Testing Plan

Immense Coverage
The test suite provided covers a wide range of functionalities in the system. It includes tests for:

Constructing and linking map elements.
Counting territories owned by a player.
Assigning troops.
Validating the game state transitions.
Simulating game actions (e.g., attacking, fortifying).
Validating player and country interactions.
By covering different aspects of the system, the tests ensure that various functionalities work correctly both in isolation and together.

Manual Testing

A large amount of our testing was conducted manually, to ensure that user inputs and the game loop functioned well. This included making sure the GUI displayed the correct values, making sure the game accepted good values and did not crash on bad values, and playing through the game extensively.

Boundary and Edge Cases
The tests include checks for edge cases and boundary conditions, such as:

Checking the handling of zero or maximum numbers of territories owned by a player.
Validating the placement of troops and their movement.
Testing interactions between neighboring and non-neighboring countries.
These tests help to ensure that the system behaves correctly in extreme or unexpected scenarios, which are often sources of bugs.

By systematically verifying each unit of functionality, checking for edge cases, validating state transitions, and using assertions to compare actual and expected outcomes, the testing approach provides strong evidence that the system works correctly. Regular and automated testing further ensures that the system remains correct even as it evolves.

The OUnit test suite covers focuses on the Map, Player, and Turn modules.

1. Map Module:
construct_map_element
add_link
Read_lines

Testing Approach:

Glass Box Testing: We know the internal structure and implementation of these functions, so we can directly test specific aspects of their behavior.
Boundary Testing: Tests like test_map_size and test_read_lines ensure that the functions handle expected inputs correctly, including the correct number of elements and non-empty results.

2. Player Module:
num_terrs_owned
troops_from_cont
troops_on_start
player_has_territories
place_troops
subTroops
addTroops
print_rules
claim_countries
Auto_assign

Testing Approach:
Black Box Testing: We focus on the inputs and outputs without considering the internal workings. For example, test_num_terrs_owned checks the number of territories owned by a player after setting up the map state, and assigning all countries to player 0.

Glass Box Testing: In some cases, knowing the internals helps us set up specific scenarios and validate expected outcomes, such as in test_place_troops where we check the number of troops placed.
Boundary Testing: Testing functions like troops_from_cont and player_has_territories involves ensuring that the edge cases (e.g., no territories owned, all territories owned) are handled correctly.

3. Turn Module:
num_terrs_owned
troops_from_cont
troops_on_start
player_has_territories
place_troops
valid_num_troops
get_valid_country
roll_dice
compare_rolls
move_troops
attack
fortify

Testing Approach:
Glass Box Testing: (Example) In test_attack, we simulate an attack and validate the changes in player and troop counts.
Boundary Testing: Ensuring that functions like valid_num_troops handle edge cases (e.g., placing maximum or minimum troops) correctly.

Conclusion

The test cases were developed using a combination of testing methodologies Black Box, Glass Box, Boundary Case, and Manual Testing. Together, these methods provided a robust test suite that allowed us to ensure that our game functioned well