Settle And Destroy (SAD)

Group 13

Jonas Wikberg
Christofer Hjalmarsson
Daniel Westerberg
Saul Amram
André Sikborn Erixon

1.

2.

3.

4.

5.

6. Functional Test Cases

6.1. Different types of troops

Test to see if there are different types troops (FUNCTIONAL REQUIREMENTS 4.4.3.1)

Input	None
Expected output	Different type of troops are constructed

1. Upgrade village barracks, AbstractBuilding.upgrade()
2. Construct troops of the two different types that is now possible within Barracks.BuildableItem[]
3. Compare the two troops Troops.getStrength()

6.2. Train military troops

Test if it is possible to build troops (FUNCTIONAL REQUIREMENTS 4.4.3.2)

Input	None
Expected output	Troops are constructed

1. Check Village.getHomeArmy().getTotalSize()
2. Build any kind of troop.
3. Compare new Village.getHomeArmy().getTotalSize() with the old value.

6.3. Armies consists of troops

Test if an army consist of troops (FUNCTIONAL REQUIREMENTS 4.4.3.3.)

Input	None
Expected output	Troops are constructed

1. Check if Army.getTotalTroops()

6.4. Resources

Test that resources are collected (FUNCTIONAL REQUIREMENTS 4.4.1.3, 4.4.1.4).

Input	A village.
Expected output	Resources are increased at a constant rate.

1. Create a new main window for a village.
2. Measure resources.
3. Wait some seconds.
4. Measure resources again.

6.5. Buildings

Test that different buildings can be constructed and upgraded. (FUNCTIONAL REQUIREMENTS 4.4.2.2, 4.4.2.3, 4.4.2.4)

Input	A village.
Expected output	A building is created and upgraded.

1. Create a new Barracks, Stable or other building class. A village is needed in the constructor.
2. Upgrade the building by calling upgrade().
3. Check that the building has been upgraded.

6.6. Map

Test the map and that it has different cells. (FUNCTIONAL REQUIREMENTS 4.4.4.1, 4.4.4.2)

Input	None.
Expected output	A map with different cells.

1. Create a new Map.
2. Check width and height.
3. Check different cells using getCell(int x, int y).

6.7. Connect to a multiplayer game

Test to connect to a multiplayer game. (FUNCTIONAL REQUIREMENTS 4.4.5.2)

Input	A multiplayer game has been started on a local or remote machine.
Expected output	A connection was made to a multiplayer game.

1. Create a new Client.
2. Call connect(host, port) to connect to the server.
3. Check the isConnected value.

6.8. Multiplayer game settings

Test to connect to a multiplayer game. (FUNCTIONAL REQUIREMENTS 4.4.5.3)

Input	None.
Expected output	Multiplayer settings have been changed.

1. Create a new HostGameDialog.
2. Change the number of players by calling setNumberOfPlayers(int).
3. Call getNumberOfPlayers.

6.9. Player specific colours

Tests that all players have unique colors. (FUNCTIONAL REQUIREMENTS 6.1.5.6)

Input	A number of players
Expected output	That all teams have different colors

1. Create a number of teams, the same amount as the amount of players.
2. Iterate through all teams.
3. Call getColor() for each team and put that color in a list.
4. For each team except first, check that its color does not already exist in the list.
5. Check that the list size is the same as the number of teams.

6.10. Assign each player a unique player name

Tests that all players have unique names. (FUNCTIONAL REQUIREMENTS 6.1.5.1)

Input	A number of players
Expected output	The same amount of teams, all with unique names

1. Create a new game.
2. Create the same number of teams as players.
3. Use getName() on each team and put that name in a name list.
4. Check that the wanted player name does not already exist in the name list.

6.11. A chosen valid Race

Tests that a player has a valid race. (FUNCTIONAL REQUIREMENTS 6.1.1.5)

Input	A player
Expected output	

1. Create a team for the player.
2. Call getRace() and check that it is a existing race (and not null).

6.12. A player shall have one village

Tests that a players has one village. (FUNCTIONAL REQUIREMENTS 6.1.1.2)

Input	A player
Expected output	

1. Create a team.
2. Call getVillage() and check that it is a village object (not null)

6.13. Move armies around the map

Tests that an army has been moved (FUNCTIONAL REQUIREMENTS 4.4.3.7)

Input	An army
Expected output	The army has moved to another position.

1. Check position of the army.
2. Move the army one step
3. Check position of the army again.

6.14. Armies never separate

Tests that the army never loose size if you just move them (no conflicts) (FUNCTIONAL REQUIREMENTS 4.4.3.4)

Input	An army
Expected output	The army has moved to another position and still consist of the same size of troops.

1. Check position of the army
2. Check the troops size of troopA, troopB, troopC
3. Move the army one step.
4. Check the troop size of troopA, troopB, troopC.

6.15. Attack Village with armies

Test that a village has been attacked (FUNCTIONAL REQUIREMENTS 4.4.3.5)

Input	An army big enough to do damage to a village
Expected output	A village that has been under attack

1. Check health on the village
2. Move the army to the village
3. Check the health on the village again.

6.16. Conflict of armies

Test that there has been a conflict between armies (FUNCTIONAL REQUIREMENTS 4..4.3.6)

Input	Army A, army B
Expected output	One army remaining with troops

1. Check getTotalSize() for both army A and army b
2. Move army A to army B's position.
3. Check getTtotalSize() for both army A and army B.