COS-214 PROJECT JJJJM

Generated by Doxygen 1.9.6

1 GitHub Repository 1

2 Hierarchical Index 3

2.1 Class Hierarchy . . . 3

3 Class Index 5

3.1 Class List . . . 5

4 File Index 7

4.1 File List . . . 7

5 Class Documentation 9

5.1 Backup Class Reference . . . 9

5.1.1 Constructor & Destructor Documentation . . . 9

5.1.1.1 Backup() . . . 9

5.1.1.2 ~Backup() . . . 9

5.1.2 Member Function Documentation . . . 10

5.1.2.1 addMemento() . . . 10

5.1.2.2 clear() . . . 10

5.1.2.3 getMemento() . . . 10

5.1.2.4 getMementoCount() . . . 10

5.2 Battalion Class Reference . . . 11

5.2.1 Constructor & Destructor Documentation . . . 11

5.2.1.1 Battalion() . . . 11

5.2.2 Member Function Documentation . . . 11

5.2.2.1 attack() . . . 11

5.2.2.2 getBattalionDestroyed() . . . 12

5.2.2.3 setNumBattalionDestroys() . . . 12

5.3 BottomNeighbour Class Reference . . . 12

5.3.1 Detailed Description . . . 13

5.3.2 Constructor & Destructor Documentation . . . 13

5.3.2.1 BottomNeighbour() . . . 13

5.3.3 Member Function Documentation . . . 13

5.3.3.1 getBottom() . . . 13

5.3.3.2 hasBottom() . . . 14

5.4 Country Class Reference . . . 14

5.4.1 Constructor & Destructor Documentation . . . 16

5.4.1.1 Country() . . . 16

5.4.2 Member Function Documentation . . . 16

5.4.2.1 attachObserver() . . . 17

5.4.2.2 checkIsDead() . . . 17

5.4.2.3 clone() . . . 17

5.4.2.4 compareAspect() [1/2] . . . 17

5.4.2.5 compareAspect() [2/2] . . . 18

5.4.2.6 compareDomestic() . . . 18
5.4.2.7 compareMilitary() . . . 19
5.4.2.8 detachObserver() . . . 19
5.4.2.9 getBorderStrength() . . . 19
5.4.2.10 getCapital() . . . 19
5.4.2.11 getCapitalSafety() . . . 20
5.4.2.12 getColor() . . . 20
5.4.2.13 getCountryRating() . . . 20
5.4.2.14 getCountryState() . . . 20
5.4.2.15 getDomesticMorale() . . . 21
5.4.2.16 getEnemies() . . . 21
5.4.2.17 getLocations() . . . 21
5.4.2.18 getMilitary() . . . 21
5.4.2.19 getName() . . . 22
5.4.2.20 getNumCitizens() . . . 22
5.4.2.21 getPoliticalStability() . . . 22
5.4.2.22 getSelfReliance() . . . 22
5.4.2.23 getState() . . . 23
5.4.2.24 getTradeRouteSafety() . . . 23
5.4.2.25 getWarSentiment() . . . 23
5.4.2.26 removeEnemy() . . . 23
5.4.2.27 resetEnemies() . . . 24
5.4.2.28 resetLocations() . . . 24
5.4.2.29 setBorderStrength() . . . 24
5.4.2.30 setCapital() . . . 24
5.4.2.31 setCapitalSafety() . . . 25
5.4.2.32 setColor() . . . 25
5.4.2.33 setColorOfDestroyedBy() . . . 25
5.4.2.34 setCountryState() . . . 26
5.4.2.35 setDomesticMorale() . . . 26
5.4.2.36 setEnemies() . . . 26
5.4.2.37 setName() . . . 26
5.4.2.38 setNumCitizens() . . . 27
5.4.2.39 setPoliticalStability() . . . 27
5.4.2.40 setSelfReliance() . . . 27
5.4.2.41 setState() . . . 28
5.4.2.42 setTradeRouteSafety() . . . 28
5.4.2.43 setWarSentiment() . . . 28
5.4.2.44 takeTurn() [1/2] . . . 28
5.4.2.45 takeTurn() [2/2] . . . 29
5.5 CountryState Class Reference . . . 29
5.5.1 Constructor & Destructor Documentation . . . 30

5.5.1.1 CountryState() [1/3] . . . 30
5.5.1.2 CountryState() [2/3] . . . 30
5.5.1.3 CountryState() [3/3] . . . 30
5.5.1.4 ~CountryState() . . . 30
5.5.2 Member Function Documentation . . . 31
5.5.2.1 clone() . . . 31
5.5.2.2 getMilitaryState() . . . 31
5.5.2.3 setMilitaryState() . . . 31
5.6 EarlyStage Class Reference . . . 31
5.6.1 Constructor & Destructor Documentation . . . 32
5.6.1.1 EarlyStage() . . . 32
5.6.1.2 ~EarlyStage() . . . 32
5.6.2 Member Function Documentation . . . 32
5.6.2.1 clone() . . . 32
5.6.2.2 getWarStage() . . . 33
5.7 EarlyStrategy Class Reference . . . 33
5.7.1 Detailed Description . . . 33
5.7.2 Member Function Documentation . . . 34
5.7.2.1 defensiveMove() . . . 34
5.7.2.2 neutralMove() . . . 35
5.7.2.3 offensiveMove() . . . 35
5.8 Iterator Class Reference . . . 36
5.8.1 Member Function Documentation . . . 36
5.8.1.1 first() . . . 36
5.8.1.2 getCurrent() . . . 36
5.8.1.3 isDone() . . . 37
5.8.1.4 next() . . . 37
5.9 LateStage Class Reference . . . 37
5.9.1 Constructor & Destructor Documentation . . . 38
5.9.1.1 LateStage() . . . 38
5.9.1.2 ~LateStage() . . . 38
5.9.2 Member Function Documentation . . . 38
5.9.2.1 clone() . . . 38
5.9.2.2 getWarStage() . . . 38
5.10 LateStrategy Class Reference . . . 39
5.10.1 Detailed Description . . . 39
5.10.2 Constructor & Destructor Documentation . . . 39
5.10.2.1 LateStrategy() . . . 39
5.10.3 Member Function Documentation . . . 39
5.10.3.1 defensiveMove() . . . 39
5.10.3.2 neutralMove() . . . 40
5.10.3.3 offensiveMove() . . . 40

5.11 LeftNeighbour Class Reference . . . 41
5.11.1 Detailed Description . . . 41
5.11.2 Constructor & Destructor Documentation . . . 41
5.11.2.1 LeftNeighbour() . . . 41
5.11.3 Member Function Documentation . . . 42
5.11.3.1 getLeft() . . . 42
5.11.3.2 hasLeft() . . . 42
5.12 Location Class Reference . . . 42
5.12.1 Constructor & Destructor Documentation . . . 44
5.12.1.1 ~Location() . . . 44
5.12.2 Member Function Documentation . . . 44
5.12.2.1 add() . . . 44
5.12.2.2 clone() . . . 44
5.12.2.3 createIterator() . . . 45
5.12.2.4 getBottom() . . . 45
5.12.2.5 getColor() . . . 45
5.12.2.6 getIsCapital() . . . 45
5.12.2.7 getIsLand() . . . 46
5.12.2.8 getLeft() . . . 46
5.12.2.9 getOwnedBy() . . . 46
5.12.2.10 getRight() . . . 46
5.12.2.11 getTop() . . . 47
5.12.2.12 getX() . . . 47
5.12.2.13 getY() . . . 47
5.12.2.14 hasBottom() . . . 47
5.12.2.15 hasLeft() . . . 48
5.12.2.16 hasRight() . . . 48
5.12.2.17 hasTop() . . . 48
5.12.2.18 setColor() . . . 48
5.12.2.19 setIsCapital() . . . 49
5.12.2.20 setIsLand() . . . 49
5.12.2.21 setOwnedBy() . . . 49
5.13 LocationIterator Class Reference . . . 50
5.13.1 Constructor & Destructor Documentation . . . 50
5.13.1.1 LocationIterator() . . . 50
5.13.1.2 ~LocationIterator() . . . 51
5.13.2 Member Function Documentation . . . 51
5.13.2.1 first() . . . 51
5.13.2.2 getCurrent() . . . 51
5.13.2.3 isDone() . . . 51
5.13.2.4 next() . . . 52
5.14 LocationObserver Class Reference . . . 52

5.15 Map Class Reference . . . 52
5.15.1 Constructor & Destructor Documentation . . . 53
5.15.1.1 Map() [1/3] . . . 53
5.15.1.2 Map() [2/3] . . . 53
5.15.1.3 ~Map() . . . 53
5.15.1.4 Map() [3/3] . . . 53
5.15.2 Member Function Documentation . . . 54
5.15.2.1 getLocation() . . . 54
5.15.2.2 getState() . . . 54
5.15.2.3 getTopLeft() . . . 54
5.15.2.4 printMap() . . . 55
5.16 MapState Class Reference . . . 55
5.16.1 Detailed Description . . . 55
5.16.2 Constructor & Destructor Documentation . . . 55
5.16.2.1 MapState() . . . 55
5.16.2.2 ~MapState() . . . 56
5.16.3 Member Function Documentation . . . 56
5.16.3.1 clone() . . . 56
5.17 Memento Class Reference . . . 56
5.17.1 Constructor & Destructor Documentation . . . 57
5.17.1.1 Memento() [1/2] . . . 57
5.17.1.2 Memento() [2/2] . . . 57
5.17.1.3 ~Memento() . . . 57
5.17.2 Member Function Documentation . . . 57
5.17.2.1 getState() . . . 57
5.17.2.2 setState() . . . 58
5.18 MiddleStage Class Reference . . . 59
5.18.1 Constructor & Destructor Documentation . . . 59
5.18.1.1 MiddleStage() . . . 59
5.18.1.2 ~MiddleStage() . . . 59
5.18.2 Member Function Documentation . . . 60
5.18.2.1 clone() . . . 60
5.18.2.2 getWarStage() . . . 60
5.19 MiddleStrategy Class Reference . . . 60
5.19.1 Detailed Description . . . 61
5.19.2 Member Function Documentation . . . 61
5.19.2.1 defensiveMove() . . . 61
5.19.2.2 neutralMove() . . . 61
5.19.2.3 offensiveMove() . . . 62
5.20 Military Class Reference . . . 62
5.20.1 Constructor & Destructor Documentation . . . 62
5.20.1.1 Military() [1/2] . . . 62

5.20.1.2 Military() [2/2] . . . 63
5.20.1.3 ~Military() . . . 63
5.21 MilitaryState Class Reference . . . 63
5.21.1 Member Function Documentation . . . 64
5.21.1.1 clone() . . . 64
5.21.1.2 getNumPlanes() . . . 64
5.21.1.3 getNumShips() . . . 64
5.21.1.4 getNumTanks() . . . 65
5.21.1.5 getNumTroops() . . . 65
5.21.1.6 setTroops() . . . 65
5.21.1.7 updateNumBattalions() . . . 65
5.21.1.8 updateNumShips() . . . 66
5.21.1.9 updateNumTanks() . . . 66
5.22 Neighbour Class Reference . . . 66
5.22.1 Detailed Description . . . 67
5.22.2 Constructor & Destructor Documentation . . . 67
5.22.2.1 Neighbour() . . . 67
5.22.2.2 ~Neighbour() . . . 67
5.22.3 Member Function Documentation . . . 67
5.22.3.1 add() . . . 68
5.23 Plane Class Reference . . . 68
5.23.1 Constructor & Destructor Documentation . . . 68
5.23.1.1 Plane() . . . 68
5.24 PlaneFactory Class Reference . . . 69
5.24.1 Detailed Description . . . 69
5.24.2 Member Function Documentation . . . 69
5.24.2.1 clone() . . . 69
5.24.2.2 manufactureVehicle() . . . 70
5.25 RightNeighbour Class Reference . . . 70
5.25.1 Detailed Description . . . 70
5.25.2 Constructor & Destructor Documentation . . . 71
5.25.2.1 RightNeighbour() . . . 71
5.25.3 Member Function Documentation . . . 72
5.25.3.1 getRight() . . . 72
5.25.3.2 hasRight() . . . 72
5.26 Ship Class Reference . . . 72
5.26.1 Detailed Description . . . 73
5.26.2 Constructor & Destructor Documentation . . . 73
5.26.2.1 Ship() . . . 73
5.27 ShipFactory Class Reference . . . 73
5.27.1 Detailed Description . . . 74
5.27.2 Member Function Documentation . . . 74

5.27.2.1 clone() . . . 74
5.27.2.2 manufactureVehicle() . . . 74
5.28 SimulationManager Class Reference . . . 75
5.28.1 Constructor & Destructor Documentation . . . 75
5.28.1.1 SimulationManager() . . . 75
5.28.1.2 ∼SimulationManager() . . . 76
5.28.2 Member Function Documentation . . . 76
5.28.2.1 designModeAction() . . . 76
5.28.2.2 finalMessage() . . . 76
5.28.2.3 isSimulationRunning() . . . 76
5.28.2.4 processMenu() . . . 76
5.28.2.5 resetSimulation() . . . 77
5.28.2.6 restoreState() . . . 77
5.28.2.7 runSimulation() . . . 77
5.28.2.8 saveState() . . . 77
5.28.2.9 takeTurn() . . . 77
5.28.2.10 viewCountrySummary() . . . 78
5.28.2.11 viewSummary() . . . 78
5.29 SimulationState Class Reference . . . 78
5.29.1 Constructor & Destructor Documentation . . . 78
5.29.1.1 SimulationState() . . . 79
5.29.1.2 ∼SimulationState() . . . 79
5.29.2 Member Function Documentation . . . 79
5.29.2.1 addSuperpowerState() . . . 79
5.29.2.2 getMapState() . . . 79
5.29.2.3 getStageContextState() . . . 80
5.29.2.4 getSuperpowerState() . . . 80
5.29.2.5 getSuperpowerStateCount() . . . 80
5.29.2.6 getTimestamp() . . . 81
5.29.2.7 setMapState() . . . 81
5.29.2.8 setStageContextState() . . . 81
5.30 StageContext Class Reference . . . 81
5.30.1 Constructor & Destructor Documentation . . . 82
5.30.1.1 ∼StageContext() . . . 83
5.30.2 Member Function Documentation . . . 83
5.30.2.1 getSimulationLength() . . . 83
5.30.2.2 moveToStage() . . . 83
5.30.2.3 setCurrentRound() . . . 83
5.30.2.4 setCurrentStage() . . . 84
5.30.2.5 setSimulationLength() . . . 84
5.30.2.6 setState() . . . 84
5.30.3 Member Data Documentation . . . 84

5.30.3.1 onlyInstance . . . 84
5.31 StageContextState Class Reference . . . 85
5.31.1 Constructor & Destructor Documentation . . . 85
5.31.1.1 StageContextState() . . . 85
5.31.1.2 ~StageContextState() . . . 85
5.31.2 Member Function Documentation . . . 86
5.31.2.1 getCurrentStage() . . . 86
5.31.2.2 getSimulationLength() . . . 86
5.31.2.3 setCurrentRound() . . . 86
5.31.2.4 setCurrentStage() . . . 86
5.31.2.5 setSimulationLength() . . . 87
5.32 Strategy Class Reference . . . 87
5.32.1 Detailed Description . . . 88
5.32.2 Constructor & Destructor Documentation . . . 88
5.32.2.1 Strategy() . . . 88
5.32.3 Member Function Documentation . . . 88
5.32.3.1 defensiveMove() . . . 88
5.32.3.2 neutralMove() . . . 88
5.32.3.3 offensiveMove() . . . 89
5.32.3.4 takeTurn() . . . 89
5.33 Superpower Class Reference . . . 89
5.33.1 Constructor & Destructor Documentation . . . 90
5.33.1.1 Superpower() [1/2] . . . 90
5.33.1.2 Superpower() [2/2] . . . 90
5.33.1.3 ~Superpower() . . . 91
5.33.2 Member Function Documentation . . . 91
5.33.2.1 addCountry() . . . 91
5.33.2.2 getAllCountries() . . . 91
5.33.2.3 getCountry() . . . 91
5.33.2.4 getCountryCount() . . . 92
5.33.2.5 getName() . . . 92
5.33.2.6 getState() . . . 92
5.33.2.7 printSummary() . . . 92
5.33.2.8 removeCountry() . . . 92
5.33.2.9 resetEnemies() . . . 93
5.33.2.10 resetLocations() . . . 93
5.34 SuperpowerState Class Reference . . . 93
5.34.1 Constructor & Destructor Documentation . . . 94
5.34.1.1 SuperpowerState() . . . 94
5.34.1.2 ~SuperpowerState() . . . 94
5.34.2 Member Function Documentation . . . 94
5.34.2.1 addCountryState() . . . 94

5.34.2.2 getCountryState() . . . 95
5.34.2.3 getCountryStateCount() . . . 95
5.34.2.4 getName() . . . 95
5.35 Tank Class Reference . . . 96
5.35.1 Detailed Description . . . 96
5.35.2 Constructor & Destructor Documentation . . . 96
5.35.2.1 Tank() . . . 96
5.36 TankFactory Class Reference . . . 97
5.36.1 Detailed Description . . . 97
5.36.2 Member Function Documentation . . . 97
5.36.2.1 clone() . . . 97
5.36.2.2 manufactureVehicle() . . . 98
5.37 Territory Class Reference . . . 98
5.37.1 Constructor & Destructor Documentation . . . 98
5.37.1.1 Territory() . . . 98
5.37.1.2 ∼Territory() . . . 99
5.37.2 Member Function Documentation . . . 99
5.37.2.1 add() . . . 99
5.38 TopNeighbour Class Reference . . . 99
5.38.1 Detailed Description . . . 100
5.38.2 Constructor & Destructor Documentation . . . 100
5.38.2.1 TopNeighbour() . . . 100
5.38.3 Member Function Documentation . . . 100
5.38.3.1 getTop() . . . 100
5.38.3.2 hasTop() . . . 101
5.39 Vehicle Class Reference . . . 101
5.39.1 Detailed Description . . . 101
5.39.2 Constructor & Destructor Documentation . . . 101
5.39.2.1 Vehicle() . . . 102
5.40 VehicleFactory Class Reference . . . 102
5.40.1 Detailed Description . . . 102
5.40.2 Constructor & Destructor Documentation . . . 102
5.40.2.1 VehicleFactory() . . . 103
5.40.3 Member Function Documentation . . . 103
5.40.3.1 clone() . . . 103
5.40.3.2 manufactureVehicle() . . . 103
5.41 WarStage Class Reference . . . 104
5.41.1 Constructor & Destructor Documentation . . . 104
5.41.1.1 WarStage() . . . 104
5.41.1.2 ∼WarStage() . . . 104
5.41.2 Member Function Documentation . . . 104
5.41.2.1 clone() . . . 105

5.41.2.2 getWarStage() . . . 105

6 File Documentation **107**

6.1 Backup.h . . . 107
6.2 Battalion.h . . . 107
6.3 BottomNeighbour.h . . . 108
6.4 Country.h . . . 108
6.5 CountryState.h . . . 110
6.6 EarlyStage.h . . . 110
6.7 EarlyStrategy.h . . . 111
6.8 Iterator.h . . . 111
6.9 LateStage.h . . . 111
6.10 LateStrategy.h . . . 112
6.11 LeftNeighbour.h . . . 112
6.12 Location.h . . . 112
6.13 LocationIterator.h . . . 113
6.14 LocationObserver.h . . . 114
6.15 Map.h . . . 114
6.16 MapState.h . . . 115
6.17 Memento.h . . . 115
6.18 MiddleStage.h . . . 115
6.19 MiddleStrategy.h . . . 116
6.20 Military.h . . . 116
6.21 MilitaryState.h . . . 116
6.22 Neighbour.h . . . 117
6.23 Plane.h . . . 118
6.24 PlaneFactory.h . . . 118
6.25 RightNeighbour.h . . . 118
6.26 Ship.h . . . 119
6.27 ShipFactory.h . . . 119
6.28 SimulationManager.h . . . 119
6.29 SimulationState.h . . . 120
6.30 StageContext.h . . . 121
6.31 StageContextState.h . . . 121
6.32 Strategy.h . . . 122
6.33 Superpower.h . . . 122
6.34 SuperpowerState.h . . . 123
6.35 Tank.h . . . 123
6.36 TankFactory.h . . . 123
6.37 Territory.h . . . 124
6.38 TopNeighbour.h . . . 124
6.39 Vehicle.h . . . 124

6.40 VehicleFactory.h 125
6.41 WarStage.h 125

Chapter 1

GitHub Repository

Link to GitHub repository

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

- Backup . . . 9
- Battalion . . . 11
- Country . . . 14
- CountryState . . . 29
- Iterator . . . 36
  - LocationIterator . . . 50
- Location . . . 42
  - Neighbour . . . 66
    - BottomNeighbour . . . 12
    - LeftNeighbour . . . 41
    - RightNeighbour . . . 70
    - TopNeighbour . . . 99
  - Territory . . . 98
- LocationObserver . . . 52
- Map . . . 52
- MapState . . . 55
- Memento . . . 56
- Military . . . 62
- MilitaryState . . . 63
- SimulationManager . . . 75
- SimulationState . . . 78
- StageContext . . . 81
- StageContextState . . . 85
- Strategy . . . 87
  - EarlyStrategy . . . 33
  - LateStrategy . . . 39
  - MiddleStrategy . . . 60
- Superpower . . . 89
- SuperpowerState . . . 93
- Vehicle . . . 101
  - Plane . . . 68
  - Ship . . . 72
  - Tank . . . 96
- VehicleFactory . . . 102

PlaneFactory 69
ShipFactory 73
TankFactory 97
WarStage 104
EarlyStage 31
LateStage 37
MiddleStage 59

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Backup 9
Battalion 11
BottomNeighbour 12
Country 14
CountryState 29
EarlyStage 31
EarlyStrategy 33
Iterator 36
LateStage 37
LateStrategy 39
LeftNeighbour 41
Location 42
LocationIterator 50
LocationObserver 52
Map 52
MapState 55
Memento 56
MiddleStage 59
MiddleStrategy 60
Military 62
MilitaryState 63
Neighbour 66
Plane 68
PlaneFactory 69
RightNeighbour 70
Ship 72
ShipFactory 73
SimulationManager 75
SimulationState 78
StageContext 81
StageContextState 85
Strategy 87
Superpower 89
SuperpowerState 93
Tank 96

TankFactory 97
Territory 98
TopNeighbour 99
Vehicle 101
VehicleFactory 102
WarStage 104

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Backup.h . . . . . ??
Battalion.h . . . . . ??
BottomNeighbour.h . . . . . ??
Country.h . . . . . ??
CountryState.h . . . . . ??
EarlyStage.h . . . . . ??
EarlyStrategy.h . . . . . ??
Iterator.h . . . . . ??
LateStage.h . . . . . ??
LateStrategy.h . . . . . ??
LeftNeighbour.h . . . . . ??
Location.h . . . . . ??
LocationIterator.h . . . . . ??
LocationObserver.h . . . . . ??
Map.h . . . . . ??
MapState.h . . . . . ??
Memento.h . . . . . ??
MiddleStage.h . . . . . ??
MiddleStrategy.h . . . . . ??
Military.h . . . . . ??
MilitaryState.h . . . . . ??
Neighbour.h . . . . . ??
Plane.h . . . . . ??
PlaneFactory.h . . . . . ??
RightNeighbour.h . . . . . ??
Ship.h . . . . . ??
ShipFactory.h . . . . . ??
SimulationManager.h . . . . . ??
SimulationState.h . . . . . ??
StageContext.h . . . . . ??
StageContextState.h . . . . . ??
Strategy.h . . . . . ??
Superpower.h . . . . . ??
SuperpowerState.h . . . . . ??
Tank.h . . . . . ??

TankFactory.h . . . . . ??
Territory.h . . . . . ??
TopNeighbour.h . . . . . ??
Vehicle.h . . . . . ??
VehicleFactory.h . . . . . ??
WarStage.h . . . . . ??

Chapter 5

Class Documentation

5.1 Backup Class Reference

Public Member Functions

- Backup ()

  Construct a new Backup object.
- ~Backup ()

  Destroy the Backup object and all its Mementos.
- void addMemento (Memento *_memento)

  Add a new memento to the backup.
- Memento * getMemento ()

  Get the last added Memento object and remove it from the backup.
- int getMementoCount ()

  Get the number of Mementos currently stored in the backup.
- void clear ()

  Delete all of the Mementos stored in the backup.

5.1.1 Constructor & Destructor Documentation

5.1.1.1 Backup()

```
Backup::Backup ( )
```

Construct a new Backup object.

5.1.1.2 ~Backup()

```
Backup::~Backup ( )
```

Destroy the Backup object and all its Mementos.

5.1.2 Member Function Documentation

5.1.2.1 addMemento()

```
void Backup::addMemento (
            Memento * _memento )
```

Add a new memento to the backup.

Parameters

_memento	: Memento∗ - the memento to add

5.1.2.2 clear()

```
void Backup::clear ( )
```

Delete all of the Mementos stored in the backup.

5.1.2.3 getMemento()

```
Memento * Backup::getMemento ( )
```

Get the last added Memento object and remove it from the backup.

Exceptions : std::out_of_range if the backup is empty

Returns

Memento∗ - pointer to the last added Memento

5.1.2.4 getMementoCount()

```
int Backup::getMementoCount ( )
```

Get the number of Mementos currently stored in the backup.

Returns 0 if the backup is empty

Returns

int

The documentation for this class was generated from the following files:

- Backup.h
- Backup.cpp

5.2 Battalion Class Reference

Public Member Functions

- **Battalion** ()

  construct a Battalion object
- Battalion (int)

  construct a Battalion object
- virtual ~**Battalion** ()

  destroys a Battallion object
- void attack (Country *enemy)

  attack method towards the enemy Country
- void setNumBattalionDestroys (int)

  sets the number Battalions an attack of a battallion kills.
- int getBattalionDestroyed ()

  retrieve number of battallions the battablion can destroy

5.2.1 Constructor & Destructor Documentation

5.2.1.1 Battalion()

```
Battalion::Battalion (
            int damage )
```

construct a Battalion object

Parameters

int	value

5.2.2 Member Function Documentation

5.2.2.1 attack()

```
void Battalion::attack (
            Country * enemy )
```

attack method towards the enemy Country

Parameters

Country-	Country reference object

5.2.2.2 getBattalionDestroyed()

```
int Battalion::getBattalionDestroyed ( )
```

retrieve number of battallions the battablion can destroy

Returns

int - number of Battalions

5.2.2.3 setNumBattalionDestroys()

```
void Battalion::setNumBattalionDestroys (
            int n )
```

sets the number Battalions an attack of a battallion kills.

Parameters

int	of the number of Battalions

The documentation for this class was generated from the following files:

- Battalion.h
- Battalion.cpp

5.3 BottomNeighbour Class Reference

```
#include <BottomNeighbour.h>
```

Inheritance diagram for BottomNeighbour:

Public Member Functions

- BottomNeighbour (Location *_neighbour)

  The constructor for bottom neighbour.
- Location * getBottom ()

  Returns bottom neighbour.
- bool hasBottom ()

  Always returns true since BottomNeighbour will always have a bottom neighbour.

Additional Inherited Members

5.3.1 Detailed Description

Author

Julian Pienaar

5.3.2 Constructor & Destructor Documentation

5.3.2.1 BottomNeighbour()

```
BottomNeighbour::BottomNeighbour (
            Location * _neighbour )
```

The constructor for bottom neighbour.

Parameters

_neighbour	: Location* - Pointer to the bottom neighbour.

5.3.3 Member Function Documentation

5.3.3.1 getBottom()

```
Location * BottomNeighbour::getBottom ( )  [virtual]
```

Returns bottom neighbour.

Returns

Location*

Reimplemented from Location.

5.3.3.2 hasBottom()

```
bool BottomNeighbour::hasBottom ( )  [virtual]
```

Always returns true since BottomNeighbour will always have a bottom neighbour.

Returns

true

Reimplemented from Location.

The documentation for this class was generated from the following files:

- BottomNeighbour.h
- BottomNeighbour.cpp

5.4 Country Class Reference

Public Member Functions

- ~**Country** ()

  destructor of Country objects
- Country (std::string _name)

  parameterised constructor for Country objects
- **Country** ()

  default constructor for Country objects
- void takeTurn (Country *countryB)

  uses state information to implement this country's next turn
- Country * takeTurn (bool *_countryIsDead)

  country takes its turn using a country from its enemies list
- void **setStrategy** ()

  sets this country's strategy based on the current war stage of the simulation
- CountryState * getState ()

  getter for this country's CountryState object
- std::string getName ()

  getter for the name of this country
- void setName (std::string _name)

  setter for the the name of this country
- Military * getMilitary ()

  getter for this country's military
- void getCountryRating (Country *countryB, double *strengthRatings)

  generates a countries strengthRating based on various state comparisons with enemy
- double compareAspect (int countryA, int countryB)

  compares two state paramters and returns countryA's advantage
- double compareAspect (double countryA, double countryB)

  compares two state paramters and returns countryA's advantage
- int getNumCitizens ()

  getter for numCitizens attribute

- void setNumCitizens (int _numCitizens)

  sets for the numCitizens attribute
- double getPoliticalStability ()

  getter for this country's politicalStability attribute
- void setPoliticalStability (double _politicalStability)

  setter for this country's politicalStability attribute
- double getDomesticMorale ()

  getter for domesticMorale attribute
- void setDomesticMorale (double _domesticMorale)

  sets the value of class attribute domesticMorale
- double getSelfReliance ()

  getter for selfReliance attribute
- void setSelfReliance (double _selfReliance)

  sets the value of class attribute selfReliance
- double getBorderStrength ()

  getter for borderStrength attribute
- void setBorderStrength (double _borderStrength)

  sets the value of class attribute borderStrength
- double getCapitalSafety ()

  getter for capitalSafety attribute, which is how safe the capital is from enemies
- void setCapitalSafety (double _capitalSafety)

  sets the value of class attribute capitalSafety
- double getWarSentiment ()

  getter for warSentiment attribute
- void setWarSentiment (double _warSentiment)

  sets the value of class attribute warSentiment
- double getTradeRouteSafety ()

  getter for tradeRouteSafety attribute
- void setTradeRouteSafety (double _tradeRouteSafety)

  sets the value of class attribute tradeRouteSafety
- CountryState ∗ getCountryState ()

  getter for this country's strategy object
- void setCountryState (CountryState ∗_countryState)

  setter for this country's state
- void compareMilitary (Country ∗a, Country ∗b, std::vector< double > ∗aspectScores)

  compares the various aspects of two countries' MilitaryState objects
- void compareDomestic (Country ∗a, Country ∗b, std::vector< double > ∗aspectScores)

  compares the various aspects of two countries' CountryState objects
- Location ∗ getCapital ()

  getter for this country's capital Location object
- void setCapital (Location ∗_capital)

  setter for this country's capital
- std::vector< Location ∗ > ∗ getLocations ()

  getter for this country's locations
- void **setLocations** (std::vector< Location ∗ > ∗_locations)

  setter for this country's locations, performs a shallow copy of the passed in locations vector
- void setColor (std::string _color)

  setter for this country's color
- std::string getColor ()

  getter for this country's color
- std::vector< Country ∗ > ∗ getEnemies ()

getter for this country's enemies

- void setEnemies (std::vector< Country * > *_enemies)

  setter for this country's enemies
- MilitaryState * **getMilitaryState** ()

  getter for this country's MilitaryState object
- void **setMilitaryState** (MilitaryState *_militaryState)

  setter for this country's MilitaryState object
- void setState (CountryState *_state)

  settervfor the country's CountryState object
- void **printSummary** ()

  print a summary of this country's state
- void attachObserver (LocationObserver *_lObserver)

  subscribes an observer to changes in this country's locations
- void detachObserver (LocationObserver *_lObserver)

  detaches an observer of this country
- void resetLocations (Map *_map)

  resets this country's locations to a passed-in state
- Country * clone ()

  creates a clone of this country
- void resetEnemies (std::vector< Country * > *_enemies)

  resets this country's enemies to a previous state
- void removeEnemy (Country *_enemy)

  removes an enemy from this country's enemies vector
- void setColorOfDestroyedBy (std::string _newColorOfDestroyedBy)

  sets the color of the colorOfDestroyedBy attribute
- bool checkIsDead (Country *countryA, Country *countryB)

  evaluates whether countryA was defeated by countryB
- void getBoost (Country *_country)

  boost all of the country's attributes by a fixed amount

5.4.1 Constructor & Destructor Documentation

5.4.1.1 Country()

```
Country::Country (
            std::string _name )
```

parameterised constructor for Country objects

Parameters

_name	the name of this country

5.4.2 Member Function Documentation

5.4.2.1 attachObserver()

```
void Country::attachObserver (
            LocationObserver * _lObserver )
```

subscribes an observer to changes in this country's locations

Parameters

observer	the observer to attach

5.4.2.2 checkIsDead()

```
bool Country::checkIsDead (
            Country * countryA,
            Country * countryB )
```

evaluates whether countryA was defeated by countryB

Parameters

countryA	the country that is checked to see if it was defeated
countryB	the country that is checked to see if it defeated countryA

Returns

true if countryA was defeated by countryB, false otherwise

5.4.2.3 clone()

```
Country * Country::clone ( )
```

creates a clone of this country

Returns

a pointer to the newly cloned Country object

5.4.2.4 compareAspect() [1/2]

```
double Country::compareAspect (
            double countryA,
            double countryB )
```

compares two state paramters and returns countryA's advantage

Parameters

countryA	state parameter of countryA
countryB	state parameter of countryB

Returns

a comparable value of the advantage of countryA over countryB

5.4.2.5 compareAspect() [2/2]

```
double Country::compareAspect (
            int countryA,
            int countryB )
```

compares two state paramters and returns countryA's advantage

Parameters

countryA	state parameter of countryA
countryB	state parameter of countryB

Returns

a comparable value of the advantage of countryA over countryB

5.4.2.6 compareDomestic()

```
void Country::compareDomestic (
            Country * a,
            Country * b,
            std::vector< double > * aspectScores )
```

compares the various aspects of two countries' CountryState objects

Parameters

a	the country implementing a strategy against b
b	the country being attacked by a
aspectScores	vector of this country's scores across each CountryState attribute

5.4.2.7 compareMilitary()

```
void Country::compareMilitary (
            Country * a,
            Country * b,
            std::vector< double > * aspectScores )
```

compares the various aspects of two countries' MilitaryState objects

Parameters

a	the country implementing a strategy against b
b	the country being attacked by a
aspectScores	vector of this country's scores across each MilitaryState attribute

5.4.2.8 detachObserver()

```
void Country::detachObserver (
            LocationObserver * _lObserver )
```

detaches an observer of this country

Parameters

_lObserver	the observer to detach

5.4.2.9 getBoost()

```
void Country::getBoost (
            Country * _country )
```

boost all of the country's attributes by a fixed amount

Parameters

_country	: Country* - The country to boost

5.4.2.10 getBorderStrength()

```
double Country::getBorderStrength ( )
```

getter for borderStrength attribute

Returns

current borderStrength of this country

5.4.2.11 getCapital()

```
Location * Country::getCapital ( )
```

getter for this country's capital Location object

Returns

this country's capital Location object

5.4.2.12 getCapitalSafety()

```
double Country::getCapitalSafety ( )
```

getter for capitalSafety attribute, which is how safe the capital is from enemies

Returns

current capitalSafety of this country

5.4.2.13 getColor()

```
std::string Country::getColor ( )
```

getter for this country's color

Returns

this country's color

5.4.2.14 getCountryRating()

```
void Country::getCountryRating (
            Country * countryB,
            double * strengthRatings )
```

generates a countries strengthRating based on various state comparisons with enemy

Parameters

countryB	the country that this country is implementing a strategy against
strengthRatings	the comparable strength ratings of this country and countryB

Returns

the strength rating of this country

5.4.2.15 getCountryState()

```
CountryState * Country::getCountryState ( )
```

getter for this country's strategy object

Returns

this country's strategy object

5.4.2.16 getDomesticMorale()

```
double Country::getDomesticMorale ( )
```

getter for domesticMorale attribute

Returns

current domesticMorale of this country

5.4.2.17 getEnemies()

```
std::vector< Country * > * Country::getEnemies ( )
```

getter for this country's enemies

Returns

a pointer to the vector of this country's enemies

5.4.2.18 getLocations()

```
std::vector< Location * > * Country::getLocations ( )
```

getter for this country's locations

Returns

vector of this country's locations

5.4.2.19 getMilitary()

```
Military * Country::getMilitary ( )
```

getter for this country's military

Returns

this country's Military object

5.4.2.20 getName()

```
std::string Country::getName ( )
```

getter for the name of this country

Returns

the name of this country

5.4.2.21 getNumCitizens()

```
int Country::getNumCitizens ( )
```

getter for numCitizens attribute

Returns

current number of citizens of this country

5.4.2.22 getPoliticalStability()

```
double Country::getPoliticalStability ( )
```

getter for this country's politicalStability attribute

Returns

current politicalStability of this country

5.4.2.23 getSelfReliance()

```
double Country::getSelfReliance ( )
```

getter for selfReliance attribute

Returns

current self reliance of this country

5.4.2.24 getState()

```
CountryState * Country::getState ( )
```

getter for this country's CountryState object

Returns

a pointer to this country's CountryState object

5.4.2.25 getTradeRouteSafety()

```
double Country::getTradeRouteSafety ( )
```

getter for tradeRouteSafety attribute

Returns

current tradeRouteSafety of this country

5.4.2.26 getWarSentiment()

```
double Country::getWarSentiment ( )
```

getter for warSentiment attribute

Returns

current warSentiment of this country

5.4.2.27 removeEnemy()

```
void Country::removeEnemy (
            Country * _enemy )
```

removes an enemy from this country's enemies vector

Parameters

_enemy	enemy to be removed

5.4.2.28 resetEnemies()

```
void Country::resetEnemies (
            std::vector< Country * > * _enemies )
```

resets this country's enemies to a previous state

Parameters

_enemies	a vector of enemies to replace this country's current enemies

5.4.2.29 resetLocations()

```
void Country::resetLocations (
            Map * _map )
```

resets this country's locations to a passed-in state

Parameters

_map	source of locations to reset to

5.4.2.30 setBorderStrength()

```
void Country::setBorderStrength (
            double _borderStrength )
```

sets the value of class attribute borderStrength

Parameters

_borderStrength	new value of borderStrength

5.4.2.31 setCapital()

```
void Country::setCapital (
            Location * _capital )
```

setter for this country's capital

Parameters

_capital	new capital Location object

5.4.2.32 setCapitalSafety()

```
void Country::setCapitalSafety (
            double _capitalSafety )
```

sets the value of class attribute capitalSafety

Parameters

_capitalSafety	new value of capitalSafety

5.4.2.33 setColor()

```
void Country::setColor (
            std::string _color )
```

setter for this country's color

Parameters

_color	: std::string - The color this country should be printed as on the map

5.4.2.34 setColorOfDestroyedBy()

```
void Country::setColorOfDestroyedBy (
            std::string _newColorOfDestroyedBy )
```

sets the color of the colorOfDestroyedBy attribute

Parameters

_newColorOfDestroyedBy	color of the country that destroyed this country

5.4.2.35 setCountryState()

```
void Country::setCountryState (
            CountryState * _countryState )
```

setter for this country's state

Parameters

_countryState	new countryState

5.4.2.36 setDomesticMorale()

```
void Country::setDomesticMorale (
            double _domesticMorale )
```

sets the value of class attribute domesticMorale

Parameters

_domesticMorale	new value of domesticMorale

5.4.2.37 setEnemies()

```
void Country::setEnemies (
            std::vector< Country * > * _enemies )
```

setter for this country's enemies

Parameters

_enemies	: std::vector<Country *> - The new vector of enemies for this country

5.4.2.38 setName()

```
void Country::setName (
            std::string _name )
```

setter for the the name of this country

Parameters

_name	the name of this country

5.4.2.39 setNumCitizens()

```
void Country::setNumCitizens (
            int _numCitizens )
```

sets for the numCitizens attribute

Parameters

_numCitizens	new value of numCitizens

5.4.2.40 setPoliticalStability()

```
void Country::setPoliticalStability (
            double _politicalStability )
```

setter for this country's politicalStability attribute

Parameters

_politicalStability	new value for this country's politicalStability attribute

5.4.2.41 setSelfReliance()

```
void Country::setSelfReliance (
            double _selfReliance )
```

sets the value of class attribute selfReliance

Parameters

_selfReliance	new value of selfReliance

5.4.2.42 setState()

```
void Country::setState (
            CountryState * _state )
```

settervfor the country's CountryState object

Parameters

_state	: CountryState∗ - The new state object

5.4.2.43 setTradeRouteSafety()

```
void Country::setTradeRouteSafety (
            double _tradeRouteSafety )
```

sets the value of class attribute tradeRouteSafety

Parameters

_tradeRouteSafety	new value of tradeRouteSafety

5.4.2.44 setWarSentiment()

```
void Country::setWarSentiment (
            double _warSentiment )
```

sets the value of class attribute warSentiment

Parameters

_warSentiment	new value of warSentiment

5.4.2.45 takeTurn() [1/2]

```
Country * Country::takeTurn (
            bool * _countryIsDead )
```

country takes its turn using a country from its enemies list

Parameters

_countryIsDead	whether this country was defeated after this turn

5.4.2.46 takeTurn() [2/2]

```
void Country::takeTurn (
            Country * countryB )
```

uses state information to implement this country's next turn

Parameters

countryB	the country that is being attacked

The documentation for this class was generated from the following files:

- Country.h
- Country.cpp

5.5 CountryState Class Reference

Public Member Functions

- CountryState ()

Create new country state.

- CountryState (Country *country)

  Construct a new Country State object.
- CountryState (const CountryState &cs)

  Construct a new Country State object.
- ~CountryState ()

  Destroy the Country State object.
- CountryState * clone ()

  Create a clone of the held country state and return it.
- MilitaryState * getMilitaryState ()

  Get the Military State object.
- void setMilitaryState (MilitaryState *_militaryState)

  Set the Military State object.
- void **setIsBeingStored** (bool _isBeingStored)

Friends

- class **Country**

5.5.1 Constructor & Destructor Documentation

5.5.1.1 CountryState() [1/3]

```
CountryState::CountryState ( )
```

Create new country state.

Parameters

country	country to create state for

5.5.1.2 CountryState() [2/3]

```
CountryState::CountryState (
            Country * country )
```

Construct a new Country State object.

Parameters

country	

5.5.1.3 CountryState() [3/3]

```
CountryState::CountryState (
            const CountryState & cs )
```

Construct a new Country State object.

Parameters

cs	

5.5.1.4 ~CountryState()

```
CountryState::~CountryState ( )
```

Destroy the Country State object.

5.5.2 Member Function Documentation

5.5.2.1 clone()

```
CountryState * CountryState::clone ( )
```

Create a clone of the held country state and return it.

Returns

CountryState*

5.5.2.2 getMilitaryState()

```
MilitaryState * CountryState::getMilitaryState ( )
```

Get the Military State object.

Returns

MilitaryState*

5.5.2.3 setMilitaryState()

```
void CountryState::setMilitaryState (
            MilitaryState * _militaryState )
```

Set the Military State object.

Parameters

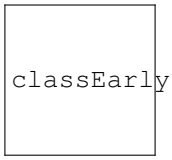
_militaryState	

The documentation for this class was generated from the following files:

- CountryState.h
- CountryState.cpp

5.6 EarlyStage Class Reference

Inheritance diagram for EarlyStage:



Public Member Functions

- int getWarStage ()

  returns warstage via an int = 0
- EarlyStage ()

  Construct a new Early Stage object.
- ~EarlyStage ()

  Destroy the Early Stage object.
- EarlyStage * clone ()

  Return a deep copy of the Early Stage object.

5.6.1 Constructor & Destructor Documentation

5.6.1.1 EarlyStage()

```
EarlyStage::EarlyStage ( )
```

Construct a new Early Stage object.

5.6.1.2 ~EarlyStage()

```
EarlyStage::~EarlyStage ( )
```

Destroy the Early Stage object.

5.6.2 Member Function Documentation

5.6.2.1 clone()

```
EarlyStage * EarlyStage::clone ( ) [virtual]
```

Return a deep copy of the Early Stage object.

Returns

EarlyStage∗

Implements WarStage.

5.6.2.2 getWarStage()

```
int EarlyStage::getWarStage ( ) [virtual]
```

returns warstage via an int = 0

Implements WarStage.

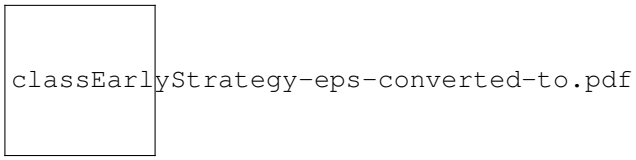
The documentation for this class was generated from the following files:

- EarlyStage.h
- EarlyStage.cpp

5.7 EarlyStrategy Class Reference

```
#include <EarlyStrategy.h>
```

Inheritance diagram for EarlyStrategy:



Public Member Functions

- **EarlyStrategy** ()

  Constructor.
- ∼**EarlyStrategy** ()

  destructor

Protected Member Functions

- void defensiveMove (Country *countryA, Country *countryB)

  virtual function representing the implementation of a turn when Country A is weaker than Country B
- void neutralMove (Country *countryA, Country *countryB)

  virtual function representing the implementation of a turn when Country A and Country B are similar in strength
- void offensiveMove (Country *countryA, Country *countryB)

  virtual function representing the implementation of a turn when Country A is stronger than Country B

5.7.1 Detailed Description

Author

Mekhail Muller

5.7.2 Member Function Documentation

5.7.2.1 defensiveMove()

```
void EarlyStrategy::defensiveMove (
            Country * countryA,
            Country * countryB )  [protected], [virtual]
```

virtual function representing the implementation of a turn when Country A is weaker than Country B

Parameters

countryA	the country that is making the move (calling country)
countryB	the country being attacked by calling country

Implements Strategy.

5.7.2.2 neutralMove()

```
void EarlyStrategy::neutralMove (
            Country * countryA,
            Country * countryB )  [protected], [virtual]
```

virtual function representing the implementation of a turn when Country A and Country B are similar in strength

Parameters

countryA	the country that is making the move (calling country)
countryB	the country being attacked by calling country

Implements Strategy.

5.7.2.3 offensiveMove()

```
void EarlyStrategy::offensiveMove (
            Country * countryA,
            Country * countryB )  [protected], [virtual]
```

virtual function representing the implementation of a turn when Country A is stronger than Country B

Parameters

countryA	the country that is making the move (calling country)
countryB	the country being attacked by calling country

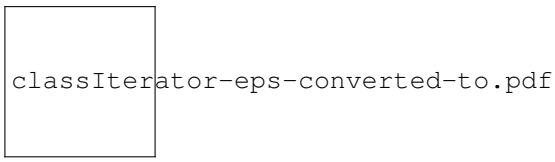
Implements Strategy.

The documentation for this class was generated from the following files:

- EarlyStrategy.h
- EarlyStrategy.cpp

5.8 Iterator Class Reference

Inheritance diagram for Iterator:



Public Member Functions

- virtual void next ()=0

  Sets current to the next location in the sequential iteration of the map.
- virtual void first ()=0

  Sets current to the top left location of the map.
- virtual bool isDone ()=0

  Returns true if sequential iteration is complete and false otherwise.
- virtual Location * getCurrent ()=0

  Returns the pointer to the current location.

Protected Attributes

- Location * **current**

5.8.1 Member Function Documentation

5.8.1.1 first()

```
virtual void Iterator::first ( ) [pure virtual]
```

Sets current to the top left location of the map.

Implemented in LocationIterator.

5.8.1.2 getCurrent()

```
virtual Location * Iterator::getCurrent ( ) [pure virtual]
```

Returns the pointer to the current location.

Returns

Location*

Implemented in LocationIterator.

5.8.1.3 isDone()

```
virtual bool Iterator::isDone ( ) [pure virtual]
```

Returns true if sequential iteration is complete and false otherwise.

Returns

boolean

Implemented in LocationIterator.

5.8.1.4 next()

```
virtual void Iterator::next ( ) [pure virtual]
```

Sets current to the next location in the sequential iteration of the map.

Exceptions : out_of_range if there is no next location.

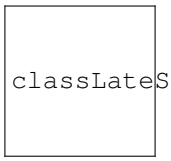
Implemented in LocationIterator.

The documentation for this class was generated from the following file:

- Iterator.h

5.9 LateStage Class Reference

Inheritance diagram for LateStage:



Public Member Functions

- int getWarStage ()

  returns warstage via int = 2
- LateStage ()

  Construct a new Late Stage object.
- ∼LateStage ()

  Destroy the Late Stage object.
- LateStage ∗ clone ()

  Return a deep copy of the Late Stage object.

5.9.1 Constructor & Destructor Documentation

5.9.1.1 LateStage()

```
LateStage::LateStage ( )
```

Construct a new Late Stage object.

5.9.1.2 ∼LateStage()

```
LateStage::∼LateStage ( )
```

Destroy the Late Stage object.

5.9.2 Member Function Documentation

5.9.2.1 clone()

```
LateStage * LateStage::clone ( ) [virtual]
```

Return a deep copy of the Late Stage object.

Returns

LateStage*

Implements WarStage.

5.9.2.2 getWarStage()

```
int LateStage::getWarStage ( ) [virtual]
```

returns warstage via int = 2

Implements WarStage.

The documentation for this class was generated from the following files:

- LateStage.h
- LateStage.cpp

5.10 LateStrategy Class Reference

```
#include <LateStrategy.h>
```

Inheritance diagram for LateStrategy:

Public Member Functions

- LateStrategy ()

  Construct a new Late Strategy object.
- ~**LateStrategy** ()

  destructor

Protected Member Functions

- void defensiveMove (Country *countryA, Country *countryB)

  virtual function representing the implementation of a turn when Country A is weaker than Country B
- void neutralMove (Country *countryA, Country *countryB)

  virtual function representing the implementation of a turn when Country A and Country B are similar in strength
- void offensiveMove (Country *countryA, Country *countryB)

  virtual function representing the implementation of a turn when Country A is stronger than Country B

5.10.1 Detailed Description

Author

Mekhail Muller

5.10.2 Constructor & Destructor Documentation

5.10.2.1 LateStrategy()

```
LateStrategy::LateStrategy ( )
```

Construct a new Late Strategy object.

5.10.3 Member Function Documentation

5.10.3.1 defensiveMove()

```
void LateStrategy::defensiveMove (
            Country * countryA,
            Country * countryB )  [protected], [virtual]
```

virtual function representing the implementation of a turn when Country A is weaker than Country B

Parameters

countryA	the country that is making the move (calling country)
countryB	the country being attacked by calling country

Implements Strategy.

5.10.3.2 neutralMove()

```
void LateStrategy::neutralMove (
            Country * countryA,
            Country * countryB )  [protected], [virtual]
```

virtual function representing the implementation of a turn when Country A and Country B are similar in strength

Parameters

countryA	the country that is making the move (calling country)
countryB	the country being attacked by calling country

Implements Strategy.

5.10.3.3 offensiveMove()

```
void LateStrategy::offensiveMove (
            Country * countryA,
            Country * countryB )  [protected], [virtual]
```

virtual function representing the implementation of a turn when Country A is stronger than Country B

Parameters

countryA	the country that is making the move (calling country)
countryB	the country being attacked by calling country

Implements Strategy.

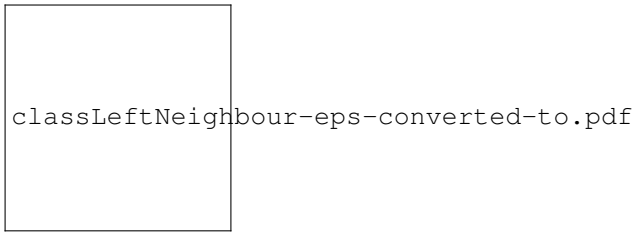
The documentation for this class was generated from the following files:

- LateStrategy.h
- LateStrategy.cpp

5.11 LeftNeighbour Class Reference

```
#include <LeftNeighbour.h>
```

Inheritance diagram for LeftNeighbour:



Public Member Functions

- LeftNeighbour (Location *_neighbour)

  The constructor for left neighbour.
- Location * getLeft ()

  Returns left neighbour.
- bool hasLeft ()

  Always returns true since LeftNeighbour will always have a left neighbour.

Additional Inherited Members

5.11.1 Detailed Description

Author

Julian Pienaar

5.11.2 Constructor & Destructor Documentation

5.11.2.1 LeftNeighbour()

```
LeftNeighbour::LeftNeighbour (
            Location * _neighbour )
```

The constructor for left neighbour.

Parameters

_neighbour	: Location* - Pointer to the left neighbour.

5.11.3 Member Function Documentation

5.11.3.1 getLeft()

```
Location * LeftNeighbour::getLeft ( )  [virtual]
```

Returns left neighbour.

Returns

Location*

Reimplemented from Location.

5.11.3.2 hasLeft()

```
bool LeftNeighbour::hasLeft ( ) [virtual]
```

Always returns true since LeftNeighbour will always have a left neighbour.

Returns

true

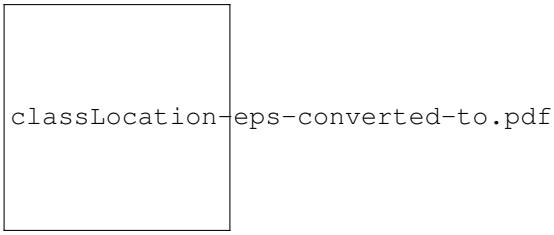
Reimplemented from Location.

The documentation for this class was generated from the following files:

- LeftNeighbour.h
- LeftNeighbour.cpp

5.12 Location Class Reference

Inheritance diagram for Location:



Public Member Functions

- virtual ~Location ()

  Destructor for the Location class.
- Iterator * createIterator ()

  Returns a new Iterator object with current set to this Location.
- virtual Location * getRight ()

  If location is NULL throw exception else call getRight on the location object.
- virtual Location * getLeft ()

  If location is NULL throw exception else call getLeft on the location object.
- virtual Location * getTop ()

  If location is NULL throw exception else call getTop on the location object.
- virtual Location * getBottom ()

  If location is NULL throw exception else call getBottom on the location object.
- virtual void add (Location *_neighbour)=0

  Abstract function specifying how to add neighbour to a location.
- virtual bool hasBottom ()

  Return false if location is NULL and call hasBottom on location otherwise.
- virtual bool hasRight ()

  Return false if location is NULL and call hasRight on location otherwise.

- virtual bool hasLeft ()

  Return false if location is NULL and call hasLeft on location otherwise.
- virtual bool hasTop ()

  Return false if location is NULL and call hasTop on location otherwise.
- Country ∗ getOwnedBy ()

  Return a pointer to the country which owns this territory.
- void setOwnedBy (Country ∗_newOwner)

  Set which country owns this territory.
- Location ∗ clone ()

  Create a copy of this locations attributes except for anything to do with it's neighbour.
- std::string getColor ()

  Return the colour attribute.
- void setColor (std::string _color)

  Set the Colour attribute.
- int getX ()

  Return the xCoordinate value.
- int getY ()

  Return the yCoordinate value.
- bool getIsLand ()

  Get the Is Land attribute.
- void setIsLand (bool _isLand)

  Set the Is Land attribute.
- bool getIsCapital ()

  Get the Is Capital attribute.
- void setIsCapital (bool _isCapital)

  Set the Is Capital attribute.

Protected Attributes

- Location ∗ **location**
- Country ∗ **ownedBy**
- LocationObserver ∗ **lObserver**
- std::string **color**
- bool **isCapital**
- bool **isLand**
- int **xCoordinate**
- int **yCoordinate**

5.12.1 Constructor & Destructor Documentation

5.12.1.1 ∼Location()

```
Location::~Location ( ) [virtual]
```

Destructor for the Location class.

5.12.2 Member Function Documentation

5.12.2.1 add()

```
virtual void Location::add (
            Location * _neighbour )  [pure virtual]
```

Abstract function specifying how to add neighbour to a location.

Parameters

_neighbour	: Location∗ - Pointer to the neighbour to be added.

Implemented in Neighbour, and Territory.

5.12.2.2 clone()

```
Location * Location::clone ( )
```

Create a copy of this locations attributes except for anything to do with it's neighbour.

Returns

Location∗

5.12.2.3 createIterator()

```
Iterator * Location::createIterator ( )
```

Returns a new Iterator object with current set to this Location.

Returns

Iterator∗

5.12.2.4 getBottom()

```
Location * Location::getBottom ( ) [virtual]
```

If location is NULL throw exception else call getBottom on the location object.

Exceptions : std::__throw_out_of_range if this location has no bottom neighbour.

Returns

Location*

Reimplemented in BottomNeighbour.

5.12.2.5 getColor()

```
string Location::getColor ( )
```

Return the colour attribute.

Returns

char

5.12.2.6 getIsCapital()

```
bool Location::getIsCapital ( )
```

Get the Is Capital attribute.

Returns

bool

5.12.2.7 getIsLand()

```
bool Location::getIsLand ( )
```

Get the Is Land attribute.

Returns

bool

5.12.2.8 getLeft()

```
Location * Location::getLeft ( ) [virtual]
```

If location is NULL throw exception else call getLeft on the location object.

Exceptions : std::__throw_out_of_range if this location has no left neighbour.

Returns

Location*

Reimplemented in LeftNeighbour.

5.12.2.9 getOwnedBy()

```
Country * Location::getOwnedBy ( )
```

Return a pointer to the country which owns this territory.

Returns

Country*

5.12.2.10 getRight()

```
Location * Location::getRight ( ) [virtual]
```

If location is NULL throw exception else call getRight on the location object.

Exceptions : std::__throw_out_of_range if this location has no right neighbour.

Returns

Location*

Reimplemented in RightNeighbour.

5.12.2.11 getTop()

```
Location * Location::getTop ( ) [virtual]
```

If location is NULL throw exception else call getTop on the location object.

Exceptions : std::__throw_out_of_range if this location has no top neighbour.

Returns

Location*

Reimplemented in TopNeighbour.

5.12.2.12 getX()

```
int Location::getX ( )
```

Return the xCoordinate value.

Returns

int

5.12.2.13 getY()

```
int Location::getY ( )
```

Return the yCoordinate value.

Returns

int

5.12.2.14 hasBottom()

```
bool Location::hasBottom ( ) [virtual]
```

Return false if location is NULL and call hasBottom on location otherwise.

Returns

true

false

Reimplemented in BottomNeighbour.

5.12.2.15 hasLeft()

```
bool Location::hasLeft ( ) [virtual]
```

Return false if location is NULL and call hasLeft on location otherwise.

Returns

true

false

Reimplemented in LeftNeighbour.

5.12.2.16 hasRight()

```
bool Location::hasRight ( ) [virtual]
```

Return false if location is NULL and call hasRight on location otherwise.

Returns

true

false

Reimplemented in RightNeighbour.

5.12.2.17 hasTop()

```
bool Location::hasTop ( ) [virtual]
```

Return false if location is NULL and call hasTop on location otherwise.

Returns

true

false

Reimplemented in TopNeighbour.

5.12.2.18 setColor()

```
void Location::setColor (
            std::string _color )
```

Set the Colour attribute.

Parameters

_colour	: char - variable to set colour to.

5.12.2.19 setIsCapital()

```
void Location::setIsCapital (
            bool _isCapital )
```

Set the Is Capital attribute.

Parameters

_isCapital	: bool - variable to set isCapital to.

5.12.2.20 setIsLand()

```
void Location::setIsLand (
            bool _isLand )
```

Set the Is Land attribute.

Parameters

_isLand	: bool - variable to set isLand to.

5.12.2.21 setOwnedBy()

```
void Location::setOwnedBy (
            Country * _newOwner )
```

Set which country owns this territory.

Parameters

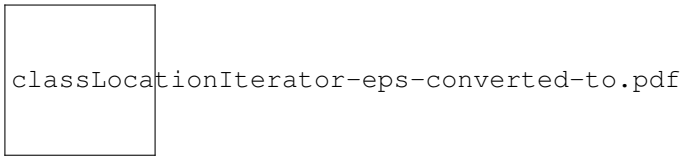
_newOwner	: Country* - the pointer to the new owner of the territory.

The documentation for this class was generated from the following files:

- Location.h
- Location.cpp

5.13 LocationIterator Class Reference

Inheritance diagram for LocationIterator:



Public Member Functions

- LocationIterator (Location *_location)

  Construct a new Location Iterator object.
- ~LocationIterator ()

  Destroy the Location Iterator object.
- void next ()

  Sets current to the next location in the sequential iteration of the map.
- void first ()

  Sets current to the top left location of the map.
- bool isDone ()

  Returns true if sequential iteration is complete and false otherwise.
- Location * getCurrent ()

  Returns the pointer to the current location.

Protected Member Functions

- bool **hasNext** ()
- Location * **nextRow** ()

Protected Attributes

- Location * **current**
- Location * **nextLocation**

5.13.1 Constructor & Destructor Documentation

5.13.1.1 LocationIterator()

```
LocationIterator::LocationIterator (
            Location * _location )
```

Construct a new Location Iterator object.

Parameters

_location	: Location∗ - pointer to the location that is set as the current location.

5.13.1.2 ∼LocationIterator()

```
LocationIterator::~LocationIterator ( )
```

Destroy the Location Iterator object.

5.13.2 Member Function Documentation

5.13.2.1 first()

```
void LocationIterator::first ( ) [virtual]
```

Sets current to the top left location of the map.

Implements Iterator.

5.13.2.2 getCurrent()

```
Location * LocationIterator::getCurrent ( ) [virtual]
```

Returns the pointer to the current location.

Returns

Location∗

Implements Iterator.

5.13.2.3 isDone()

```
bool LocationIterator::isDone ( ) [virtual]
```

Returns true if sequential iteration is complete and false otherwise.

Returns

boolean

Implements Iterator.

5.13.2.4 next()

```
void LocationIterator::next ( ) [virtual]
```

Sets current to the next location in the sequential iteration of the map.

Exceptions : out_of_range if there is no next location.

Implements Iterator.

The documentation for this class was generated from the following files:

- LocationIterator.h
- LocationIterator.cpp

5.14 LocationObserver Class Reference

Public Member Functions

- **LocationObserver** (Location *_location)
- void **updateLocation** (std::string _newColor)

The documentation for this class was generated from the following files:

- LocationObserver.h
- LocationObserver.cpp

5.15 Map Class Reference

Public Member Functions

- Map ()

  Construct a new Map object initializing all locations in the Map.
- Map (Location *_cloneTopLeft)

  Construct a new Map object and set topLeft to _cloneTopLeft.
- ~Map ()

  Call delete on each location in the Map using an iterator.
- Map (Map *_oldMap)

  Construct a copy of the passed in map pointer.
- Location * getLocation (int _x, int _y)

  Get the Location object with matching x and y coordinates.
- Location * getTopLeft ()

  Get the topLeft location.
- void printMap ()

  Prints to console a representation of the map. Each location will have a colour representing a country that owns said location.
- MapState * getState ()

  Get the State object of the Map.

5.15.1 Constructor & Destructor Documentation

5.15.1.1 Map() [1/3]

```
Map::Map ( )
```

Construct a new Map object initializing all locations in the Map.

5.15.1.2 Map() [2/3]

```
Map::Map (
            Location * _cloneTopLeft )
```

Construct a new Map object and set topLeft to _cloneTopLeft.

Parameters

_cloneTopLeft	

5.15.1.3 ~Map()

```
Map::~Map ( )
```

Call delete on each location in the Map using an iterator.

5.15.1.4 Map() [3/3]

```
Map::Map (
            Map * _oldMap )
```

Construct a copy of the passed in map pointer.

Parameters

map	: Map* - the pointer to be copied into the new map.

5.15.2 Member Function Documentation

5.15.2.1 getLocation()

```
Location * Map::getLocation (
            int _x,
            int _y )
```

Get the Location object with matching x and y coordinates.

Exceptions : std::__throw_out_of_range if _x>24 or _x<0 or _y>26 or _y<0.

Parameters

↩ _↩ x	: int - The x coordinate of the location to be returned.
↩ _↩ y	: int - The y coordinate of the location to be returned.

Returns

Location∗

5.15.2.2 getState()

```
MapState * Map::getState ( )
```

Get the State object of the Map.

Returns

MapState∗

5.15.2.3 getTopLeft()

```
Location * Map::getTopLeft ( )
```

Get the topLeft location.

Returns

Location∗

5.15.2.4 printMap()

```
void Map::printMap ( )
```

Prints to console a representation of the map. Each location will have a colour representing a country that owns said location.

The documentation for this class was generated from the following files:

- Map.h
- Map.cpp

5.16 MapState Class Reference

```
#include <MapState.h>
```

Public Member Functions

- MapState (Map *_m)

  Construct a new Map State object from a passed in Map pointer. Calls copy contructor of the Map class passing in m as parameter and sets mapState equal to the result.
- ~MapState ()

  Destructor for the MapState class. Deletes tha map state object held by the class.
- Map * clone ()

  Return a clone of the mapState object.

5.16.1 Detailed Description

Author

Julian Pienaar

5.16.2 Constructor & Destructor Documentation

5.16.2.1 MapState()

```
MapState::MapState (
            Map * _m )
```

Construct a new Map State object from a passed in Map pointer. Calls copy contructor of the Map class passing in m as parameter and sets mapState equal to the result.

Parameters

↩ _↩ *m*	: Map∗ - Pointer to the Map object to be made into a state.

5.16.2.2 ∼MapState()

```
MapState::~MapState ( )
```

Destructor for the MapState class. Deletes tha map state object held by the class.

5.16.3 Member Function Documentation

5.16.3.1 clone()

```
Map * MapState::clone ( )
```

Return a clone of the mapState object.

Returns

Map∗

The documentation for this class was generated from the following files:

- MapState.h
- MapState.cpp

5.17 Memento Class Reference

Public Member Functions

- Memento ()

  Construct a new Memento object.
- Memento (SimulationState ∗_simulationState)

  Construct a new Memento object and save the passed in state.
- ∼Memento ()

  Destroy the Memento object and delete its stored SimulationState.
- SimulationState ∗ getState ()

  Get the SimulationState object stored by the Memento.
- void setState (SimulationState ∗_simulationState)

  Set the SimulationState object stored by the Memento.

5.17.1 Constructor & Destructor Documentation

5.17.1.1 Memento() [1/2]

```
Memento::Memento ( )
```

Construct a new Memento object.

5.17.1.2 Memento() [2/2]

```
Memento::Memento (
            SimulationState * _simulationState )
```

Construct a new Memento object and save the passed in state.

Parameters

_state	: SimulationState∗ - The state to save

5.17.1.3 ∼Memento()

```
Memento::~Memento ( )
```

Destroy the Memento object and delete its stored SimulationState.

5.17.2 Member Function Documentation

5.17.2.1 getState()

```
SimulationState * Memento::getState ( )
```

Get the SimulationState object stored by the Memento.

Exceptions : std::out_of_range if the Memento does not hold a SimulationState

Returns

SimulationState∗

5.17.2.2 setState()

```
void Memento::setState (
            SimulationState * _simulationState )
```

Set the SimulationState object stored by the Memento.

Parameters

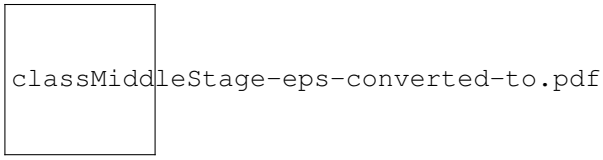
_state	: SimulationState∗ - the state to store

The documentation for this class was generated from the following files:

- Memento.h
- Memento.cpp

5.18 MiddleStage Class Reference

Inheritance diagram for MiddleStage:



Public Member Functions

- int getWarStage ()

  returns warstage via int = 1
- MiddleStage ()

  Construct a new Middle Stage object.
- ∼MiddleStage ()

  Destroy the Middle Stage object.
- MiddleStage ∗ clone ()

  Return a deep copy of the Middle Stage object.

5.18.1 Constructor & Destructor Documentation

5.18.1.1 MiddleStage()

```
MiddleStage::MiddleStage ( )
```

Construct a new Middle Stage object.

5.18.1.2 ∼MiddleStage()

```
MiddleStage::∼MiddleStage ( )
```

Destroy the Middle Stage object.

5.18.2 Member Function Documentation

5.18.2.1 clone()

```
MiddleStage * MiddleStage::clone ( )  [virtual]
```

Return a deep copy of the Middle Stage object.

Returns

MiddleStage*

Implements WarStage.

5.18.2.2 getWarStage()

```
int MiddleStage::getWarStage ( )  [virtual]
```

returns warstage via int = 1

Implements WarStage.

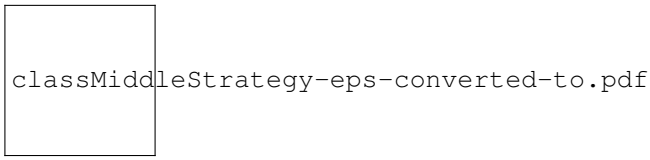
The documentation for this class was generated from the following files:

- MiddleStage.h
- MiddleStage.cpp

5.19 MiddleStrategy Class Reference

```
#include <MiddleStrategy.h>
```

Inheritance diagram for MiddleStrategy:



Public Member Functions

- **MiddleStrategy** ()

  Constructor.
- ~**MiddleStrategy** ()

  destructor

Protected Member Functions

- void defensiveMove (Country *countryA, Country *countryB)

  virtual function representing the implementation of a turn when Country A is weaker than Country B
- void neutralMove (Country *countryA, Country *countryB)

  virtual function representing the implementation of a turn when Country A and Country B are similar in strength
- void offensiveMove (Country *countryA, Country *countryB)

  virtual function representing the implementation of a turn when Country A is stronger than Country B

5.19.1 Detailed Description

Author

Mekhail Muller

5.19.2 Member Function Documentation

5.19.2.1 defensiveMove()

```
void MiddleStrategy::defensiveMove (
            Country * countryA,
            Country * countryB )  [protected], [virtual]
```

virtual function representing the implementation of a turn when Country A is weaker than Country B

Parameters

countryA	the country that is making the move (calling country)
countryB	the country being attacked by calling country

Implements Strategy.

5.19.2.2 neutralMove()

```
void MiddleStrategy::neutralMove (
            Country * countryA,
            Country * countryB )  [protected], [virtual]
```

virtual function representing the implementation of a turn when Country A and Country B are similar in strength

Parameters

countryA	the country that is making the move (calling country)
countryB	the country being attacked by calling country

Implements Strategy.

5.19.2.3 offensiveMove()

```
void MiddleStrategy::offensiveMove (
            Country * countryA,
            Country * countryB )  [protected], [virtual]
```

virtual function representing the implementation of a turn when Country A is stronger than Country B

Parameters

countryA	the country that is making the move (calling country)
countryB	the country being attacked by calling country

Implements Strategy.

The documentation for this class was generated from the following files:

- MiddleStrategy.h
- MiddleStrategy.cpp

5.20 Military Class Reference

Public Member Functions

- Military ()

  Construct a new Military object.
- Military (Military *_military)

  Construct a new Military object.
- ~Military ()

  Destroy the Military object.
- void **attack** (Country *_country)

5.20.1 Constructor & Destructor Documentation

5.20.1.1 Military() [1/2]

```
Military::Military ( )
```

Construct a new Military object.

5.20.1.2 Military() [2/2]

```
Military::Military (
            Military * _military )
```

Construct a new Military object.

5.20.1.3 ~Military()

```
Military::~Military ( )
```

Destroy the Military object.

The documentation for this class was generated from the following files:

- Military.h
- Military.cpp

5.21 MilitaryState Class Reference

Public Member Functions

- **MilitaryState** ()

  Default constructor.
- ~**MilitaryState** ()

  Default destructor.
- void **setShips** (std::vector< Ship * > *_ships)

  setter for the ships in this military
- void **setPlanes** (std::vector< Plane * > *_planes)

  setter for the planes in this military
- void **setTanks** (std::vector< Tank * > *_tanks)

  setter for the tanks in this military
- void **setBattalions** (std::vector< Battalion * > *_battalions)

  Setter for the battalions.
- void setTroops (int _troops)

  setter for the number of troops
- void **setVehicleFactories** (std::vector< VehicleFactory * > *_vehicleFactories)

  setter for the VehicleFactories object
- int getNumTroops ()

  getter number of troops (integer)
- void **updateNumTroops** (int _numTroops, bool isAddition)

  updates the number of troops in this military
- int getNumTanks ()

  getter for the number of tanks in this military
- void updateNumTanks (int _numTanks, bool isAddition)

  updates the number of tanks through addition or subtraction
- int getNumShips ()

getter for the number of ships in this military

- void updateNumShips (int _numShips, bool isAddition)

  updates the number of ships through addition or subtraction
- int getNumPlanes ()

  getter for the number of planes in this military
- void **updateNumPlanes** (int _numPlanes, bool isAddition)

  Get the number of Vehicle Factories object.
- int **getNumBattalions** ()

  Get the number of battalion objects in this military.
- void updateNumBattalions (int _numBattalions, bool isAddition)

  updates the number of battalions through addition or subtraction
- MilitaryState * clone ()

  Create a deep copy of this object.

5.21.1 Member Function Documentation

5.21.1.1 clone()

```
MilitaryState * MilitaryState::clone ( )
```

Create a deep copy of this object.

Returns

MilitaryState*

5.21.1.2 getNumPlanes()

```
int MilitaryState::getNumPlanes ( )
```

getter for the number of planes in this military

Returns

number of planes in this military (integer)

5.21.1.3 getNumShips()

```
int MilitaryState::getNumShips ( )
```

getter for the number of ships in this military

Returns

the number of ships in this military (integer)

5.21.1.4 getNumTanks()

```
int MilitaryState::getNumTanks ( )
```

getter for the number of tanks in this military

Returns

the number of tanks in this military

5.21.1.5 getNumTroops()

```
int MilitaryState::getNumTroops ( )
```

getter number of troops (integer)

Returns

number of troops in this military (integer)

5.21.1.6 setTroops()

```
void MilitaryState::setTroops (
            int _troops )
```

setter for the number of troops

5.21.1.7 updateNumBattalions()

```
void MilitaryState::updateNumBattalions (
            int _numBattalions,
            bool isAddition )
```

updates the number of battalions through addition or subtraction

Parameters

_numBattalions	the number of battalions to add or subtract
isAddition	true if the number of battalions is to be added, false if it is to be subtracted

5.21.1.8 updateNumShips()

```
void MilitaryState::updateNumShips (
            int _numShips,
            bool isAddition )
```

updates the number of ships through addition or subtraction

Parameters

_numShips	the number of ships to add or subtract
isAddition	true if the number of ships is to be added, false if it is to be subtracted

5.21.1.9 updateNumTanks()

```
void MilitaryState::updateNumTanks (
            int _numTanks,
            bool isAddition )
```

updates the number of tanks through addition or subtraction

Parameters

_numTanks	the number of tanks to add or subtract
isAddition	true if the number of tanks is to be added, false if it is to be subtracted

The documentation for this class was generated from the following files:

- MilitaryState.h
- MilitaryState.cpp

5.22 Neighbour Class Reference

```
#include <Neighbour.h>
```

Inheritance diagram for Neighbour:

Public Member Functions

- Neighbour (Location *_neighbour)

  Constructor for Neighbour class.
- virtual ~Neighbour ()

  Destructor for the Neighbour class.
- virtual void add (Location *_neighbour)

  If location is NULL set location to equal _neighbour else call add on location sending _neighbour in as the parameter.

Protected Attributes

- Location * **neighbour**

5.22.1 Detailed Description

Author

Julian Pienaar

5.22.2 Constructor & Destructor Documentation

5.22.2.1 Neighbour()

```
Neighbour::Neighbour (
            Location * _neighbour )
```

Constructor for Neighbour class.

Parameters

_neighbour	: Location* - Pointer to the added neighbour location.

5.22.2.2 ~Neighbour()

```
Neighbour::~Neighbour ( )  [virtual]
```

Destructor for the Neighbour class.

5.22.3 Member Function Documentation

5.22.3.1 add()

```
void Neighbour::add (
            Location * _neighbour )  [virtual]
```

If location is NULL set location to equal _neighbour else call add on location sending _neighbour in as the parameter.

Parameters

_neighbour	: Location* - Pointer to the location to be added.

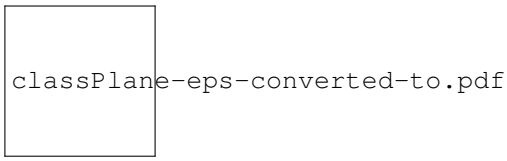
Implements Location.

The documentation for this class was generated from the following files:

- Neighbour.h
- Neighbour.cpp

5.23 Plane Class Reference

Inheritance diagram for Plane:



Public Member Functions

- Plane ()

  constructs the Plane Object
- ~**Plane** ()

  destroys the Plane Object

5.23.1 Constructor & Destructor Documentation

5.23.1.1 Plane()

```
Plane::Plane ( )
```

constructs the Plane Object

Author

Jake Mahloko

The documentation for this class was generated from the following files:

- Plane.h
- Plane.cpp

5.24 PlaneFactory Class Reference

```
#include <PlaneFactory.h>
```

Inheritance diagram for PlaneFactory:

Public Member Functions

- **PlaneFactory** ()

  construct a new PlaneFactory object
- ∼**PlaneFactory** ()

  destroy a PlaneFactory object
- Vehicle ∗ manufactureVehicle ()

  creates Plane object which inherits from Vehicle
- PlaneFactory ∗ clone ()

  Create a deep copy of the current object.

5.24.1 Detailed Description

Author

Jake Mahloko brief PlaneFactory Inherits from the VehicleFactory class which it uses the factory method to create plane objects and return pointer to the object

5.24.2 Member Function Documentation

5.24.2.1 clone()

```
PlaneFactory * PlaneFactory::clone ( )  [virtual]
```

Create a deep copy of the current object.

Returns

PlaneFactory∗

Implements VehicleFactory.

5.24.2.2 manufactureVehicle()

```
Vehicle * PlaneFactory::manufactureVehicle ( )  [virtual]
```

creates Plane object which inherits from Vehicle

Returns

Vehicle∗ but this class will send a Plane object

Implements VehicleFactory.

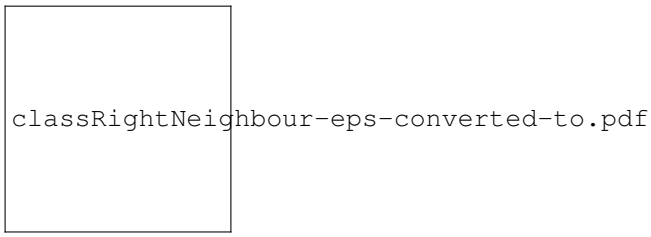
The documentation for this class was generated from the following files:

- PlaneFactory.h
- PlaneFactory.cpp

5.25 RightNeighbour Class Reference

```
#include <RightNeighbour.h>
```

Inheritance diagram for RightNeighbour:



Public Member Functions

- RightNeighbour (Location ∗_neighbour)

  The constructor for right neighbour.
- Location ∗ getRight ()

  Returns right neighbour.
- bool hasRight ()

  Always returns true since RightNeighbour will always have a right neighbour.

Additional Inherited Members

5.25.1 Detailed Description

Author

Julian Pienaar

5.25.2 Constructor & Destructor Documentation

5.25.2.1 RightNeighbour()

```
RightNeighbour::RightNeighbour (
            Location * _neighbour )
```

The constructor for right neighbour.

Parameters

_neighbour	: Location∗ - Pointer to the right neighbour.

5.25.3 Member Function Documentation

5.25.3.1 getRight()

```
Location * RightNeighbour::getRight ( ) [virtual]
```

Returns right neighbour.

Returns

Location∗

Reimplemented from Location.

5.25.3.2 hasRight()

```
bool RightNeighbour::hasRight ( ) [virtual]
```

Always returns true since RightNeighbour will always have a right neighbour.

Returns

true

Reimplemented from Location.

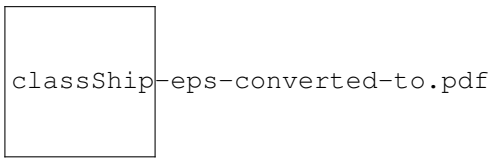
The documentation for this class was generated from the following files:

- RightNeighbour.h
- RightNeighbour.cpp

5.26 Ship Class Reference

```
#include <Ship.h>
```

Inheritance diagram for Ship:



Public Member Functions

- Ship ()

  constructs the Ship Object
- ~**Ship** ()

  destroys the Ship Object

5.26.1 Detailed Description

Author

Jacob Mahloko

5.26.2 Constructor & Destructor Documentation

5.26.2.1 Ship()

```
Ship::Ship ( )
```

constructs the Ship Object

Author

Jake Mahloko

The documentation for this class was generated from the following files:

- Ship.h
- Ship.cpp

5.27 ShipFactory Class Reference

```
#include <ShipFactory.h>
```

Inheritance diagram for ShipFactory:

classShipFactory-eps-converted-to.pdf

Public Member Functions

- **ShipFactory** ()

  construct a new ShipFactory object
- ~**ShipFactory** ()

  destroy a ShipFactory object
- Vehicle * manufactureVehicle ()

  creates Ship object which inherits from Vehicle
- ShipFactory * clone ()

  Create a deep copy of the current object.

5.27.1 Detailed Description

Author

Jacob Mahloko brief ShipFactory Inherits from the VehicleFactory class which it uses the factory method to create Ship objects and return pointer to the object

5.27.2 Member Function Documentation

5.27.2.1 clone()

```
ShipFactory * ShipFactory::clone ( )  [virtual]
```

Create a deep copy of the current object.

Returns

ShipFactory*

Implements VehicleFactory.

5.27.2.2 manufactureVehicle()

```
Vehicle * ShipFactory::manufactureVehicle ( )  [virtual]
```

creates Ship object which inherits from Vehicle

Returns

Vehicle* but this class will send a Ship object

Implements VehicleFactory.

The documentation for this class was generated from the following files:

- ShipFactory.h
- ShipFactory.cpp

5.28 SimulationManager Class Reference

Public Member Functions

- SimulationManager ()

  Construct a new Simulation Manager object.
- virtual ~SimulationManager ()

  Destroy the Simulation Manager object and free all memory.
- void runSimulation ()

  Run the simulation from an initial state to completion.

Protected Member Functions

- bool restoreState ()

  Restore the state of the simulation to the last saved state.
- void saveState ()

  Save the current state of the simulation in the backup.
- void resetSimulation ()

  Setup the simulation to its initial state.
- bool isSimulationRunning ()

  Check whether the simulation has completed or not.
- void takeTurn ()

  Take a turn in the simulation by having each country take a turn.
- void viewSummary ()

  Provide the user with a summary of the simulation at this point in time.
- void processMenu ()

  Display the menu of options available to the user and perform that action.
- void viewCountrySummary ()

  Display a detailed summary of a country within the simulation.
- void designModeAction ()

  Change the state of the simulation in design mode.
- void finalMessage ()

  Display the last message of the simulation providing a summary of the simulation as a whole after completion.

5.28.1 Constructor & Destructor Documentation

5.28.1.1 SimulationManager()

```
SimulationManager::SimulationManager ( )
```

Construct a new Simulation Manager object.

5.28.1.2 ∼SimulationManager()

```
SimulationManager::∼SimulationManager ( ) [virtual]
```

Destroy the Simulation Manager object and free all memory.

5.28.2 Member Function Documentation

5.28.2.1 designModeAction()

```
void SimulationManager::designModeAction ( ) [protected]
```

Change the state of the simulation in design mode.

5.28.2.2 finalMessage()

```
void SimulationManager::finalMessage ( ) [protected]
```

Display the last message of the simulation providing a summary of the simulation as a whole after completion.

5.28.2.3 isSimulationRunning()

```
bool SimulationManager::isSimulationRunning ( ) [protected]
```

Check whether the simulation has completed or not.

Returns

true if the simulation has not completed

false if the simulation has completed

5.28.2.4 processMenu()

```
void SimulationManager::processMenu ( ) [protected]
```

Display the menu of options available to the user and perform that action.

In design mode will allow the user to change the state of the simulation

5.28.2.5 resetSimulation()

```
void SimulationManager::resetSimulation ( ) [protected]
```

Setup the simulation to its initial state.

5.28.2.6 restoreState()

```
bool SimulationManager::restoreState ( ) [protected]
```

Restore the state of the simulation to the last saved state.

Returns

true if the state was restored

false if the state was not restored

5.28.2.7 runSimulation()

```
void SimulationManager::runSimulation ( )
```

Run the simulation from an initial state to completion.

5.28.2.8 saveState()

```
void SimulationManager::saveState ( ) [protected]
```

Save the current state of the simulation in the backup.

5.28.2.9 takeTurn()

```
void SimulationManager::takeTurn ( ) [protected]
```

Take a turn in the simulation by having each country take a turn.

Increments the turn count and facilitates the switching of war stages. Also saves the state of the system before the turn is taken

5.28.2.10 viewCountrySummary()

```
void SimulationManager::viewCountrySummary ( ) [protected]
```

Display a detailed summary of a country within the simulation.

5.28.2.11 viewSummary()

```
void SimulationManager::viewSummary ( ) [protected]
```

Provide the user with a summary of the simulation at this point in time.

Also view the menu of next options available to the user

The documentation for this class was generated from the following files:

- SimulationManager.h
- SimulationManager.cpp

5.29 SimulationState Class Reference

Public Member Functions

- SimulationState ()

  Construct a new Simulation State object.
- ~SimulationState ()

  Destroy the Simulation State object and delete all held state objects.
- void setMapState (MapState *_mapState)

  Set the Map State object stored by the SimulationState.
- void setStageContextState (StageContextState *_stageContextState)

  Set the Stage Context State object stored by the SimulationState.
- void addSuperpowerState (SuperpowerState *_superpowerState)

  Add a SuperpowerState object to the SimulationState.
- int getSuperpowerStateCount ()

  Get the number of SuperpowerState objects stored by the SimulationState.
- SuperpowerState * getSuperpowerState (int _index)

  Get the SuperpowerState object stored by the SimulationState.
- MapState * getMapState ()

  Get the Map State object stored by the SimulationState.
- StageContextState * getStageContextState ()

  Get the StageContextState object stored by the SimulationState.
- std::time_t getTimestamp ()

  Get the Timestamp object stored by the SimulationState.

5.29.1 Constructor & Destructor Documentation

5.29.1.1 SimulationState()

```
SimulationState::SimulationState ( )
```

Construct a new Simulation State object.

Sets the timestamp to the current time

5.29.1.2 ~SimulationState()

```
SimulationState::~SimulationState ( )
```

Destroy the Simulation State object and delete all held state objects.

5.29.2 Member Function Documentation

5.29.2.1 addSuperpowerState()

```
void SimulationState::addSuperpowerState (
            SuperpowerState * _superpowerState )
```

Add a SuperpowerState object to the SimulationState.

Parameters

_superpowerState	: SuperpowerState* - Pointer to the SuperpowerState object.

5.29.2.2 getMapState()

```
MapState * SimulationState::getMapState ( )
```

Get the Map State object stored by the SimulationState.

Exceptions : std::out_of_range if the SimulationState does not hold a MapState

Returns

MapState*

5.29.2.3 getStageContextState()

```
StageContextState * SimulationState::getStageContextState ( )
```

Get the StageContextState object stored by the SimulationState.

Exceptions : std::out_of_range if the SimulationState does not hold a StageContextState

Returns

StageContextState*

5.29.2.4 getSuperpowerState()

```
SuperpowerState * SimulationState::getSuperpowerState (
            int _index )
```

Get the SuperpowerState object stored by the SimulationState.

Exceptions : std::out_of_range if the index is out of range

Parameters

index	: int - Index of the SuperpowerState object to return.

Returns

SuperpowerState*

5.29.2.5 getSuperpowerStateCount()

```
int SimulationState::getSuperpowerStateCount ( )
```

Get the number of SuperpowerState objects stored by the SimulationState.

Returns

int

5.29.2.6 getTimestamp()

```
time_t SimulationState::getTimestamp ( )
```

Get the Timestamp object stored by the SimulationState.

Returns

std::time_t

5.29.2.7 setMapState()

```
void SimulationState::setMapState (
            MapState * _mapState )
```

Set the Map State object stored by the SimulationState.

If a MapState is already stored, it is deleted

Parameters

_mapState	: MapState∗ - Pointer to the MapState object.

5.29.2.8 setStageContextState()

```
void SimulationState::setStageContextState (
            StageContextState * _stageContextState )
```

Set the Stage Context State object stored by the SimulationState.

Parameters

_stageContextState	: StageContextState∗ - Pointer to the StageContextState object.

The documentation for this class was generated from the following files:

- SimulationState.h
- SimulationState.cpp

5.30 StageContext Class Reference

Public Member Functions

- StageContextState ∗ **getState** ()

Creates a copy of the singleton class and returns for storage purposes.

- int **getCurrentRound** ()

  returns the current round we are on
- int **getWarStage** ()

  handle() function: Calculates if currentRound is within its current warstage, and returns warstage int
- void **incrementRound** ()

  increments round/turn to go to the next round
- void setSimulationLength (int _length)

  sets the simulationLength: how many turns/rounds do we want to run the simulation for
- ~StageContext ()

  Destroy the Stage Context object.
- void setCurrentRound (int _round)

  Set the Current Round object.
- void setCurrentStage (WarStage *_stage)

  Set the Current Stage object.
- void setState (StageContextState *_state)

  Set the StageContext equal to the StageContextState.
- int getSimulationLength ()

  Get the Simulation Length.
- void moveToStage (int _stage)

  Move to the war stage specified by the parameter.

Static Public Member Functions

- static StageContext * **getInstance** ()

  initialises our singleton object and returns it

Protected Member Functions

- **StageContext** (int _simulationLength)

Protected Attributes

- int **simulationLength**
- int **currentRound**
- WarStage * **currentStage**

Static Protected Attributes

- static StageContext * onlyInstance = NULL

5.30.1 Constructor & Destructor Documentation

5.30.1.1 ~StageContext()

```
StageContext::~StageContext ( )
```

Destroy the Stage Context object.

5.30.2 Member Function Documentation

5.30.2.1 getSimulationLength()

```
int StageContext::getSimulationLength ( )
```

Get the Simulation Length.

Returns

int

5.30.2.2 moveToStage()

```
void StageContext::moveToStage (
            int _stage )
```

Move to the war stage specified by the parameter.

Parameters

_stage	: int - the stage we want to move to

5.30.2.3 setCurrentRound()

```
void StageContext::setCurrentRound (
            int _round )
```

Set the Current Round object.

Parameters

_round	: int - the round we want to set the current round to

5.30.2.4 setCurrentStage()

```
void StageContext::setCurrentStage (
            WarStage * _stage )
```

Set the Current Stage object.

Parameters

_stage	: WarStage* - the stage we want to set the current stage to

5.30.2.5 setSimulationLength()

```
void StageContext::setSimulationLength (
            int _length )
```

sets the simulationLength: how many turns/rounds do we want to run the simulation for

Parameters

length,the	length wanted for the simulation

5.30.2.6 setState()

```
void StageContext::setState (
            StageContextState * _state )
```

Set the StageContext equal to the StageContextState.

Parameters

_state	: StageContextState* - the state we want to set the StageContext to

5.30.3 Member Data Documentation

5.30.3.1 onlyInstance

```
StageContext * StageContext::onlyInstance = NULL  [static], [protected]
```

Author

Mekhail Muller

The documentation for this class was generated from the following files:

- StageContext.h
- StageContext.cpp

5.31 StageContextState Class Reference

Public Member Functions

- StageContextState ()

  Construct a new StageContext State object.
- ~StageContextState ()

  Destroy the StageContext State object.
- void setSimulationLength (int _length)

  Set the Simulation Length.
- int getSimulationLength ()

  Get the simulation length.
- void setCurrentRound (int _round)

  Set the Current Round.
- int **getCurrentRound** ()

  returns the current round we are on
- void setCurrentStage (WarStage *_stage)

  Set the Current Stage.
- WarStage * getCurrentStage ()

  Get the Current Stage.

5.31.1 Constructor & Destructor Documentation

5.31.1.1 StageContextState()

```
StageContextState::StageContextState ( )
```

Construct a new StageContext State object.

5.31.1.2 ~StageContextState()

```
StageContextState::~StageContextState ( )
```

Destroy the StageContext State object.

5.31.2 Member Function Documentation

5.31.2.1 getCurrentStage()

```
WarStage * StageContextState::getCurrentStage ( )
```

Get the Current Stage.

Returns

WarStage*

5.31.2.2 getSimulationLength()

```
int StageContextState::getSimulationLength ( )
```

Get the simulation length.

Returns

int : the length of the simulation

5.31.2.3 setCurrentRound()

```
void StageContextState::setCurrentRound (
            int _round )
```

Set the Current Round.

Parameters

_round	: int - the round we want to set the current round to

5.31.2.4 setCurrentStage()

```
void StageContextState::setCurrentStage (
            WarStage * _stage )
```

Set the Current Stage.

Parameters

_stage	: WarStage∗ - the stage we want to set the current stage to

5.31.2.5 setSimulationLength()

```
void StageContextState::setSimulationLength (
            int _length )
```

Set the Simulation Length.

Parameters

_length	: int - the length of the simulation

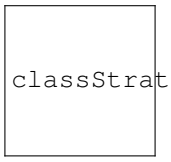
The documentation for this class was generated from the following files:

- StageContextState.h
- StageContextState.cpp

5.32 Strategy Class Reference

```
#include <Strategy.h>
```

Inheritance diagram for Strategy:



Public Member Functions

- Strategy ()

  Constructor: initialises Country variable.
- virtual void takeTurn (double ∗strengthRatings, Country ∗countryA, Country ∗countryB)

  virtual function representing the implementation of a strategy.

Protected Member Functions

- virtual void offensiveMove (Country ∗countryA, Country ∗countryB)=0

  virtual function representing the implementation of a turn when Country A is stronger than Country B
- virtual void neutralMove (Country ∗countryA, Country ∗countryB)=0

  virtual function representing the implementation of a turn when Country A has equal strength with Country B
- virtual void defensiveMove (Country ∗countryA, Country ∗countryB)=0

  virtual function representing the implementation of a turn when Country A is weaker than Country B

5.32.1 Detailed Description

This class determines which strategy to choose for a country.

5.32.2 Constructor & Destructor Documentation

5.32.2.1 Strategy()

```
Strategy::Strategy ( )
```

Constructor: initialises Country variable.

5.32.3 Member Function Documentation

5.32.3.1 defensiveMove()

```
virtual void Strategy::defensiveMove (
            Country * countryA,
            Country * countryB )  [protected], [pure virtual]
```

virtual function representing the implementation of a turn when Country A is weaker than Country B

Parameters

countryA	the country that is making the move (calling country)
countryB	the country being attacked by calling country

Implemented in EarlyStrategy, LateStrategy, and MiddleStrategy.

5.32.3.2 neutralMove()

```
virtual void Strategy::neutralMove (
            Country * countryA,
            Country * countryB )  [protected], [pure virtual]
```

virtual function representing the implementation of a turn when Country A has equal strength with Country B

Parameters

countryA	the country that is making the move (calling country)
countryB	the country being attacked by calling country

Implemented in EarlyStrategy, LateStrategy, and MiddleStrategy.

5.32.3.3 offensiveMove()

```
virtual void Strategy::offensiveMove (
            Country * countryA,
            Country * countryB )  [protected], [pure virtual]
```

virtual function representing the implementation of a turn when Country A is stronger than Country B

Parameters

countryA	the country that is making the move (calling country)
countryB	the country being attacked by calling country

Implemented in EarlyStrategy, LateStrategy, and MiddleStrategy.

5.32.3.4 takeTurn()

```
void Strategy::takeTurn (
            double * strengthRatings,
            Country * countryA,
            Country * countryB )  [virtual]
```

virtual function representing the implementation of a strategy.

Parameters

strengthRatings	double array holding the strength values of the 2 countries
countryA	the country that is making the move (calling country)
countryB	the country being attacked by calling country

The documentation for this class was generated from the following files:

- Strategy.h
- Strategy.cpp

5.33 Superpower Class Reference

Public Member Functions

- Superpower (std::string _name)

  Construct a new Superpower object.

- Superpower (SuperpowerState *_state)

  Reconstruct a superpower object from a SuperpowerState.
- ~Superpower ()

  Destroy the Superpower object and delete all of its countries.
- std::string getName ()

  Get the name of the superpower.
- void addCountry (Country *_country)

  Add a country to the superpower.
- int getCountryCount ()

  Get the number of countries owned by the superpower.
- Country * getCountry (int _index)

  Get the country at the specified index.
- void removeCountry (Country *_country)

  Remove the passed in country from the superpower.
- SuperpowerState * getState ()

  Get the state of the superpower.
- void printSummary ()

  Prints a summary of the superpower and all its countries.
- void resetLocations (Map *_map)

  Link the countries owned by the superpower to the passed in map after the simulation has been restored.
- void resetEnemies (std::vector< Country * > *_enemies)

  Link the countries owned by the superpower's enemies to the passed in vector after the simulation has been restored.
- std::vector< Country * > * getAllCountries ()

  Get the vector of countries owned by the superpower.

5.33.1 Constructor & Destructor Documentation

5.33.1.1 Superpower() [1/2]

```
Superpower::Superpower (
            std::string _name )
```

Construct a new Superpower object.

Parameters

name	: std::string - the name of the superpower

5.33.1.2 Superpower() [2/2]

```
Superpower::Superpower (
            SuperpowerState * _state )
```

Reconstruct a superpower object from a SuperpowerState.

Parameters

state	: SuperpowerState∗ - the state to reconstruct from

5.33.1.3 ∼Superpower()

```
Superpower::~Superpower ( )
```

Destroy the Superpower object and delete all of its countries.

5.33.2 Member Function Documentation

5.33.2.1 addCountry()

```
void Superpower::addCountry (
            Country * _country )
```

Add a country to the superpower.

Parameters

country	: Country∗ - the country to add

5.33.2.2 getAllCountries()

```
std::vector< Country * > * Superpower::getAllCountries ( )
```

Get the vector of countries owned by the superpower.

Returns

std::vector<Country ∗>∗

5.33.2.3 getCountry()

```
Country * Superpower::getCountry (
            int _index )
```

Get the country at the specified index.

Exceptions : std::out_of_range if the index is out of range

Parameters

index	: int - the index of the country to get

Returns

Country∗ - pointer to the country at the specified index

5.33.2.4 getCountryCount()

```
int Superpower::getCountryCount ( )
```

Get the number of countries owned by the superpower.

Returns

int

5.33.2.5 getName()

```
string Superpower::getName ( )
```

Get the name of the superpower.

Returns

std::string

5.33.2.6 getState()

```
SuperpowerState * Superpower::getState ( )
```

Get the state of the superpower.

Returns

SuperpowerState∗ - pointer to the state of the superpower

5.33.2.7 printSummary()

```
void Superpower::printSummary ( )
```

Prints a summary of the superpower and all its countries.

5.33.2.8 removeCountry()

```
void Superpower::removeCountry (
            Country * _country )
```

Remove the passed in country from the superpower.

Exceptions : std::out_of_range if the country is not owned by the superpower

Parameters

country	: Country∗ - the country to remove

5.33.2.9 resetEnemies()

```
void Superpower::resetEnemies (
            std::vector< Country * > * _enemies )
```

Link the countries owned by the superpower's enemies to the passed in vector after the simulation has been restored.

Parameters

_enemies	: std::vector<Superpower∗> - vector of pointers to the superpowers enemies

5.33.2.10 resetLocations()

```
void Superpower::resetLocations (
            Map * _map )
```

Link the countries owned by the superpower to the passed in map after the simulation has been restored.

Parameters

_map	: Map∗ - pointer to the current map

The documentation for this class was generated from the following files:

- Superpower.h
- Superpower.cpp

5.34 SuperpowerState Class Reference

Public Member Functions

- SuperpowerState (std::string _name)

  Construct a new Superpower State object.
- ∼SuperpowerState ()

  Destroy the Superpower State object.
- void addCountryState (CountryState ∗_countryState)

  Add a CountryState object to the SuperpowerState.

- std::string getName ()

  Get the Name object.
- int getCountryStateCount ()

  Get the number of CountryState objects stored by the SuperpowerState.
- CountryState ∗ getCountryState (int _index)

  Get the Country States object.

Protected Attributes

- std::string **name**
- std::vector< CountryState ∗ > ∗ **countryStates**

5.34.1 Constructor & Destructor Documentation

5.34.1.1 SuperpowerState()

```
SuperpowerState::SuperpowerState (
            std::string _name )
```

Construct a new Superpower State object.

Parameters

_name	: std::string - The name of the Superpower.

5.34.1.2 ∼SuperpowerState()

```
SuperpowerState::~SuperpowerState ( )
```

Destroy the Superpower State object.

5.34.2 Member Function Documentation

5.34.2.1 addCountryState()

```
void SuperpowerState::addCountryState (
            CountryState * _countryState )
```

Add a CountryState object to the SuperpowerState.

Parameters

_countryState	: CountryState∗ - Pointer to the CountryState object.

5.34.2.2 getCountryState()

```
CountryState * SuperpowerState::getCountryState (
            int _index )
```

Get the Country States object.

Exceptions : std::__throw_out_of_range if _index is out of range.

Returns

CountryState∗

5.34.2.3 getCountryStateCount()

```
int SuperpowerState::getCountryStateCount ( )
```

Get the number of CountryState objects stored by the SuperpowerState.

Returns

int

5.34.2.4 getName()

```
string SuperpowerState::getName ( )
```

Get the Name object.

Returns

std::string

The documentation for this class was generated from the following files:

- SuperpowerState.h
- SuperpowerState.cpp

5.35 Tank Class Reference

```
#include <Tank.h>
```

Inheritance diagram for Tank:

Public Member Functions

- Tank ()

  constructor for the tank object
- ~**Tank** ()

  destructor for the tank object

5.35.1 Detailed Description

Author

Jake Mahloko

5.35.2 Constructor & Destructor Documentation

5.35.2.1 Tank()

```
Tank::Tank ( )
```

constructor for the tank object

Author

Jake Mahloko

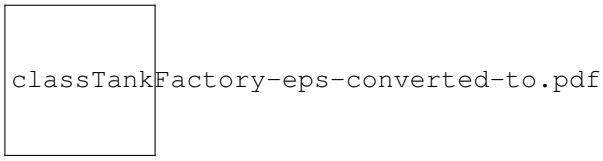
The documentation for this class was generated from the following files:

- Tank.h
- Tank.cpp

5.36 TankFactory Class Reference

```
#include <TankFactory.h>
```

Inheritance diagram for TankFactory:



Public Member Functions

- **TankFactory** ()

  construct a new TankFactory object
- ~**TankFactory** ()

  destroy a TankFactory object
- Vehicle * manufactureVehicle ()

  creates Tank object which inherits from Vehicle
- TankFactory * clone ()

  Create a deep copy of the current object.

5.36.1 Detailed Description

Author

Jacob Mahloko brief TankFactory Inherits from the VehicleFactory class which it uses the factory method to create Tank objects and return pointer to the object

5.36.2 Member Function Documentation

5.36.2.1 clone()

```
TankFactory * TankFactory::clone ( )  [virtual]
```

Create a deep copy of the current object.

Returns

TankFactory*

Implements VehicleFactory.

5.36.2.2 manufactureVehicle()

```
Vehicle * TankFactory::manufactureVehicle ( )  [virtual]
```

creates Tank object which inherits from Vehicle

Returns

Vehicle∗ but this class will send a Tank object

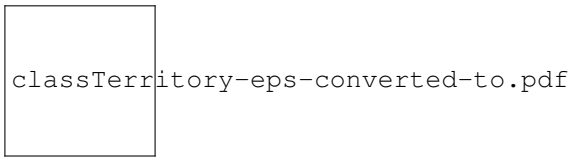
Implements VehicleFactory.

The documentation for this class was generated from the following files:

- TankFactory.h
- TankFactory.cpp

5.37 Territory Class Reference

Inheritance diagram for Territory:



Public Member Functions

- Territory (int _x, int _y, std::string _color="\x1B[44m")

  Construct a new Territory object.
- ∼Territory ()

  Destructor for the Territory class. Delete location if location is not NULL.
- void add (Location ∗_neighbour)

  If location is NULL set location to equal _neighbour else call add on location sending _neighbour in as the parameter.

Additional Inherited Members

5.37.1 Constructor & Destructor Documentation

5.37.1.1 Territory()

```
Territory::Territory (
            int _x,
            int _y,
            std::string _color = "\x1B[44m" )
```

Construct a new Territory object.

Parameters

_x	: int - x coordinate of the location.
_y	: int - y coordinate of the location.
_colour	: char - colour of the territory.

5.37.1.2 ~Territory()

```
Territory::~Territory ( )
```

Destructor for the Territory class. Delete location if location is not NULL.

5.37.2 Member Function Documentation

5.37.2.1 add()

```
void Territory::add (
            Location * _neighbour )  [virtual]
```

If location is NULL set location to equal _neighbour else call add on location sending _neighbour in as the parameter.

Parameters

_neighbour	: Location* - Pointer to the location to be added.

Implements Location.

The documentation for this class was generated from the following files:

- Territory.h
- Territory.cpp

5.38 TopNeighbour Class Reference

```
#include <TopNeighbour.h>
```

Inheritance diagram for TopNeighbour:

classTopNeighbour-eps-converted-to.pdf

Public Member Functions

- TopNeighbour (Location *_neighbour)

  The constructor for top neighbour.
- Location * getTop ()

  Returns top neighbour.
- bool hasTop ()

  Always returns true since TopNeighbour will always have a top neighbour.

Additional Inherited Members

5.38.1 Detailed Description

Author

Julian Pienaar

5.38.2 Constructor & Destructor Documentation

5.38.2.1 TopNeighbour()

```
TopNeighbour::TopNeighbour (
            Location * _neighbour )
```

The constructor for top neighbour.

Parameters

_neighbour	: Location* - Pointer to top neighbour.

5.38.3 Member Function Documentation

5.38.3.1 getTop()

```
Location * TopNeighbour::getTop ( )  [virtual]
```

Returns top neighbour.

Returns

Location*

Reimplemented from Location.

5.38.3.2 hasTop()

```
bool TopNeighbour::hasTop ( ) [virtual]
```

Always returns true since TopNeighbour will always have a top neighbour.

Returns

true

Reimplemented from Location.

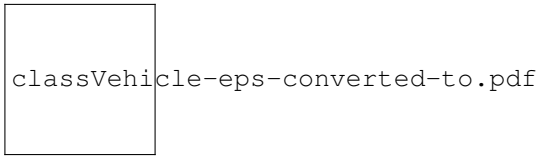
The documentation for this class was generated from the following files:

- TopNeighbour.h
- TopNeighbour.cpp

5.39 Vehicle Class Reference

```
#include <Vehicle.h>
```

Inheritance diagram for Vehicle:



Public Member Functions

- Vehicle ()

  constructs a Vehicle object
- virtual ~**Vehicle** ()

  destroys a Vehicle object

5.39.1 Detailed Description

Author

Jake Mahloko

5.39.2 Constructor & Destructor Documentation

5.39.2.1 Vehicle()

```
Vehicle::Vehicle ( )
```

constructs a Vehicle object

Author

Jake Mahloko

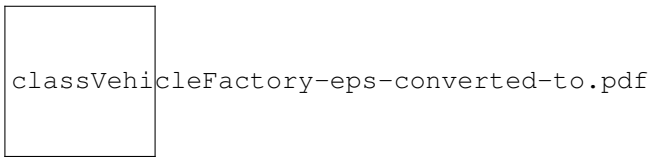
The documentation for this class was generated from the following files:

- Vehicle.h
- Vehicle.cpp

5.40 VehicleFactory Class Reference

```
#include <VehicleFactory.h>
```

Inheritance diagram for VehicleFactory:



Public Member Functions

- VehicleFactory ()

  construct a new VehichleFactory Object
- virtual ∼**VehicleFactory** ()

  destroy a VehicleFactiry object
- virtual Vehicle ∗ manufactureVehicle ()=0

  pure virtual -children classes creates Vehicles referenced objects
- virtual VehicleFactory ∗ clone ()=0

  Create a deep copy of the current object.

5.40.1 Detailed Description

Author

Jake Mahloko brief VehicleFactory class is an abstract class that services as a template class for the children class of VehicleFactory. The factories are used to create objects of Vehicles

5.40.2 Constructor & Destructor Documentation

5.40.2.1 VehicleFactory()

```
VehicleFactory::VehicleFactory ( )
```

construct a new VehichleFactory Object

Author

Jake Mahloko

5.40.3 Member Function Documentation

5.40.3.1 clone()

```
virtual VehicleFactory * VehicleFactory::clone ( )  [pure virtual]
```

Create a deep copy of the current object.

Returns

VehicleFactory∗

Implemented in PlaneFactory, ShipFactory, and TankFactory.

5.40.3.2 manufactureVehicle()

```
virtual Vehicle * VehicleFactory::manufactureVehicle ( )  [pure virtual]
```

pure virtual -children classes creates Vehicles referenced objects

Returns

Vehicle∗ -newly created vehicle

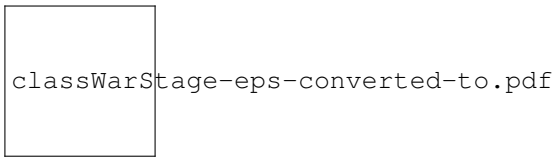
Implemented in PlaneFactory, ShipFactory, and TankFactory.

The documentation for this class was generated from the following files:

- VehicleFactory.h
- VehicleFactory.cpp

5.41 WarStage Class Reference

Inheritance diagram for WarStage:



Public Member Functions

- virtual int getWarStage ()=0

  Virtual function for returning the warstage.
- WarStage ()

  Construct a new War Stage object.
- virtual ∼WarStage ()

  Virtual destructor for the War Stage object.
- virtual WarStage ∗ clone ()=0

  Return a deep copy of the War Stage object.

5.41.1 Constructor & Destructor Documentation

5.41.1.1 WarStage()

```
WarStage::WarStage ( )
```

Construct a new War Stage object.

5.41.1.2 ∼WarStage()

```
WarStage::∼WarStage ( )  [virtual]
```

Virtual destructor for the War Stage object.

5.41.2 Member Function Documentation

5.41.2.1 clone()

```
virtual WarStage * WarStage::clone ( )  [pure virtual]
```

Return a deep copy of the War Stage object.

Returns

WarStage*

Implemented in EarlyStage, LateStage, and MiddleStage.

5.41.2.2 getWarStage()

```
virtual int WarStage::getWarStage ( )  [pure virtual]
```

Virtual function for returning the warstage.

Implemented in EarlyStage, LateStage, and MiddleStage.

The documentation for this class was generated from the following files:

- WarStage.h
- WarStage.cpp

Chapter 6

File Documentation

6.1 Backup.h

```
1
2
3 #ifndef __Backup_h__
4 #define __Backup_h__
5
6 // #include "Memento.h"
7
8 #include <vector>
9
10 class Memento;
11 class Backup;
12
13 class Backup
14 {
15 public:
20     Backup();
21
26     ~Backup();
27
33     void addMemento(Memento *_memento);
34
42     Memento *getMemento();
43
51     int getMementoCount();
52
57     void clear();
58
59 private:
60     std::vector<Memento *> *mementos;
61 };
62
63 #endif
```

6.2 Battalion.h

```
1
2
3 #ifndef __Battalion_h__
4 #define __Battalion_h__
5
6 class Country;
7
8 class Battalion
9 {
10 public:
14     Battalion();
15
20     Battalion(int);
21
25     virtual ~Battalion();
26
31     void attack(Country *enemy);
32
37     void setNumBattalionDestroys(int);
```

```
38
43     int getBattalionDestroyed();
44
45 private:
46     int numBattalionDestroys;
47     int groupSize;
48 };
49
50 #endif
```

6.3 BottomNeighbour.h

```
1
2
3 #ifndef __BottomNeighbour_h__
4 #define __BottomNeighbour_h__
5
6 #include "Neighbour.h"
7
8 class BottomNeighbour : public Neighbour
9 {
10
11 public:
17     BottomNeighbour(Location *_neighbour);
18
24     Location *getBottom();
25
31     bool hasBottom();
32 };
33
34 #endif
```

6.4 Country.h

```
1
2
3 #ifndef __Country_h__
4 #define __Country_h__
5
6 #include <vector>
7 #include <string>
8 #include <exception>
9 #include <cstdlib>
10
11 class WarStage;
12 class Superpower;
13 class Military;
14 class CountryState;
15 class Citizen;
16 class Strategy;
17 class Country;
18 class MapState;
19 class MilitaryState;
20 class Location;
21 class LocationObserver;
22 class Map;
23
24 class Country
25 {
26 public:
30   ~Country();
31
37   Country(std::string _name);
38
42   Country();
43
49   void takeTurn(Country *countryB);
50
56   Country* takeTurn(bool *_countryIsDead);
57
61   void setStrategy();
62
68   CountryState *getState();
69
75   std::string getName();
76
82   void setName(std::string _name);
83
89   Military *getMilitary();
```

```
void getCountryRating(Country *countryB, double *strengthRatings);

double compareAspect(int countryA, int countryB);

double compareAspect(double countryA, double countryB);

int getNumCitizens();

void setNumCitizens(int _numCitizens);

double getPoliticalStability();

void setPoliticalStability(double _politicalStability);

double getDomesticMorale();

void setDomesticMorale(double _domesticMorale);

double getSelfReliance();

void setSelfReliance(double _selfReliance);

double getBorderStrength();

void setBorderStrength(double _borderStrength);

double getCapitalSafety();

void setCapitalSafety(double _capitalSafety);

double getWarSentiment();

void setWarSentiment(double _warSentiment);

double getTradeRouteSafety();

void setTradeRouteSafety(double _tradeRouteSafety);

CountryState *getCountryState();

void setCountryState(CountryState *_countryState);

void compareMilitary(Country *a, Country *b, std::vector<double> *aspectScores);

void compareDomestic(Country *a, Country *b, std::vector<double> *aspectScores);

Location *getCapital();

void setCapital(Location *_capital);

std::vector<Location *> *getLocations();

void setLocations(std::vector<Location *> *_locations);

void setColor(std::string _color);

std::string getColor();

std::vector<Country *> *getEnemies();

void setEnemies(std::vector<Country *> *_enemies);

MilitaryState *getMilitaryState();

void setMilitaryState(MilitaryState *_militaryState);

void setState(CountryState *_state);

void printSummary();

void attachObserver(LocationObserver *_lObserver);

void detachObserver(LocationObserver *_lObserver);

void resetLocations(Map *_map);

Country *clone();

void resetEnemies(std::vector<Country *> *_enemies);

void removeEnemy(Country *_enemy);

void setColorOfDestroyedBy(std::string _newColorOfDestroyedBy);

bool checkIsDead(Country* countryA, Country* countryB);
```

```
  void getBoost(Country* _country);
private:
  // bool checkIsDead();
  std::string colorOfDestroyedBy="\x1B[100m";
  Strategy *strategy;
  Military *military;
  CountryState *countryState;
  std::vector<LocationObserver *> *locationObservers;
};

#endif
```

6.5 CountryState.h

```


#ifndef __CountryState_h__
#define __CountryState_h__

#include <ctime>
#include <string>
#include <vector>

class Country;
class Location;
class MilitaryState;

class CountryState
{
public:
  CountryState();

  CountryState(Country *country);

  CountryState(const CountryState &cs);

  ~CountryState();

  CountryState *clone();

  MilitaryState *getMilitaryState();

  void setMilitaryState(MilitaryState *_militaryState);

  void setIsBeingStored(bool _isBeingStored);

private:
  friend class Country;
  std::string name;
  int numCitizens;
  double domesticMorale;
  double selfReliance;
  double borderStrength;
  double capitalSafety;
  double warSentiment;
  double tradeRouteSafety;
  double politicalStability;
  MilitaryState *militaryState;
  Location *capital;
  std::string color;
  std::vector<Country *> *enemies;
  std::vector<Location *> *locations;
  bool isBeingStored;
};

#endif
```

6.6 EarlyStage.h

```


#ifndef __EarlyStage_h__
#define __EarlyStage_h__

#include "WarStage.h"

class EarlyStage;

class EarlyStage : public WarStage
```

```
{
public:
    int getWarStage();

    EarlyStage();

    ~EarlyStage();

    EarlyStage *clone();
};

#endif
```

6.7 EarlyStrategy.h

```


#ifndef EARLY_STRATEGY_H
#define EARLY_STRATEGY_H

#include "Strategy.h"

#include <iostream>
#include <cmath>

class EarlyStrategy : public Strategy
{
public:
  EarlyStrategy();

  ~EarlyStrategy();

protected:
  void defensiveMove(Country *countryA, Country *countryB);

  void neutralMove(Country *countryA, Country *countryB);

  void offensiveMove(Country *countryA, Country *countryB);
};

#endif // EARLY_STRATEGY_H
```

6.8 Iterator.h

```


#ifndef __Iterator_h__
#define __Iterator_h__

#include <exception>
#include "Location.h"

class Location;

class Iterator
{

public:
    virtual void next() = 0;

    virtual void first() = 0;

    virtual bool isDone() = 0;

    virtual Location *getCurrent() = 0;

protected:
    Location *current;
};

#endif
```

6.9 LateStage.h

```
```

```

#ifndef __LateStage_h__
#define __LateStage_h__

#include "WarStage.h"

class LateStage;

class LateStage : public WarStage
{

public:
        int getWarStage();

        LateStage();

        ~LateStage();

        LateStage *clone();
};

#endif
```

6.10 LateStrategy.h

```


#ifndef LATE_STRATEGY_H
#define LATE_STRATEGY_H

#include "Strategy.h"
#include <cmath>

class LateStrategy : public Strategy
{
public:
  LateStrategy();

  ~LateStrategy();

protected:
  void defensiveMove(Country *countryA, Country *countryB);

  void neutralMove(Country *countryA, Country *countryB);

  void offensiveMove(Country *countryA, Country *countryB);
};

#endif
```

6.11 LeftNeighbour.h

```


#ifndef __LeftNeighbour_h__
#define __LeftNeighbour_h__

#include "Neighbour.h"

class LeftNeighbour : public Neighbour
{

public:
    LeftNeighbour(Location *_neighbour);

    Location *getLeft();

    bool hasLeft();
};

#endif
```

6.12 Location.h

```

```

```

#ifndef __Location_h__
#define __Location_h__

#include <string>

class Map;
class LocationObserver;
class Iterator;
class Country;

class Location
{

public:
    virtual ~Location();

    Iterator *createIterator();

    virtual Location *getRight();

    virtual Location *getLeft();

    virtual Location *getTop();

    virtual Location *getBottom();

    virtual void add(Location *_neighbour) = 0;

    virtual bool hasBottom();

    virtual bool hasRight();

    virtual bool hasLeft();

    virtual bool hasTop();

    Country *getOwnedBy();

    void setOwnedBy(Country *_newOwner);

    Location *clone();

    std::string getColor();

    void setColor(/*char _colour*/ std::string _color);

    int getX();

    int getY();

    bool getIsLand();

    void setIsLand(bool _isLand);

    bool getIsCapital();

    void setIsCapital(bool _isCapital);

protected:
    Location *location;
    Country *ownedBy;
    LocationObserver *lObserver;
    std::string color;
    bool isCapital;
    bool isLand;
    int xCoordinate, yCoordinate;
};

#endif
```

6.13 LocationIterator.h

```


#ifndef __LocationIterator_h__
#define __LocationIterator_h__

#include <exception>
#include "Iterator.h"

class Location;

```

```
class LocationIterator : public Iterator
{

public:
    LocationIterator(Location *_location);

    ~LocationIterator();

    void next();

    void first();

    bool isDone();

    Location *getCurrent();

protected:
    Location *current;
    Location *nextLocation;
    bool hasNext();
    Location *nextRow();
};

#endif
```

6.14 LocationObserver.h

```


#ifndef __LocationObserver_h__
#define __LocationObserver_h__

#include <string>

class Location;

class LocationObserver
{
public:
    LocationObserver(Location *_location);
    ~LocationObserver();
    void updateLocation(std::string _newColor);

private:
    Location *location;
};
#endif
```

6.15 Map.h

```


#ifndef __Map_h__
#define __Map_h__

class Location;
class Territory;
class MapState;

#include "Territory.h"

class Map
{

public:
    Map();

    Map(Location *_cloneTopLeft);

    ~Map();

    Map(Map *_oldMap);

    Location *getLocation(int _x, int _y);

    Location *getTopLeft();

    void printMap();

```

```
    MapState *getState();

private:
    void printLocation(std::string _col);
    Location *topLeft;
};

#endif
```

6.16 MapState.h

```


#ifndef __MapState_h__
#define __MapState_h__

#include <ctime>
#include "Map.h"

class MapState
{

public:
    MapState(Map *_m);

    ~MapState();

    Map *clone();

private:
    Map *mapState;
    std::time_t timestamp;
};

#endif
```

6.17 Memento.h

```


#ifndef __Memento_h__
#define __Memento_h__

class Backup;
class SimulationState;

class Memento
{
public:
    Memento();

    Memento(SimulationState *_simulationState);

    ~Memento();

    SimulationState *getState();

    void setState(SimulationState *_simulationState);

private:
    SimulationState *state;
};

#endif
```

6.18 MiddleStage.h

```


#ifndef __MiddleStage_h__
#define __MiddleStage_h__

#include "WarStage.h"

// class WarStage;
```

```
class MiddleStage;

class MiddleStage : public WarStage
{

public:
        int getWarStage();

        MiddleStage();

        ~MiddleStage();

        MiddleStage *clone();
};

#endif
```

6.19 MiddleStrategy.h

```


#ifndef MIDDLE_STRATEGY_H
#define MIDDLE_STRATEGY_H

#include <iostream>
#include <cmath>

#include "Strategy.h"

class MiddleStrategy : public Strategy
{
public:
  MiddleStrategy();

  ~MiddleStrategy();

protected:
  void defensiveMove(Country* countryA, Country* countryB);

  void neutralMove(Country* countryA, Country* countryB);

  void offensiveMove(Country* countryA, Country* countryB);
};

#endif // MIDDLE_STRATEGY_H
```

6.20 Military.h

```


#ifndef __Military_h__
#define __Military_h__

#include <vector>

class Country;

class Military
{
public:
  Military();

  Military(Military * _military);

  ~Military();

  void attack(Country * _country);
};

#endif
```

6.21 MilitaryState.h

```


```

```
#ifndef __MilitaryState_h__
#define __MilitaryState_h__

#include <vector>
#include <exception>
#include <stdexcept>
#include <iostream>

class Plane;
class Ship;
class Tank;
class Battalion;
class VehicleFactory;

class MilitaryState
{

public:
  MilitaryState();

  ~MilitaryState();

  void setShips(std::vector<Ship *> *_ships);

  void setPlanes(std::vector<Plane *> *_planes);

  void setTanks(std::vector<Tank *> *_tanks);

  void setBattalions(std::vector<Battalion *> *_battalions);

  void setTroops(int _troops);

  void setVehicleFactories(std::vector<VehicleFactory *> *_vehicleFactories);

  int getNumTroops();

  void updateNumTroops(int _numTroops, bool isAddition);

  int getNumTanks();

  void updateNumTanks(int _numTanks, bool isAddition);

  int getNumShips();

  void updateNumShips(int _numShips, bool isAddition);

  int getNumPlanes();

  void updateNumPlanes(int _numPlanes, bool isAddition);

  int getNumBattalions();

  void updateNumBattalions(int _numBattalions, bool isAddition);

  MilitaryState *clone();

private:
  int numTroops;
  std::vector<Tank *> *tanks;
  std::vector<Ship *> *ships;
  std::vector<Plane *> *planes;
  std::vector<Battalion *> *battalions;
  std::vector<VehicleFactory *> *vehicleFactories;
};

#endif
```

6.22 Neighbour.h

```


#ifndef __Neighbour_h__
#define __Neighbour_h__

#include "Location.h"

class Neighbour : public Location
{

public:
    Neighbour(Location *_neighbour);

    virtual ~Neighbour();
```

```
24
30     virtual void add(Location *_neighbour);
31
32 protected:
33     Location *neighbour;
34 };
35
36 #endif
```

6.23 Plane.h

```
1
2
3 #ifndef __Plane_h__
4 #define __Plane_h__
5
6 class Country;
7
8 #include "Vehicle.h"
9 #include <exception>
10
11 class Plane : public Vehicle
12 {
13
14 public:
18     Plane();
19
23     ~Plane();
24 };
25
26 #endif
```

6.24 PlaneFactory.h

```
1
2
3 #ifndef __PlaneFactory_h__
4 #define __PlaneFactory_h__
5
6 #include "VehicleFactory.h"
7 #include <exception>
8
13 class PlaneFactory : public VehicleFactory
14 {
15
16 public:
20     PlaneFactory();
24     ~PlaneFactory();
29     Vehicle *manufactureVehicle();
30     // Vehicle *manufactureVehicle(int, int, int);
31
37     PlaneFactory *clone();
38 };
39
40 #endif
```

6.25 RightNeighbour.h

```
1
2
3 #ifndef __RightNeighbour_h__
4 #define __RightNeighbour_h__
5
6 #include "Neighbour.h"
7
8 class RightNeighbour : public Neighbour
9 {
10
11 public:
17     RightNeighbour(Location *_neighbour);
18
24     Location *getRight();
25
31     bool hasRight();
32 };
33
34 #endif
```

6.26 Ship.h

```


#ifndef __Ship_h__
#define __Ship_h__

#include <exception>

#include "Vehicle.h"
#include <exception>

class Ship : public Vehicle
{
public:
    Ship();

    ~Ship();
};

#endif
```

6.27 ShipFactory.h

```


#include <exception>

#ifndef __ShipFactory_h__
#define __ShipFactory_h__

#include "VehicleFactory.h"
class ShipFactory : public VehicleFactory
{

public:
    ShipFactory();
    ~ShipFactory();
    Vehicle *manufactureVehicle();
    // Vehicle *manufactureVehicle(int, int);

    ShipFactory *clone();
};

#endif
```

6.28 SimulationManager.h

```


#ifndef __SimulationManager_h__
#define __SimulationManager_h__

#include <vector>

class Map;
class SimulationState;
class Superpower;
class Backup;
class SimulationManager;
class Country;

class SimulationManager
{
public:
    SimulationManager();

    virtual ~SimulationManager();

    void runSimulation();

protected:
    bool restoreState();

    void saveState();

    void resetSimulation();

    bool isSimulationRunning();
```

```

    void takeTurn();

    void viewSummary();

    void processMenu();

    void viewCountrySummary();

    void designModeAction();

     void finalMessage();

private:
     Map *map;
     std::vector<Superpower *> *superpowers;
     Backup *backup;
     bool designMode, isRunning;
     int turnCount, maxTurnCount;
     void setSuperpowers();
     void setDesignMode();
     void setUpUK(Country *_uk);
     void setUpFrance(Country *_france);
     void setUpBalkans(Country *_balkans);
     void setUpSpainPortugal(Country *_spainPortugal);
     void setUpSovietUnion(Country *_sovietUnion);
     void setUpScandanavia(Country *_scandanavia);
     void setUpGermany(Country *_germany);
     void setUpItaly(Country *_italy);
     void changeSimulationLength();
     void removeCountry();
     void alterCountryState();
     void changeWarStage();
     void changeBorderStrength(Country *_country);
     void changePopulation(Country *_country);
     void changePoliticalStability(Country *_country);
     void changeSelfReliance(Country *_country);
     void changeWarSentiment(Country *_country);
     void changeTradeRouteSafety(Country *_country);
     void changeMilitaryAttributes(Country *_country);
};

#endif
```

6.29 SimulationState.h

```


#ifndef __SimulationState_h__
#define __SimulationState_h__

#include <vector>
#include <ctime>

class MapState;
class SuperpowerState;
class StageContextState;

class SimulationState
{
public:
    SimulationState();

    ~SimulationState();

    void setMapState(MapState *_mapState);

    void setStageContextState(StageContextState *_stageContextState);

    void addSuperpowerState(SuperpowerState *_superpowerState);

    int getSuperpowerStateCount();

    SuperpowerState *getSuperpowerState(int _index);

    MapState *getMapState();

    StageContextState *getStageContextState();

    std::time_t getTimestamp();

private:
    MapState *mapState;
```

```
    StageContextState *stageContextState;
    std::vector<SuperpowerState *> *superpowerStates;
    std::time_t timestamp;
};

#endif
```

6.30 StageContext.h

```


#ifndef __StageContext_h__
#define __StageContext_h__

#include <string>

class Country;
class WarStage;
class StageContextState;

class StageContext
{
public:
    StageContextState *getState();

    int getCurrentRound();

    static StageContext *getInstance();

    int getWarStage();

    void incrementRound();

    void setSimulationLength(int _length);

    ~StageContext();

    void setCurrentRound(int _round);

    void setCurrentStage(WarStage *_stage);

    void setState(StageContextState *_state);

    int getSimulationLength();

    void moveToStage(int _stage);

protected:
    StageContext();
    StageContext(int _simulationLength);
    int simulationLength;
    int currentRound;
    static StageContext *onlyInstance;
    WarStage *currentStage;
};

#endif
```

6.31 StageContextState.h

```


#ifndef __StageContextState_h__
#define __StageContextState_h__

class StageContext;
class WarStage;

class StageContextState
{
public:
    StageContextState();

    ~StageContextState();

    void setSimulationLength(int _length);

    int getSimulationLength();

```

```
    void setCurrentRound(int _round);

    int getCurrentRound();

    void setCurrentStage(WarStage *_stage);

    WarStage *getCurrentStage();

private:
    int simulationLength;
    int currentRound;
    WarStage *currentStage;
};
#endif
```

6.32 Strategy.h

```


#ifndef __Strategy_h__
#define __Strategy_h__

// #include "Country.h"
#include <exception>

class Country;

class Strategy
{
public:
  Strategy(); //Let strategy be reusable by not holding a country object

  virtual void takeTurn(double* strengthRatings, Country* countryA, Country* countryB);

protected:
  virtual void offensiveMove(Country* countryA, Country* countryB) = 0;

  virtual void neutralMove(Country* countryA, Country* countryB) = 0;

  virtual void defensiveMove(Country* countryA, Country* countryB) = 0;



};

#endif
```

6.33 Superpower.h

```


#ifndef __Superpower_h__
#define __Superpower_h__

#include <string>
#include <vector>

class Country;
class SuperpowerState;
class Map;

class Superpower
{
public:
  Superpower(std::string _name);

  Superpower(SuperpowerState *_state);

  ~Superpower();

  std::string getName();

  void addCountry(Country *_country);

  int getCountryCount();

  Country *getCountry(int _index);

  void removeCountry(Country *_country);
```

```

  SuperpowerState *getState();

  void printSummary();

  void resetLocations(Map *_map);

  void resetEnemies(std::vector<Country *> *_enemies);

  std::vector<Country *> *getAllCountries();

private:
  std::vector<Country *> *countries;
  std::string name;
};

#endif
```

6.34 SuperpowerState.h

```


#ifndef __SuperpowerState_h__
#define __SuperpowerState_h__

#include <vector>
#include <string>

class CountryState;

class SuperpowerState
{
public:
    SuperpowerState(std::string _name);

    ~SuperpowerState();

    void addCountryState(CountryState *_countryState);

    std::string getName();

    int getCountryStateCount();

    CountryState *getCountryState(int _index);

protected:
    std::string name;
    std::vector<CountryState *> *countryStates;
};
#endif
```

6.35 Tank.h

```


#ifndef __Tank__h__
#define __Tank__h__

#include <exception>
#include "Vehicle.h"

class Tank : public Vehicle
{

public:
    Tank();

    ~Tank();
};

#endif
```

6.36 TankFactory.h

```
```

```

#ifndef __TankFactory_h__
#define __TankFactory_h__

#include "VehicleFactory.h"
#include <exception>
class TankFactory : public VehicleFactory
{

public:
    TankFactory();
    ~TankFactory();
    Vehicle *manufactureVehicle();
    TankFactory *clone();
};

#endif
```

6.37 Territory.h

```


#ifndef __Territory_h__
#define __Territory_h__

#include <exception>
#include "Location.h"
#include "LocationObserver.h"

class LocationIterator;
class Neighbour;
class Country;

class Territory : public Location
{

public:
    Territory(int _x, int _y, std::string _color = "\x1B[44m");

    ~Territory();

    void add(Location *_neighbour);
};

#endif
```

6.38 TopNeighbour.h

```


#ifndef __TopNeighbour_h__
#define __TopNeighbour_h__

#include "Neighbour.h"

class TopNeighbour : public Neighbour
{

public:
    TopNeighbour(Location *_neighbour);

    Location *getTop();

    bool hasTop();
};

#endif
```

6.39 Vehicle.h

```


#ifndef __Vehicle_h__
#define __Vehicle_h__

```

```
#include <exception>

class Vehicle
{

public:
    Vehicle();

    virtual ~Vehicle();
};

#endif
```

6.40 VehicleFactory.h

```


#ifndef __VehicleFactory_h__
#define __VehicleFactory_h__

#include "Vehicle.h"
#include <exception>
class VehicleFactory
{
public:
    VehicleFactory();
    virtual ~VehicleFactory();
    virtual Vehicle *manufactureVehicle() = 0;

    virtual VehicleFactory *clone() = 0;
};

#endif
```

6.41 WarStage.h

```


#include <exception>
#include <string>

#ifndef __WarStage_h__
#define __WarStage_h__

class Country;
class WarStage;

class WarStage
{
public:
        virtual int getWarStage() = 0;

        WarStage();

        virtual ~WarStage();

        virtual WarStage *clone() = 0;
};

#endif
```