

- Board: Function get
  - **PASS** Test Case 1: Get card at index 25 throws Exception

- Card: Function toString
  - **PASS** Test Case 1: : toString returns the cards codename word

- Clue: Function getClueWord
  - **PASS** Test Case 1: getClueWord returns word.

- Clue: Function getClueNum
  - **PASS** Test Case 1: getClueNum returns number associated with clue.

- Extractor: Function build
  - **PASS** Test Case 1: Words file read successfully
  - **PASS** Test Case 2: List of string words created

- GameManager: Function isTurnOver
  - **PASS** Test Case 1: Red chooses a blue card
  - **PASS** Test Case 2: Red chooses a red card
  - **PASS** Test Case 3: Red chooses an assassin
  - **PASS** Test Case 4: Red chooses a bystander
  - **PASS** Test Case 5: Blue chooses a red card
  - **PASS** Test Case 6: Blue chooses a blue card
  - **PASS** Test Case 7: Blue chooses a bystander
  - **PASS** Test Case 8: Blue chooses an assassin

- GameManager: Function isGameOver
  - **PASS** Test Case 1: Game is over because all reds cards are chosen
  - **PASS** Test Case 2: Game is not over because the board contains the assassin card, at least one blue card, and at least one red card.
  - **PASS** Test Case 3: Game is over because the assassin card is selected

- GameManager: Function declareWinner
  - **PASS** Test Case 1: red is determined the winner for having selected his last required card
  - **PASS** Test Case 2: Blue is determined the winner for having selected his last required card
  - **PASS** Test Case 3: red is determined the winner because Blue player selected Red teams last card
  - **PASS** Test Case 4: Blue is determined the winner because Red player selected Blue teams last card
  - **PASS** Test Case 5: red is determined the winner because Blue team selected Assassin Card
  - **PASS** Test Case 6: blue is determined the winner because Red team selected Assassin Card
- GameManager: Function doNextTurn
  - **PASS** Test Case 1: There is no next turn due to the game being over
  - **PASS** Test Case 2: It is the spymaster's turn
  - **PASS** Test Case 3: It is the Operative's turn
- GameManager: Function endHumanTurn
  - **PASS** Test Case 1: The return value of endHumanTurn() is false.
- GameManager: Function getBlueScore
  - **PASS** Test Case 1: Returns the score for Blue Team
- GameManager: Function getRedScore
  - **PASS** Test Case 1: Returns the score for Red Team
- GameManager: Function getWinner
  - **PASS** Test Case 1: Red team is the winner
  - **PASS** Test Case 2: Blue team is the winner
- KeyCard :: Function parse
  - **PASS** Test Case 1: List returned is not null
  - **PASS** Test Case 2: List contains 25 keycards

- **PASS** Test Case 3: Number of assassins is 1
- **PASS** Test Case 4: Number of bystanders is 7
- **PASS** Test Case 5: Number of red cards is 8 or 9
- **PASS** Test Case 6: Number of blue cards is 8 or 9

4 References

A Description of Input Files

No input files used as the user selects the game settings through an initial setup screen.

B Description of Output Files

No output files were used. Logs were examined when needed to verify that tests pass. The system will display information through a game board in a graphical user interface. It will also create a log of all the events and a real-time score board to display to the user.