Validation and Verification Rev2 - A World Apart

Team Santa Hates Poor Kids

Jim Wu, 001409055
Ian Yang, 001217664
Gabriel Castagner, 001412885
Junhao Wang, 001215428

Supervised by:
Dr. Jacques Carette
Software Engineering 4GP6

1. Introduction

This document contains all of the automated unit and system tests for the next installment of “A World Apart.” Along with this document will have instructions on how to execute the automated tests and will list all tests that have the their corresponding requirements partially or completely implemented in the game as of March 9th 2018. The reader should reference the Requirements Document Revision 2.1 for all functional requirements used in for the automated tests. These functional requirements include Functional Requirements, Primary Gameplay Requirements, Alternate Gameplay Requirements and Menu and Other Systems Requirements.

2. Terminology

The following are acronyms and frequently used terms are used throughout the document are listed below to increase clarity for the reader. Please review them if the functional requirements or tests seem unclear as to what they are referencing.

Overworld: The main map view that has the connection of nodes.
Combat mode: The phase that is associated with combat and fighting.
Grid: This refers to the map’s square view that is populated by N by N cells.
Cells: This refers to the current location that an object, character, or enemy may be located on in the grid. The name of the cell refers to the name of the object, character or enemy that populates it and is otherwise null.

3. Issue Tracking

This section covers functional requirements that have been added, modified or removed because they either were redundant, duplicated or forgotten during previous versions of the requirement documentation. These documents will be simply listed that identified as the numbering has been related to the previous revision of the document. You can find all of the changed and added functional requirements in the Requirements Document Revision 2.1

3.1 Removed Requirements

Core Requirements

1. Nodes will not cross each others paths.

PGR Requirements

1. When it is the AI's turn, the AI shall be able to move the chosen character.
2. When the player has moved the chosen character, they maybe be able to select one of the four actions.

3. If the player loses the game, they are given a graphic indication and are returned to the main menu. The le is deemed as unplayable.

3.2 Modified Requirements

FR Requirements

1. If the player selects the exe file, a new build of the game is created and the game is initialized. -> If the player selects the exe file, the game is initialized from it's previous build.

PGR Requirements

1. If the player wins the game, they are shown the winning graphical indication and are returned to the main menu. The le is deemed as unplayable. -> If the player wins or loses the game, they are shown the respected graphical indication and are returned to the main menu. The le is deemed as unplayable.

3.3 Added Requirements

Core Requirements

1. The Combat map will be created with the correct N by N size.

PGR Requirements

1. When it is the AI's turn, the AI shall be able to select a multitude of actions to perform to best situate itself given the subsequent criteria.
2. The player and the AI can decide either to attack the health values or the armour values of their respective enemies.
3. The Player may not be able to interrupt the AI's turn while the AI is still performing it's turn.

AGR Requirements

1. Items will be listed in a grid system with respected image and border colour to indicate item type.
2. When items are selected, their information will be displayed to the screen.
3. If items are of the consumable type, the use of this item will remove the item from play and the player's item list.
4. If items are of the expendable type, the use of this item will decremented until they reach 0, where they will be refilled when the combat phase ends.
5. If items are of the rechargeable type, the use of the item will not allow the player to use the item again until the number of cooldown turns been completed.
6. If items are of the passive type, the item applies it's ability every turn for the course of the combat phase. These items are not "used" like other items.
7. If items are of the instant type, the item can be used for every turn of the assigned character.
8. If an item is equipped to character, no other characters are able to equip that same instance of the item.
9. If the mouse is hovering over a desired item and the item changes the stats of a character, those stats will be updated on the selected character with incremented values being coloured in blue, decremented stats being coloured in red and unchanged values being coloured in white.

MR Requirements

1. The player should be able to access the the Item's menu from the Overworld menu by clicking the respective button or the respected hotkey.

3.4 Untested Requirements

Due to the nature of current implementation status. Certain aspects of the requirements cannot be tested right now.

Regions

- Requirements regarding regions cannot be tested beyond the lack of region assignments because the system for region transition (saving and loading the overworld,and party persistency are not fully implemented yet)

Saving and Loading Overworld

- Requirements dealing with the saving and loading of the overworld cannot be tested at the moment due to a technical issue with the non-serializability of GameObjects. This means an alternate structure must be manually developed to save the position and attributes of the party, nodes (and their states), clutter placement and Tilesets of the overworld. This is still in development as of now.

Final Boss Encounters

- Final bosses have not been developed for the testing region as of yet, thus, currently it is impossible to defeat it. Without this feature, tests regarding certain win conditions related to finishing the region cannot be tested as of yet.

Advanced Merchant Features

- Beyond basic rudimentary menu features for the merchant, supply exchange is still being developed, testing with respect to this feature cannot be done without full implementation, and will most likely require manual testing. This is due to the fact that automated testing usually cannot go beyond opening the menu. Simulating user interactions without directly calling menu functions is infeasible currently.

4. Automated System Tests

The following section will cover System testing. Each system test will have an associated script(s) in Unity. These tests put together multiple functions to test a relevant part of the user experience. Each test will have a related user scenario or requirement that justifies the need for the test. Tests for functionality not yet implemented will assert false, but will be provided a process in which they will be tested once said functionality is implemented. Tests for functionality not yet implemented will assert false. Any requirements that included equipping items or equipment, interacting with the merchant, final bosses, overall game win/lose conditions, node logic functionality and displaying tooltips have not been included and can be treated as an assertFalse. The development team has not implemented these requirements yet and must still determine how these requirements will be implemented.

4.1 World System Tests

World System Test 1(advance testing)
Test Name: NodeLoadsCombat()
Related Requirement:
PGR33. If the combat phase is triggered, the player is moved to the Combat phase with current equipment layouts and resources.
CMR7. Each node should not deadlock the character with a negative outcome that makes them lose the game. (This test ensures the players can reach the first combat node.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->SystemTest->NodeLoadsCombat
Description: This test automates the process of the player travelling from the starting node to the nearest combat node**.** Pathing is dynamically calculated and player clicks are simulated using debug functions.

World System Test 2(advance testing)
Test Name: NodeRepeatDeath()
Related Requirement:
PGR22. A player may travel to any node that is connected to the current node if they have the correct amount of resources.
PGR27. The game is lost of the player runs out of the supply resource, unable to move.
PGR27. The game is lost of the player runs out of the supply resource, unable to move.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->SystemTest->NodeRepeatDeath()
Description: This test automates the process of the player travelling between two nodes over and over again in order to exhaust the supplies needed to travel. Player clicks are simulated using the debug

functions. The initial supply has been artificially lowered to keep the test within Unity's 30 seconds timeout period.

4.2 Combat System Tests

Combat System Test 1
Test Name: Player_HasFirst_Turn()
Related Requirement: PGR1 - The player is always has the first turn.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->GridTests-> Player_HasFirst_Turn
Description: This test checks to see if when a new combat phase begins, one of the characters player number is equal to the CurrentPlayerNumber which should be zero.
Process:
Start test combat stage
Check if first turn player is player 1 (the player character)

Combat System Test 2
Test Name: Turns_Alternate()
Related Requirement: PGR2 - The combat phase's turns happen on individual character basis.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->GridTests-> Turns_Alternate
Description: This test checks to see that when a character end's their turn, that the next CurrentPlayerNumber is not equal to the character that had previously finished their turn.
Process:
Start test combat stage
End Turn
Check current player is not player 1 (the player character)

Combat System Test 3
Test Name: Player_HasNoControlWhen_NotItsTurn()
Related Requirement: PGR10 - The Player may not be able to interrupt the AI's turn while the AI is still performing its turn.
PGR3 - Turn player cannot be interrupted
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->GridTests-> Player_HasNoControlWhen_NotItsTurn
Description: This test checks to see that when the player is not the CurrentPlayerNumber, than the player is not playing and is unable to make character selections.
Process:
This is an unimplemented feature, and is an empty test

Combat System Test 4
Test Name: Game_EndsWhenOnlyOne_Player()

Related Requirement: PGR4 - The player wins if all objectives are completed.
PGR5 - The player loses if all character die or an objective is failed.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->GridTests-> Game_EndsWhenOnlyOne_Player
Description: This test checks to see that when there is only one character or one enemy alive in the combat phase, isGameOver becomes true and ends the current combat phase section.
Process:
Start test combat stage
Make all Player 2 units take 1000 damage
Check if Player 1 has won

Combat System Test 5
Test Name: PlayerLoses_GoTo_Menu()
Related Requirement: PGR5 - Player Loses when all characters have died
PGR7 - If the player loses, then the player is shown a closing dialogue sequence and is returned to the main menu. The file is deemed unplayable.

Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->GridTests-> PlayerLoses_GoTo_Menu
Description: This test checks to see that when the player loses the current combat phase, the player is redirected to the menu view by checking if GetActiveScene is the menu scene.
Process:
Start test combat stage
Make all Player 1 units take 1000 damage
Check if Player 2 has won
Check if current scene is main menu

Combat System Test 6
Test Name: PlayerWins_GoTo_Map()
Related Requirement: PGR6 - If the player wins, then the system updates the file with the success while having a GUI output to player the outcome of the fight. The player is then returned to the Overhead map view.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->GridTests-> PlayerWins_GoTo_Map
Description: This test checks to see that when the player wins the current combat phase, the player is redirected to the overworld view by checking if GetActiveScene is the map scene.
Process:
Start test combat stage
Make all Player 2 units take 1000 damage
Check if Player 1 has won
Check if current scene is map

Combat System Test 7

Test Name: Player_CanAccessMenuDuring_Action()
Related Requirement: PGR8 - While a character is performing an action or the AI's turn is being conducted, the player shall be able to enter the menus, pausing the game.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->GridTests-> Player_CanAccessMenuDuring_Action
Description: This test checks to see that when that when the player has ended their turn, they are still able to open the menu. This test covers both the player's action being conducted and the AI's turn being conducted.
Process:
Start test combat stage
Start Next Turn
Try to open menu
Check if menu is open

Combat System Test 8
Test Name: Player_CannotSaveLoadDuring_Action()
Related Requirement: PGR9 - While a character is performing an action or the AI's turn is being conducted, the player shall be able to enter the menus, pausing the game.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->GridTests-> Player_CannotSaveLoadDuring_Action
Description: This test checks to see that when that when the player has ended their turn, they are not able to save or load from the menu
Process:
Start test combat stage
Start Next Turn
Try to open menu
Check that both save and load buttons are uninteractable

Combat System Test 9
Test Name: Player_CanSelect_Character()
Related Requirement: PGR11 - When it is the player's turn, the player shall be able to move the chosen character.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->GridTests-> Player_CanSelect_Character
Description: This test checks to see that when player's turn begins, the corresponding character in the character que is selected as the character for the turn.
Process:
Start test combat stage
Select a player 1 character

Combat System Test 10
Test Name: Player_CanMove_Character()

Related Requirement: PGR11 - When it is the player's turn, the player shall be able to move the chosen character.
PGR16 - When it is the AI's movement phase, the AI will not be able to move the chosen character on an obstacle on the map.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->GridTests-> Player_CanMove_Character
Description: This test checks to see that when the player has the ability to move the desired character, the available tiles that they are able to move the character to is not equal to null. This means that the there are available tiles the character can move to.
Process:
Start test combat stage
Select a player 1 character
Attempt to move the character
Check if Character has been moved to the correct location

Combat System Test 11
Test Name: Character_CannotMoveTo_Obstacle()
Related Requirement: PGR12 - When it is the player's movement phase, the player will not be able to move the chosen character on an obstacle on the map.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->GridTests-> Character_CannotMoveTo_Obstacle
Description: This test checks to see that when the player has the ability to move the desired character, the system checks to see if the listed set of available destinations does not contain any cells that contain an obstacle.
Process:
Start test combat stage
Select a player 1 character
Get all available move location
Check if the move locations are unoccupied

Combat System Test 12
Test Name: Character_CanPerformPossible_Actions()
Related Requirement: PGR13 - When the player has moved the chosen character, they maybe be able to select one of the four actions.
When the player is in the action phase of a chosen character, they will not be able to choose an actions that does not meet the subsequent criteria.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->GridTests-> Character_CanPerformPossible_Actions
Description: This test checks to see that when the player is performing their current character's action phase, that the character can attack an enemy within the listed number of cells.
Process:
Start test combat stage
Select a player 1 character

Check if player can attack
(There are other features to be checked, but are unimplemented)

Combat System Test 13
Test Name: Player_CannotSelectUndoable_Action()
Related Requirement: PGR14 - When the player is in the action phase of a chosen character, they will not be able to choose an actions that does not meet the subsequent criteria.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->GridTests-> Player_CannotSelectUndoable_Action
Description: This test checks to see that when the player is performing their current character's action phase, that the player cannot select the attack button if the character is not in range.
Process:
Start test combat stage
Select a player 1 character
Check if player can press the attack button (Shouldn't be able to due to not in range)
(There are other features to be checked, but are unimplemented)

Combat System Test 14
Test Name: AI_IsNot_Dumb()
Related Requirement: PGR15 - When it is the AI's turn, the AI shall be able to select a multitude of actions to perform to best situate itself given the subsequent criteria.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->GridTests-> AI_IsNot_Dumb
Description: This test checks to see if the AI actually thinks about its decisions (It currently doesn't)
Process:
Start test combat stage
Select a player 1 character
Check if player can press the attack button (Shouldn't be able to due to not in range)
(There are other features to be checked, but are unimplemented)

Combat System Test 15
Test Name: Player_CanCheck_Stats()
Related Requirement: PGR17 - The player shall be able to view character stats and enemy stats at anytime.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->GridTests-> Player_CanCheck_Stats
Description: This test checks to see that the player can check the stats of the characters on the grid.
Process:
Start test combat stage
Select a player 1 character
Check if the info panel is there and that it is correct

Combat System Test 16

Test Name: Unit_RemovedOn_Death()
Related Requirement: PGR18 - If a character dies, then they are removed from play.
PGR19 - If an enemy dies, then they are removed from play.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->GridTests-> Unit_RemovedOn_Death
Description: This test checks to see that when a given character or enemy dies, it is removed from play.
Process:
Start test combat stage
Deal 1000 damage to a unit
Check if unit is in the master unit list

Combat System Test 17
Test Name: Unit_DiesPreempt_Order()
Related Requirement: PGR20 - If a character dies and there remains only one final character, the immediate turn after the current phase is the remaining character.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->GridTests-> Unit_DiesPreempt_Order
Description: This test checks to see that when a given character or enemy dies, that deceased unit is now removed from the turn pool. This is checked by when applying a fatal amount of damage to the unit first-in-line to the turn selection, the second-in-line takes that player's place.
Process:
Start test combat stage
Deal 1000 damage to an enemy unit
Check if turn order is now player 2

Combat System Test 18
Test Name: TurnOrderDisplayed()
Related Requirement: CMR10. During the combat phase, the individual character and enemy turns are listed at the bottom of the screen.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->GridTests-> TurnOrderDisplayed
Description: This test checks to see that the order of the characters and enemy turns are individually listed at the bottom of the screen.
Process:
Start test combat stage
Inspect order of characters and enemies

4.3 Menu System Tests

Menu Test 1
Test Name: MainMenuOpened()
Related Requirement: MR1- The Main Menu is opened when the game is first initialized or when the player exits the current game session.
FC1 - If the player selects the exe file, the game is initialized from it's previous build.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->MapTest-> MainMenuOpened
Description: This test checks to see that when the game first runs, the current game session is "Menu" which is the main menu.
Process:
Start Menu
Check current scene is menu
Go to Map
Simulate going back to menu
Check current scene is menu

Menu Test 2
Test Name: MainMenuClosed()
Related Requirement: MR2 - The Main Menu is closed when the player exits the game or enters a game session.
FC3 - If the player its the close window button, the game terminates.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->MapTest-> MainMenuClosed
Description: This test checks to see that the game session is "null" when the game closes.
Process:
Start the menu scene
Simulate game exit
Check if scene is null

Menu Test 3
Test Name: GameMenuOpenClose()
Related Requirement: MR3 - The Game Menu is opened when the esc key is pressed or the menu button is pressed.
MR11- The Game Menu is closed when the esc key is pressed or the close menu button is pressed.
All sub menus can be closed if the player presses the esc key or the corresponding close menu key.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->MapTest-> GameMenuOpenClose
Description: This test checks to see that when the user hits the esc key, a menu will spawn if there is no menu open and the old menu will close if there was.
Process:

Start TestBattle
Simulate hitting the escape key
Check that menu is open
Simulate hitting the escape key
Check that menu is closed

Menu Test 4
Test Name: AudioMenuOpenClose()
Related Requirement: MR4 - For both menus, if the audio button is selected then Audio options is opened.
MR6- For both Audio and Graphics menus, if the back menu is selected then the changes are saved and the respected menu is closed.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->MapTest-> AudioMenuOpenClose
Description: This test checks to see that when the user selects the Audio Menu button after opening the in-game menu, an audio menu exists. It also checks that the menu can be closed.
Process:
Start TestBattle
Simulate hitting the escape key
Simulate navigation to audio menu
Check that audio menu is open
Simulate hitting the escape key
Check that menu is closed

Menu Test 5
Test Name: GraphicMenuOpenClose()
Related Requirement: MR5 - For both menus, if the graphics button is selected then Graphics options is opened.
MR6- For both Audio and Graphics menus, if the back menu is selected then the changes are saved and the respected menu is closed.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->MenuTest-> GraphicMenuOpenClose
Description: This test checks to see that when the user selects the Graphics Menu button after opening the in-game menu, an audio menu exists. It also checks that the menu can be closed.
Start TestBattle
Simulate hitting the escape key
Simulate navigation to graphics menu
Check that graphics menu is open
Simulate hitting the escape key
Check that menu is closed

Menu Test 6

Test Name: NewGameMenuInit()
Related Requirement: MR8- A new game menu is created if the player presses the New Game button from the Main Menu.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->MenuTest-> NewGameMenuInit
Description: This test checks to see that when the new game menu button is selected, that a new game is initialized.
Process:
Open Menu
Simulate hitting New Game button
Check if new game has been created

Menu Test 8
Test Name: LoadGameMenuOpenClose()
Related Requirement: MR9- A Load game menu is created if the player presses the Load Game button from the Main Menu.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->MenuTest-> LoadGameMenuOpenClose
Description: This test checks to see that when the load game menu button is selected, that a load game menu is opened.
Process:
Start Menu
Simulate the load game button
Check that load game menu exists

Menu Test 9
Test Name: SaveGameMenuOpenClose()
Related Requirement: MR10- A Save game menu is created if the player presses the Save Game button from the Main Menu.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->MenuTest-> SaveGameMenuOpenClose
Description: This test checks to see that when the save game menu button is selected, that a save game menu is opened.
Process:
Start TestBattle
Simulate hitting escape key
Simulate navigating to save menu
Check that save menu exists

Menu Test 10
Test Name: ItemMenuOpenClose()
Related Requirement: MR12 - The player should be able to access the the Item's menu from the Overworld menu by clicking the respective button or the respected hotkey.

Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->MenuTest-> ItemMenuOpenClose
Description: This test checks to see that when the Items menu button is selected, the Items menu is opened and the player can now start to view the Items.
Process:
Start Map
Simulate hitting the inventory button
Check inventory key open
Simulate hitting the inventory key
Check inventory is closed

Menu Test 11
Test Name: ToolTipDisplay()
Related Requirement: CMR17. Each statistical value has a tool tip area that will display a tool tip menu when the player has hovered over the value for a given period of time.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->MenuTest-> ToolTipDisplay
Description: This test checks to see that if the player's mouse hovers inside of the designated tooltip area for each stat for the respected period, the tool tip will generate.
Process:
For Each stat:
Place mouse inside of the designated area
Check for tool tip to pop up within given time period. 5 seconds is too long.

Menu Test 12
Test Name: ToolTipClippingCheck()
Related Requirement: CMR20. The tool tip will always fit on screen, never trailing off screen.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->MenuTest-> ToolTipClippingCheck
Description: This test checks to see that if the player's mouse hovers inside of the designated tooltip area for each stat for the respected period, the tool tip will generate.
Process:
For Each stat:
Place mouse inside of the designated area
Check to see if the tool tip generates off screen in any manner, being completely offscreen or clipped.

Menu Test 13
Test Name: ToolTipRemoval()
Related Requirement: CMR21. The tool tip disappears if the mouse is hovered away.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->MenuTest-> ToolTipRemoval
Description: This test checks to see that if the player's mouse hovers inside of the designated tooltip area for each stat for the respected period, the tool tip will generate.

Process:
For Each stat:
Place mouse inside of the designated area to generate the tool tip.
Check to see if each tool tip is erased when the mouse is moved.

4.4 Item System Tests

Item System Test 1
Test Name: ListedItems_Grid()
Related Requirement: AGR8 - Items will be listed in a grid system with respected image and border colour to indicate item type.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->ItemTest-> ListedItems_Grid()
Description: This test checks if the item menu organized the items in a grid fashion that is inside of the main panel.
Process:
Start ItemMenu
Check if the listed items have been organized in a grid format inside of panel.
Add/ delete an item by pressing the keys a or s
Check to see if the grid has been refreshed in the same grid format.

5. Map Unit Tests

The following section will the automatic unit tests used to ensure the requirements are being met. Each test will have an associated function in unity. These tests will be grouped by category in regards to functional requirements. Tests for functionality not yet implemented will assert false. Any requirements that included equipping items or equipment, interacting with the merchant, final bosses, overall game win/lose conditions, node logic functionality and displaying tooltips have not been included and can be treated as an assertFalse.. The development team has not implemented these requirements yet and must still determine how these requirements will be implemented.

5.1 Map Generation Tests

Map Generation Test 1
Test: AtLestTenNodes()
Related Requirement: CMR2 - Each generated map has at least 10 nodes

Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->MapTest->AtLestTenNodes
Description: This test checks the total number of nodes on the overworld map and asserts that there are at least 10 nodes on the map.

Map Generation Test 2
Test: EachNodeThreeNeighbours()
Related Requirement: CMR2 - Each generated node has at least three neighbours
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->MapTest->EachNodeThreeNeighbours
Description: This test checks each node generated on the overworld and asserts that each node has at least three neighbours in its neighbour list.

Map Generation Test 3
Test: EachNodeLessThanEightNeighbours()
Related Requirement: CMR2 - Each generated node has no more than 8 neighbours
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->MapTest->EachNodeLessThanEightNeighbours
Description: This test checks each node generated on the overworld and asserts that each node has no more than eight neighbours in its neighbour list.

Map Generation Test 4
Test: EachNodeReachable()
Related Requirement: CMR1 - Every node on a map is reachable by every other node in a region.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->MapTest-> EachNodeReachable

Description: This test starts at the first node, and recursively attempts to visit all of neighbours of each node. It keeps track of the nodes visited to prevent multiple visits to the same node. After the all linked nodes to the initial node has been visited, the test asserts that all nodes in the game exist in the set of nodes visited.

Map Generation Test 5
Test: EachNodeHasEvent()
Related Requirement: CMR6 - Each node in the map has an associated event [expected to fail]
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->MapTest-> EachNodeHasEvent
Description: This test checks every node generated for an associated event type attribute that is not an empty string. An empty string implies an event has not yet been assigned to this node. This test only passes if the assert for every node is successful.

Map Generation Test 6
Test: EachNodeInARegion()
Related Requirement:Each node on the overworld map belongs to a region()[expected to fail]
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->MapTest-> EachNodeInARegion
Description: This test checks every node generated for a region enumeration that allots it in a region from a set of possible regions. This test applies the assertion to every node on the map, and thus will fail if any node is enumerated as Region.NULL.

Map Generation Test 7
Test: NodeTravelHasCost()
Related Requirement: CMR5, CMR7- Every node has an associated travel cost and does not allow the player to become stranded. Each node should not deadlock the character with a negative outcome that makes them lose the game.[expected to fail]
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->MapTest-> NodeTravelHasCost
Description: This test checks the cost of travelling between all available nodes to the first node. This test asserts the costs for travelling between nodes are non-zero, positive values.

Map Generation Test 8
Test: NodeNoOverLap()
Related Requirement: CMR3 - Nodes will not overlap with each other
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->MapTest-> NodeNoOverLap
Description: This test checks that no nodes are within 0.1 distance of each other on the overworld map. This is roughly the diameter of each node, thus the minimum separation between nodes to ensure no overlap.

Map Generation Test 9

Test Name: Grid_IsCreatedWithCorrect_NumberOfCells()
Related Requirement: CMR4 - The Combat map will be created with the correct N by N size.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->GridTest-> Grid_IsCreatedWithCorrect_NumberOfCells
Description: This test checks that the when the combat map is first generated, that the map has the correct number of cells based on it's given N by M values.

CMR8. Each node has a cost of supplies in that the player needs to get to the specic node.

5.2 Combat Unit Tests

Combat Unit Test 1
Test Name: Unit_TakesCorrectAmountOf_Damage()
Related Requirement: PGR36 - The player and the AI can decide either to attack the health values or the armour values of their respective enemies.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->GridTest-> Unit_TakesCorrectAmountOf_Damage
Description: This test checks if the a character or enemy takes damage, that the resulting damage is applied to the character's health.

Combat Unit Test 2
Test Name: CharacterTypeAndBaseStatsInit()
Related Requirement: CMR9. Each character is given a class type and base stats values.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->GridTest-> CharacterTypeAndBaseStatsInit
Description: This test checks if each character is given a class type with corresponding stat values.
These are a list of combat requirements that need to be included

Combat Unit Test 3
Test Name: RandomOrderKept()
Related Requirement:CMR11. The character and enemy turns orders are randomly mixed up and are kept in the same order for the duration of the combat phase.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->GridTest-> RandomOrderKept
Description: This test checks two things. 1. Ensure that the orders for each turn is kept. 2. The second order generated is not the same as the previous one.

Combat Unit Test 4
Test Name: CharacterRemovedFromPlay()

Related Requirement: CMR12. If an enemy or character dies, they're turns are removed from the combat phase.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->GridTest-> CharacterRemovedFromPlay
Description: This test checks to see that if a character dies in combat, that they are removed from the play for their next turn, ie they do not get to have their turn.

Combat Unit Test 5
Test Name: HealthArmourInit()
Related Requirement: CMR13. Characters and Enemies have default health and Armour.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->GridTest-> HealthArmourInit
Description: This test checks to see that characters and enemies start with their default health and armour values at the beginning of a combat phase.

Combat Unit Test 6
Test Name: HealthArmourAttackSelection()
Related Requirement: CMR14. The player and the AI can decide either to attack the health values or the armour values of their respected enemies.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->GridTest-> HealthArmourAttackSelection
Description: This test checks to see that characters and enemies can attack the respective side's health or armour values.

Combat Unit Test 7
Test Name: ArmourAttackSelection_Zero()
Related Requirement: CMR15. When the Armour value reaches 0, that selection cannot be made.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->GridTest-> ArmourAttackSelection_Zero
Description: This test checks to see that characters and enemies cannot attack Armour values when a character or enemy has a armour value of 0.

Combat Unit Test 8
Test Name: ArmourInverseProportion()
Related Requirement: CMR16. The Armour value must provide a reduced damage value that is of some sort of inverse exponential or logarithmic value to promote the intensive to remove amour.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->GridTest-> ArmourInverseProportion
Description: This test checks to see that characters with different armour values will have an inverse exponential/logarithmic health reduction relation. This will be done by comparing them with mathematically scaled model results and to see if the results fall within Unity's isApproximatleyEqual test result.

5.3 Menu Unit Tests

Menu Unit Test 1
Test Name: ToolTipBreakdown()
Related Requirement: CMR18, CMR19. The tool tip displays how the value is derived. The tool tip will show different ways on how the value is changed.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->MenuTest-> ToolTipBreakdown
Description: This test checks to see that the tool tip display values match up with the value that had been saved to the sterilized object.

5.4 Item Unit Tests

Item Unit Test 1
Test Name: ItemDisplayed()
Related Requirement: AGR9 - When items are selected, their information will be displayed to the screen.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->ItemTest-> ItemDisplayed
Description: This test checks to see if the desired item has duplicated its image to the infodisplay panel along with the correct info.

Item Unit Test 2
Test Name: ConsumableItem_Consumed()
Related Requirement: AGR10. If items are of the consumable type, the use of this item will remove the item from play and the player's item list.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->ItemTest-> ConsumableItem_Consumed
Description: This test checks to see if a consumable Item will be destroyed from the inventory if used.

Item Unit Test 3
Test Name: ExpendableItem_Refilled()
Related Requirement: AGR11. If items are of the expendable type, the use of this item will decremented until they reach 0, where they will be refilled when the combat phase ends.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->ItemTest-> ExpenableItem_Refilled
Description: This test checks to see if an expendable Item will be refilled for the next combat phase if used.

Item Unit Test 4
Test Name: RechargableItem_Recharged()

Related Requirement: AGR12. If items are of the rechargeable type, the use of the item will not allow the player to use the item again until the number of cooldown turns been completed.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->ItemTest-> RechargableItem_Recharged
Description: This test checks to see if an rechargeable Item will be recharged after a set number of turns have passed if used.

Item Unit Test 5
Test Name: PassiveItem_Active()
Related Requirement: AGR13. If items are of the passive type, the item applies it's ability every turn for the course of the combat phase. These items are not "used" like other items.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->ItemTest-> PassiveItem_Active
Description: This test checks to see if an equipped passive Item will be change the stats of character without overwriting the previous values or changing other values.

Item Unit Test 6
Test Name: InstantItem_Active()
Related Requirement: AGR14. If items are of the instant type, the item can be used for every turn of the assigned character.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->ItemTest-> InstantItem_Active
Description: This test checks to see if an equipped Instant Item will can be used for the next turn phase for the rest of the combat phase.

Item Unit Test 7
Test Name: Item_Hidden()
Related Requirement: AGR15. If an item is equipped to character, no other characters are able to equip that same instance of the item.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->ItemTest-> Item_Hidden
Description: This test checks to see if when an Item has been equipped, that no other character can equip the same item.

Item Unit Test 8
Test Name: StatChange_MouseHoverOverItem()
Related Requirement: AGR16. If the mouse is hovering over a desired item and the item changes the stats of a character, those stats will be updated on the selected charac-
ter with incremented values being coloured in blue, decremented stats
being coloured in red and unchanged values being coloured in white.
Access: Unity -> Window->Test Runner-> Play Mode->Turn Based Strategy Framework->Assembly-CSharp.dll->ItemTest-> StatChange_MouseHoverOverItem

Description: This test checks to see if the player hovers the mouse over an item in the itempool, the stats changes are reflected in the InfoPanel with the correct values. The test will also validate if the text colours for the stats have changed.