System Verification and Validation Plan for SE 4G06 An AI-based Approach to Designing Board Games

Team #6, Board Gamers
Ilao Michael, ilaom
Bedi Hargun, bedih
Dang Jeffrey, dangj12
Ada Jonah, karaatan
Mai Tianzheng, mait6

April 6, 2023

1 Revision History

Date	Version	Notes
November 2nd, 2022	1.0	All Team Members Contributed
April 5th, 2023	2.0	Jeffrey Dang - Final Revision

Contents

List of Figures

2 Symbols, Abbreviations and Acronyms

symbol	description
V&V or V&V Plan	Refers to this Validation & Verification Plan document

This document will provide a road map of the Verification and Validation plan for the documents, implementation and the finished system.

3 General Information

3.1 Summary

BoardGamers are developing an AI-based, game simulation engine that can help identify pitfalls in a game's mechanics and find imbalances in the game's scoring system. The three main sub-systems in our system are:

- **AI Agent:** An AI model trained to play the game at hand acting as one of the players of the game to give insights into the possible paths to be taken. It would analyze all possible paths that a player can take at any point in the game. It would take as input the Game State information (current state and available moves for the player) and output the next move that the player will take.
- **Game Engine:** Abstract representation of the actual game as software. Contains all the game rules, Observation Space (state of the game), and Action Space (possible moves and commands that a player can choose from). It is also responsible for validating input from the AI Agent and outputting the Game State with the moves available for the player.
- **Data Visualizer:** Read the Game Engine and AI Agent log, and provide diverse graphic representations of data and information of moves to Game State transitions. It will take the output from both the AI Agent and Game Engine to provide a way for the user to decipher imbalances in the game.

3.2 Objectives

The primary objectives for this ~~project~~ document will be:

- **Demonstrate software correctness**: A focus on software correctness is important as the stakeholders expect to be able to use AI for prototyping the board games, thus it is important to provide accurate data as it can have a dramatic outcome on the final product of

the board game if stakeholders make a crucial decision based on the information produced by the system.

- **Ensure maintainability**: As the modules of the system will be implemented through an iterative process, ensuring high maintainability makes the implementation of new features for the Game Engine easy for the developers.
- **Build around interoperability**: The system is structured around the main modules, the Game Engine and the AI Agents, so building around interoperability is important to keep the system functional.
- **Sustain software modularity**: After the implementing and integrating the modules for Game Engine and the AI, decoupling each of the modules to the utmost allow for different AI Agents and the Game Engine to be exchanged when testing and comparing the viability of AI Agents.

3.3 Relevant Documentation

The relevant documentations for this report are our SRS, MIS, MG and Hazard Analysis documents. We will be using these documents as a reference for planning and executing the plan specified in this report.

Relevant Links:

- SRS: https://github.com/Dorps/aiboardgame/blob/main/docs/SRS/SRS.pdf
- MIS: https://github.com/Dorps/aiboardgame/blob/main/docs/Design/MIS/MIS.pdf
- MG: https://github.com/Dorps/aiboardgame/blob/main/docs/Design/MG/MG.pdf
- Hazard Analysis: https://github.com/Dorps/aiboardgame/blob/main/docs/HazardAnalysis/HazardAnalysis.pdf

4 Plan

This section will introduce the plans for verifying the SRS, Design Verification and Validation, Implementation, Automated Testing and External Software. This section will also introduce the team responsible for verification and validation.

4.1 Verification and Validation Team

Team Member	Role	Role Description
Jeffrey Dang	Developer	Responsible for verification of functional requirements related to the AI Agent
Michael Ilao	Developer	Responsible for verification of functional requirements related to the AI Agent
Hargun Bedi	Developer	Responsible for verification of functional requirements related to the architecture simulation
Tianzheng Mai	Developer	Responsible for verification of functional requirements related to the data visualizer
Jonah Ada	Developer	Responsible for verification of functional requirements related to the architecture simulation
Dr. Vladimir Reinharz	Supervisor	Responsible for final verification of AI Game Agent performance and analysis of game logs
Dr. Sebastien Mosser	Supervisor	Responsible for final verification of architecture and simulation

Roles are subject to change based on the development progress and developers are able to swap to assist in development in other components of the project as see fit.

4.2 SRS Verification Plan

- For Functional requirement verification in SRS, we will provide the software access to Dr.Sebastien and Dr.Vladimir so that they can cus-

tomize the game input, run it multiple times, and verify if our software satisfies all requirements listed in the SRS.

- For non-Functional requirements verification in SRS, we plan to invite a group of test users to use our software. After they test the software, we will hold an informal interview and ask them a series of structural questions about the usability and use experience. We will collect and compare their answers to verify whether the software successfully satisfies all non-functional requirements.

4.3 Design Verification Plan

- Classmates in SFWR 4G06 will be invited to review the design documents and create feedback and issues in the GitHub repo.

- The team will hold a review session with Dr. Sebastien and Dr. Vladimir after the project is fully completed. During the review session, Dr.Sebastien will go over the design document and verify whether all features in the design documents have been successfully implemented.

4.4 Verification and Validation Plan

V&V Plan will be validated in 3 stages:
1st Stage - Team Members Validations: Each team member will review the V&V Plan and validate each test case and the rest of the document.
2nd Stage - V&V Feedback From Other Groups: Our team will review the issues created on GitHub by other groups to validate the document based on the feedback of other groups.
3rd Stage - V&V Assignment Feedback From the TA: The last stage for validating the V&V Plan will be to consider the feedback of the TA and revise the plan based the feedback and any possible follow-up questions and discussion with the TA.

4.5 Implementation Verification Plan

There will be multiple verification sources to verify the correctness of the implementation. These stages will be:

1st Stage - Unit Tests: Each group member will write unit tests (outlined or will be outlined in this document) to verify their particular piece of code (function, module, subsystem etc.) works as expected. There will be suite of unit tests throughout the project to be run anytime there is a major update during the implementation phase.
2nd Stage - Subsystem Tests: Each group member will test their entire subsystem while developing and at the end of the implementation of the specific subsystem to make sure there are no errors when integrating the modules and other parts of the subsystem.
3rd Stage - Integration Tests: Group as a whole will test the system throughout the implementation process when they integrate different modules or subsystems to make sure different modules will work as expected when integrated together.
4th Stage - Code Reviews: There will be two types of code reviews:

- **Peer Review:** Team members will provide feedback to each others code after each module is being completed. This will ensure consistency throughout the code base and will catch errors that the author of the code may have missed.
- **Review From Supervisors:** The group will also have code review sessions with supervisors to verify the implementation. These code reviews can be formal meetings or informal quick ~~question~~ questions and answers through communication channels such as Discord and Microsoft Teams.

4.6 Automated Testing and Verification Tools

The main tool that will be used to automated testing will be PyTest and unnittest part of the Standard Python Library. For testing coverage Coverage.py will test code coverage of the system, testing coverage will be only used on the Game Engine as the AI Agent's cannot be unit tested easily. Pylint and Prettier will be used to lint the code to keep a consistent style. Finally performance will be measure with timeit Python library. A more detailed outline of the automated tools can be found in the Development Plan

4.7 Software Validation Plan

The system can be externally compared to the physical rules given by the Game Designer and verification can occur when comparing Game States when certain actions are applied. The game rules will dictate the correctness of the software by seeing if the expected Game State was produced after an action is applied to an already existing Game State. Furthermore, for more in-depth verification, the Game Designer can review the system to verify if the simulation is a valid representation of the physical copy of the board game.

5 System Test Description

5.1 Tests for Generalized Functional Requirements

This section will be for the generalized version of the tests as this architecture can be applied to any general board game.

5.1.1 Area of Testing AI Agents

AI Agent is one of the three main subsystems of the software system and its correctness is crucial to the success of the system. Thus, there will be extensive testing to verify the AI Agent modules. The tests in this subsection will verify the AI Agent's functional requirements FR1 to FR6.

Please refer to the traceability matrix to see which tests are related to which functional requirements.

FR Test 1
Control: Manual
Initial State: It is AI's turn and AI Agent has no information about the state of the game.
Input: State of the game is passed from the Game Engine.
Output: The AI Agent should not output anything if there is no error. Should output an error message if the input type is not as expected.

Test Case Derivation: Passing the Game State is not a request from the Game Engine but rather a response to AI Agent's request for Game State. Thus, there shouldn't be any output except for the case that there is a mismatch between the expected input format and the actual input (input format is not specified here because it will change based on the game that is being balanced and the AI library used to develop that particular AI Agent).
~~**How test will be performed:**~~**How the test will be performed:** Start the simulation and make sure the right of play is on AI Agent for that round. When passed the state look for the response and make sure the response code is true.

FR Test 2
Control: Manual
Initial State: There is an AI Agent in the system that can play the game.
Input: New AI model to replace the existing AI Agent.
Output: When the AI Agent is swapped, there is no specific output expected.
Test Case Derivation: Replacing the code for two AI Agents will not produce any output without running the simulation first.
How the test will be performed: However, to check whether the new AI Agent is compatible with the system, run the simulation and check whether the new AI Agent outputs an action from the Action Space or gives an error to finalize the test and determine the whether the requirement is met or not.

FR Test 3
Control: Manual
Initial State: The simulation is running and it is AI Agent's turn to play.
Input: AI Agent takes the Game State as an input.
Output: AI Agent outputs an action from the Action Space and logs its decision.
Test Case Derivation: The AI Agent needs to be observable, so it should always outputs the logs of its decision.
How the test will be performed: Run the simulation and wait AI Agent to play at least one turn. Then check the data visualization manual to see whether the AI Agent output any decision logs. If yes, the test passes, if not test fails.

FR Test 4
Control: Automatic
Initial State: The simulation is running and it is AI Agent's turn to play.
Input: AI Agent takes the Game State as an input.
Output: AI Agent outputs an action from the Action Space.
Test Case Derivation: The AI Agent always needs to output a valid action from the Action Space.
How the test will be performed: Run the simulation and wait AI Agent to play at least one turn. Then system automatically checks the AI Agents output. If the output is from the Action Space then the test passes, if not test fails.

FR Test 5
Control: Manual
Initial State: The simulation is running and it is AI Agent's turn to play.
Input: AI Agent takes its Observation Space as an input.
Output: AI Agent outputs an action from the Action Space with only the information that the AI Agent's Observation Space and not with the information from the global Game State.
Test Case Derivation: The AI Agent always needs to output a valid action from the Action Space but AI Agent should have limited knowledge of the Game State based on its Observation Space to ensure the fact that AI Agent will replicate a human player.
How the test will be performed: Run the simulation and wait AI Agent to play at least one turn. Then check the Observation Space that the AI Agent receives and make sure it is the Observation Space that is specific to the AI Agent and not the global Game State.

FR Test 6
Control: Automatic
Initial State: The simulation is running and it is AI Agent's turn to play.
Input: AI Agent takes the Game State as an input.
Output: AI Agent outputs an action from the Action Space.

Test Case Derivation: The AI Agent always needs to output a valid action from the Action Space to make sure that the simulation can keep going and not be stuck waiting AI Agent to output an action.
How the test will be performed: Run the simulation and wait AI Agent to play at least one turn. Then system automatically checks the AI Agents output. If the output is from the Action Space then the test passes, if there is no action in the output then the test fails.

5.1.2 Area of Testing Game Engine

The bulk of unit tests will be to ensure the Game Engine is working correctly. If the Game Engine is not working as intended and there are bugs and errors in the logic, the AI Agents will not be able to learn efficiently and correctly. This subsection will cover all the functional requirements related to the Game Engine that is functional requirements FR7 to FR11.

FR Test 7
Control: Automatic
Initial State: Game has not been initialized
Input: Initialize (Start) the game
Output: Start of Game State is initialized (ex: Board is created, cards are distributed to players, etc...) will vary based on Game Engine currently in system
Test Case Derivation: Before a game can be played by the AI Agents (or human players), some setup is required to properly play the game.
How the test will be performed: Test will be performed by initializing the game and before any players can make their turn, the Game State is checked to ensure all conditions are met for the specific game.

FR Test 8
Control: Automatic
Initial State: Game is a turn before a winning/end state
Input: AI Agent makes a move to progress the game into an end state.
Output: The expected result is that the game should end and players are unable to make any more moves.

Test Case Derivation: A game should be complete-able or the AI Agents will play forever, also the AI Agents learn from a reward given for certain moves, if they are unable to reach the winning/end state they cannot learn. This test will need to be performed on all actions that can lead to an end state, if there are 9 different ways to win a test will be provided for each one.
How the test will be performed: The game will initialized with a modified state, that where taking a certain move will end the game and having the player win. The AI does not specifically need to make that move right away, but when they do the test needs to check if the Game Engine is in the end state and no moves are able to be played.

FR Test 9
Control: Automatic
Initial State: Game is in any state after starting the game
Input: AI Agent makes a specific move to progress the game into a new state.
Output: After receiving input from the AI Agent for a specific move, the Game State should change accordingly.
Test Case Derivation: Each different move a AI Agent can make should modify the Game State accordingly. The Game Engine must ensure that each move works as expected.
How the test will be performed: The test will be performed by having the AI Agent performing all moves given a certain state and verifying that the Game State changes accordingly.

FR Test 10
Control: Automatic
Initial State: Game is in any state after being initialized
Input: AI Agent requests the action state (move list).
Output: The move list must be consistent throughout the simulation and not change depending on the Game State.
Test Case Derivation: For the AI Agent's to learn their output actions must be the same throughout the simulation or there will be inconsistencies in their learning.
How the test will be performed: The test will be performed by requesting the move list, making a move, getting the new state, then requesting the

move list again. The two lists should be the exact same.

FR Test 11
Control: Automatic
Initial State: Game is in a specific single state
Input: AI Agent makes a move from the Action Space, that has no random component to it.
Output: The Game State must change accordingly to the move given. This will be repeated multiple times and the Game State must be the same after each move given the same starting state.
Test Case Derivation: This test is to ensure the Action Space is consistent across moves given the same move and same state must progress the state to a new state, that is the same every time where the action has no randomness to it.
How the test will be performed: The test will be run multiple times on a single specific given state, the same action will be taken and the new resulting Game State must be the same across all tests.

FR Test 12
Control: Automatic
Initial State: Game is in any state after being initialized
Input: AI Agent provides an invalid input to the Game Engine.
Output: The Game Engine must validate the input and output the appropriate error.
Test Case Derivation: This test case is used to validate the Game Engine's and AI Agent's ability to communicate with each other. It is essential for the two system to talk to each other in a standardized and predictable manner and any invalid input must be checked.
How the test will be performed: The test will be performed by having the AI Agent give invalid input (e.g, incorrect arguments or incorrect types) and the Game Engine must validate and output an appropriate error (different errors will be shown for different types of invalid inputs).

FR Test 13
Control: Automatic

Initial State: Game is in any state after being initialized
Input: AI Agent provides an illegal move.
Output: The Game Engine must move to the end state.
Test Case Derivation: This test case is used to validate the Game Engine's ability to discourage invalid moves for the AI Agent. It is essential for the AI Agent to learn to give only legal moves to the Game Engine such that the game can move forward.
How the test will be performed: The test will be performed by having the AI Agent input illegal moves that lead the game to the end state given a certain state where the game is not near the end state. All the illegal moves must move the game directly to the end state without executing any part of the move.

5.1.3 Area of Testing Data Visualizer

The tests in this subsection tend to ensure the system can obtain correct data points from the AI Agents and Game Engines and output different charts to visualize the data for users. It will cover all the functional requirements related to the game data visualization analysis that is function requirements from FR12 to FR15.

FR Test 14
Control: Automatic
Initial State: Win condition is reached in Game Engine
Input: Information log from Game Engines and AI Agent
Output: The information log is read and collected
Test Case Derivation: The system will read the information log and provide correct data points that can be used in the data visualizer.
How the test will be performed: The test will be performed by reading the information log from Game Engines and AI Agents after the win condition is reached. If the information log is successfully read from the system, it will pass the correct data points value to the data visualizer.

FR Test 15
Control: Automatic
Initial State: The information log is read and collected

Input: Select target data points
Output: Generate either a selection success message or failure message
Test Case Derivation: After the system confirmed the correct selection, it will prepare the selected data points for data visualization.
How the test will be performed: The test will be performed by checking the validity of the data point selection. Valid selection will prepare data points for visualization, and invalid selection will return a failure message to request the correct option.

FR Test 16
Control: Manual
Initial State: The system finishes selecting target data points.
Input: Initialize the data visualizer
Output: The system will output a variety of data visualization charts.
Test Case Derivation: The data visualizer will provide different graphic representations of data such as pie charts, bar graphs, and line charts which efficiently show insights into the data.
How the test will be performed: The data visualizer will receive selected data points input and generate different data visualization charts for the users. The charts translate the data points into expected visual contexts. The test will verify whether the data visualizer translates the data points to the expected visual contexts that users can easily understand.

FR Test 17
Control: Automatic
Initial State: Data visualization process completed
Input: selected data points
Output: evaluation output from the data comparison
Test Case Derivation: The system will compare different data points and analyze an optimal solution for the user.
How the test will be performed: The test will be performed by comparing multiple data points input and ~~return~~ returning an optimal solution to the users. The test will be run multiple times to verify if the evaluation output is the most optimal solution.

5.2 Tests for Tic-Tac-Toe Functional Requirements

This section is an example of how these tests can be specified under a specific board game, in this case, it is demonstrated on Tic-Tac-Toe. Area of Testing Data Visualizer will be redundant and is not included in this section.

5.2.1 Area of Testing AI Agents

Please refer to the traceability matrix to see which tests are related to which functional requirements.

FR Test 1
Control: Manual
Initial State: It is AI's turn and AI Agent has no information about the state of the game.
Input: State of the game is passed from the Game Engine.
Output: The AI Agent should not output anything if there is no error. Should output an error message if the input type is not as expected.
Test Case Derivation: Passing the Game State is not a request from the Game Engine but rather a response to AI Agent's request for Game State. Thus, there shouldn't be any output except for the case that there is a mismatch between the expected input format and the actual input (input format is not specified here because it will change based on the game that is being balanced and the AI library used to develop that particular AI Agent).

How the test will be performed: Start the simulation and ensure the right of play is on AI Agent for that round. When passed the state looks for the response and make sure the response code is true.

FR Test 2
Control: Manual
Initial State: There is an AI Agent in the system that can play the game.
Input: New AI model to replace the existing AI Agent.
Output: When the AI Agent is swapped, there is no specific output expected.

Test Case Derivation: Replacing the code for two AI Agents will not produce any output without running the simulation first.
How the test will be performed: However, to check whether the new AI Agent is compatible with the system, run the simulation and check whether the new AI Agent outputs an action on to the tic-tac-toe board or gives an error to finalize the test and determine the whether the requirement is met or not.

FR Test 3
Control: Manual
Initial State: The simulation is running and it is AI Agent's turn to play.
Input: AI Agent takes the tic-tac-toe game environment as an input.
Output: AI Agent outputs an action on an existing Tic-Tac-Toe space and logs its decision.
Test Case Derivation: The AI Agent needs to be observable, so it should always outputs the logs of its decision.
How the test will be performed: Run the simulation and wait AI Agent to play at least one turn. Then check the data visualization manual to see whether the AI Agent output valid moves on the Tic-Tac-Toe board. If yes, the test passes, if not test fails.

FR Test 4
Control: Automatic
Initial State: The simulation is running and it is AI Agent's turn to play.
Input: AI Agent takes the Game State as an input.
Output: AI Agent outputs an action from the Action Space.
Test Case Derivation: The AI Agent always needs to output a valid action from the Action Space.
How the test will be performed: Run the simulation and wait AI Agent to play at least one turn. Then system automatically checks the AI Agents output. If the output is valid in the context of Tic-Tac-Toe (no previously marked squares, within the 3x3 board, etc.) then the test passes, if not test fails.

FR Test 5

This test is redundant in the game of Tic-Tac-Toe, but can apply to board games with hidden information between players.
Control: Manual
Initial State: The simulation is running and it is AI Agent's turn to play.
Input: AI Agent takes its Observation Space as an input.
Output: AI Agent outputs an action from the Action Space with only the information that the AI Agent's Observation Space and not with the information from the global Game State.
Test Case Derivation: The AI Agent always needs to output a valid action from the Action Space but AI Agent should have limited knowledge of the Game State based on its Observation Space to ensure the fact that AI Agent will replicate a human player.
How the test will be performed: Run the simulation and wait AI Agent to play at least one turn. Then check the Observation Space that the AI Agent receives and make sure it is the Observation Space that is specific to the AI Agent and not the global Game State.

FR Test 6
Control: Automatic
Initial State: The simulation is running and it is AI Agent's turn to play.
Input: AI Agent takes the Tic-Tac-Toe environment as an input.
Output: AI Agent outputs an action valid in the context of Tic-Tac-Toe.
Test Case Derivation: The AI Agent always needs to output a legal move to make sure that the simulation can keep going and not be stuck waiting for AI Agent or looping in Tic-Tac-Toe as the game can go on forever if players are able to play moves on already marked squares.
How the test will be performed: Run the simulation and wait AI Agent to play at least one turn. Then system automatically checks the AI Agents output. If the output is valid in the context of Tic-Tac-Toe then the test passes, if there is no action in the output then the test fails.

5.2.2 Area of Testing Game Engine

FR Test 7
Control: Automatic
Initial State: Game has not been initialized
Input: Initialize Tic-Tac-Toe game environment

Output: Tic-Tac-Toe board has been initialized with all positions empty
Test Case Derivation: Before a game can be played by the AI Agents (or human players), some setup is required to properly play the game.
How the test will be performed: Test will be performed by initializing the Tic-Tac-Toe board and then checking if all positions are available to be filled by the AI Agents.

FR Test 8
Control: Automatic
Initial State: Tic-Tac-Toe game state is left where there's only one legal move left on the board
Input: AI Agent makes the final legal move left on the Tic-Tac-Toe board.
Output: The expected result is that the game should end and players are unable to make any more moves afterwardss.
Test Case Derivation: A game should be complete-able or the AI Agents will play forever, also the AI Agents learn from a reward given for certain moves, if they are unable to reach the winning/end state they cannot learn. This test will need to be performed on all actions that can lead to an end state.
How the test will be performed: The game will be initialized with state where all but one position on the grid is marked, that where taking the last move will end the game. The AI does not specifically need to make that move right away, but when they do the test needs to check if the Game Engine is in the end state and no moves are able to be played.

FR Test 9
Control: Automatic
Initial State: Game is in any state after starting the game
Input: AI Agent makes a specific move to progress the game into a new state.
Output: After receiving input from the AI Agent for a position on the Tic-Tac-Toe board, the board should fill up the same position and modify the board
Test Case Derivation: Each different move an AI Agent can make should modify the Tic-Tac-Toe board accordingly. The Game Engine must ensure that each move works as expected.

How the test will be performed: The test will be performed by having the AI Agent perform all moves on an empty board space at least once.

FR Test 10
Control: Automatic
Initial State: Game is in any state after being initialized
Input: AI Agent requests the possible positions to be played.
Output: The possible positions to be played must be consistent throughout the simulation and not change depending on the board state.
Test Case Derivation: For the AI Agent's to learn their available output actions must be the same throughout the simulation or there will be inconsistencies in their learning.
How the test will be performed: The test will be performed by requesting the possible positions list, making a move, getting the new state, then requesting the possible positions lists. The two lists should be the exact same.

FR Test 11
Control: Automatic
Initial State: Game is in a specific single state
Input: AI Agent plays a legal position on the Tic-Tac-Toe baord
Output: The Game State should be filled with the legal position played This will be repeated multiple times and the board must be the same after each move given the same starting state.
Test Case Derivation: This test is to ensure the Action Space is consistent across moves given the same move and same state must progress the state to a new state, that is the same every time where the action has no randomness to it.
How the test will be performed: The test will be run multiple times on a single specific given state, the same action will be taken and the new resulting Game State must be the same across all tests.

FR Test 12
Control: Automatic
Initial State: Tic-Tac-Toe board game environment is initialized
Input: AI Agent plays a position non-existent on the Tic-Tac-Toe board.

Output: The Game Engine must validate the input and output the appropriate error.
Test Case Derivation: This test case is used to validate the Game Engine's and AI Agent's ability to communicate with each other. It is essential for the two system to talk to each other in a standardized and predictable manner and any invalid input must be checked.
How the test will be performed: The test will be performed by having the AI Agent give invalid input (e.g, incorrect arguments or incorrect types) and the Game Engine must validate and output an appropriate error (different errors will be shown for different types of invalid inputs).

FR Test 13
Control: Automatic
Initial State: Tic-Tac-Toe board game environment is initialized
Input: AI Agent plays a position non-existent on the Tic-Tac-Toe board.
Output: The Game Engine must move to the end state.
Test Case Derivation: This test case is used to validate the Game Engine's ability to discourage invalid moves for the AI Agent. It is essential for the AI Agent to learn to give only legal moves to the Game Engine such that the game can move forward.
How the test will be performed: The test will be performed by having the AI Agent input illegal position moves on the board that lead the game to the end state given a certain state where the game is not near the end state. All the illegal moves must move the game directly to the end state without executing any part of the move.

5.3 Tests for Nonfunctional Requirements

NFR Test 1
Type: Manual
Initial State: AI model is untrained.
Input: Learning data from running the simulation at least 1000 times.
Output: After each 100 times of running the simulation, AI model should improve its win rate.
How the test will be performed: Run the simulation at least 1000 times with an untrained AI model. After each 100 simulation, record the win rate of the AI model and compare them to other data points. At the end of the

1000 simulation, AI model should improve its win rate.

NFR Test 2
Type: Manual
Initial State: AI Agent is trained and system is ready to be used for a new user.
Input: Start command for a new simulation.
Output: An entire logs of AI Agent outputs for a full game simulation.
How the test will be performed: After explaining the system in 30 minutes or less to a new user, user should be able to run the simulation by himself and get the entire output log for a full game simulation.

NFR Test 3
Type: Manual
Initial State: Game Engine is implemented and AI Agents are trained.
Input: A trained new AI Agent replaced with one of the pre-existing AI Agents in the system.
Output: The system should generate logs of AI Agent and be able to run the simulation the same as it would have with the pre-existing AI Agents integrated into the system.
How the test will be performed: Run the simulation with the pre-existing AI Agents and keep AI logs, then replace the pre-existing AI with the newly trained AI Agent and rerun the simulation. Compare the logs and the run times for both simulations to determine any oddities in the newly trained data to determine if it has been a successful replacement.

NFR Test 4
Type: Manual
Initial State: Our system has never been installed on a machine running macOS Monterey and newer.
Input: N/A
Output: Our system installs properly and atleast 1 simulation is correctly executed on the machine.
How the test will be performed: We will try installing all our files necessary for the system on a machine with macOS Monterey and newer installed

on it. If the installation is successful and at least 1 simulation is able to run on it then we treat this test case as passed, otherwise it will fail.

NFR Test 5
Type: Manual
Initial State: Our system has never been installed on a machine running Windows 10 and newer.
Input: N/A
Output: Our system installs properly and ~~atleast~~ at least 1 simulation is correctly executed on the machine.
How the test will be performed: We will try installing all our files necessary for the system on a machine with Windows 10 and newer installed on it. If the installation is successful and at least 1 simulation is able to run on it then we treat this test case as passed, otherwise, it will fail.

NFR Test 6
Type: Manual
Initial State: Our system has never been installed on one of Aliance Canada's Computer Cluster machines.
Input: N/A
Output: Our system installs properly and ~~atleast~~ at least 1 simulation is correctly executed on the machine.
How the test will be performed: We will try installing all our files necessary for the system on one of ~~Aliance~~ Alliance Canada's Computer Cluster machines. If the installation is successful and at least 1 simulation is able to run on it then we treat this test case as passed, otherwise, it will fail.

NFR Test 7
Type: Automatic
Initial State: Game is initialized with maximum number of players
Input: N/A
Expected Output: The simulation takes less than 5 minutes.
How the test will be performed: Hypothetically, the simulation with maximum number of players should take the longest as there will be many more aspects that the AI Agent will need to consider for every decision when

compared to the minimum number of players required. If the simulation takes more than 5 minutes then this test case is a fail, otherwise it will be marked as a pass.

NFR Test 8
Type: Manual
Initial State: Our system have never implemented additional rules and mechanics
Input: N/A
Expected Output: The system extends additional rules and mechanics after 2 days of development time
How the test will be performed: We will try to implement new features and change our algorithms that extends additional rules and mechanics in two days of development time. If all additional rules and mechanics are successfully implemented into our software within these 2 day, we will treat this test case as passed, otherwise it will fail.

NFR Test 9
Type: Manual
Initial State: The system finishes selecting target data points.
Input: N/A
Expected Output: Clear visualization charts in different colors, lines, and shapes.
How the test will be performed: We will manually run the system multiple times and observe whether the Data Visualizer outputs clear graphic representation of data in different colors, lines and shapes. ~~If the output charts are clear in different colors, lines, and shapes,~~ Using the survey in the Appendix, if the user rates readable of the data above a 3, we will treat this test case as passed, otherwise it will fail.

5.4 Traceability Between Test Cases and Requirements

	FR Test 1	FR Test 2	FR Test 3	FR Test 4	FR Test 5	FR Test 6	FR Test 7	FR Test 8	FR Test 9	FR Test 10	FR Test 11	FR Test 12	FR Test 13	FR Test 14	FR Test 15	FR Test 16	FR Test 17
FR1	x																
FR2		x															
FR3			x														
FR4				x													
FR5					x												
FR6						x											
FR7												x	x				
FR8								x									
FR9										x							
FR10							x	x	x								
FR11										x	x						
FR12																x	
FR13														x			
FR14															x		
FR15																	x

Figure 1: Functional Requirements and Test Case Traceability Matrix

	NFR Test 1	NFR Test 2	NFR Test 3	NFR Test 4	NFR Test 5	NFR Test 6	NFR Test 7	NFR Test 8	NFR Test 9
NR1	x								
NR2		x							
NR3			x						
NR4				x	x	x			
NR5							x		
NR6								x	
NR7									x

Figure 2: Non-Functional Requirements and Test Case Traceability Matrix

6 Unit Test Description

~~Unit tests will be filled after the system design phase is complete~~~~Please ignore the entire section 6 until the design phase is complete and V&V plan is revised.~~Note: JSON Object array refers to an object array with all data from input file, or any modified object array that is return from the the following functions: getAllData, getAllDataExEnd, getDataWithMergedActions. Excluding the first test, all other functions help with **FR15 Test**, and **FR16 Test**.

Test ID	Function Name	Inputs	Output
DV1	getAllData	None	JSON object array of the file generated from the Game Engine module
DV2	getAllData- ExEnd	None	JSON object array with only the data needed for the graphs
DV3	getDataWith- MergedActions	None	JSON object array that merges the action and action_details fields into one
DV4	getSimulation- Data	JSON Object Array, Number of Simulations	Array of Simulations
DV5	getPlayerData	Object with 1 Simulation data, Player ID	Object with Simulation data for the specified player
DV6	getFrequency- Map	None	Object Array with each object having action name and frequency set to 0

DV7	getCountMap	None	Object Array with each object having action name and count set to 0
DV8	getFrequencyMapFor- Player	JSON Object Array with merged actions, Number of Simulations, Player ID	Frequency Map Object for the specified player for the specified simulations
DV9	getCountMapFor- Player	JSON Object Array with merged actions, Number of Simulations, Player ID	Count Map Object for the specified player for the specified simulations
DV10	getAllNonZeroAct- ions	Frequency/Count Map of a Player	Frequency/Count Map that only has non-zero frequencies/counts
DV11	getScores	JSON Object Array with merged actions, Number of Simulations	Object with final scores data for the number simulations specified
DV12	getPlayers	JSON Object Array	Array with sequence of Player IDs

DV13	getSimulations	JSON Object Array	Array with sequence of simulation IDs
DV14	getNumberOfMoves	JSON Object Array with merged actions, Number of Simulations, Player ID	Number of moves for the player specified for the number of simulations specified
DV15	getMap	JSON Object Array with merged actions, Number of Simulations, Player ID	Frequency map for specified player's actions for the number of simulations specified, 2D array of action names where each index is the move number

7 Appendix

7.1 Symbolic Parameters

N/A

7.2 Usability Survey Questions

- After explanation from the developers, how confident are you able to navigate the system on scale of 1-5? (5 - very confident)
- How readable is data from the Data Visualizer from a scale 1-5? (5 - easily readable)
- How easy is it to filter to the wanted data points from the Data Visualizer? (5 - very easy)

Appendix — Reflection

The skills required to complete verification and validation for the project will be specific library experience in PyTest and unnittest, dynamic testing knowledge, and static testing knowledge.
For library experience in PyTest and unnittest will be crucial in dealing with tests requiring automated testing to give us immediate feedback without manual interaction for cases like FR 6 and FR Test 14. Without the automated testing libraries, it would take too much time to test manually.
Dynamic testing knowledge is essential to understand as mentioned before, we will use an automated testing protocol for some of the tests to make it much easier to confirm if core aspects of the system are still functional during iterative implementation.
Static testing knowledge will be required to commit to code reviews to ensure that coding principles are kept throughout the implementation or additional features after the initial implementation has been completed.
The PyTest and unnittest can be learned through either video tutorials or using starter code examples to learn from. The group will begin with starter code examples as we all have had prior experience with these libraries already, if we need more assistance opt to use video tutorials.
Dynamic testing knowledge can be learned through the same suit as the Python libraries are automated testing libraries.
Static testing knowledge can be obtained through online sources or by gaining experience through our capstone supervisors. The group as a whole shall gain knowledge through the means of our capstone supervisors because it is a very lucrative resource that we may not always have access to and the supervisors are knowledgeable on the topic of our project. As a last resort, online resources are always available for us to research.