

# COS214PROJECT

1st

Generated by Doxygen 1.9.5



<b>1 Hierarchical Index</b>	<b>1</b>
1.1 Class Hierarchy	1
<b>2 Class Index</b>	<b>3</b>
2.1 Class List	3
<b>3 File Index</b>	<b>7</b>
3.1 File List	7
<b>4 Class Documentation</b>	<b>9</b>
4.1 Ambush Class Reference	9
4.1.1 Detailed Description	10
4.1.2 Constructor & Destructor Documentation	10
4.1.2.1 Ambush()	10
4.1.3 Member Function Documentation	11
4.1.3.1 attack()	11
4.1.4 Member Data Documentation	11
4.1.4.1 stealth	11
4.2 ArmySupplies Class Reference	11
4.2.1 Detailed Description	12
4.2.2 Member Function Documentation	12
4.2.2.1 getAmount()	12
4.3 Assassinate Class Reference	12
4.3.1 Detailed Description	13
4.3.2 Constructor & Destructor Documentation	13
4.3.2.1 Assassinate()	13
4.3.3 Member Function Documentation	14
4.3.3.1 attack()	14
4.3.4 Member Data Documentation	14
4.3.4.1 alive	14
4.3.4.2 stealth	15
4.4 Bannerman Class Reference	15
4.4.1 Member Function Documentation	17
4.4.1.1 attach()	17
4.4.1.2 attack()	17
4.4.1.3 changeStrategy()	17
4.4.1.4 decreaseFavour()	18
4.4.1.5 decreaseFood()	18
4.4.1.6 decreaseMedical()	18
4.4.1.7 decreasePower()	18
4.4.1.8 decreaseWeapons()	19
4.4.1.9 detach()	19
4.4.1.10 getDamage()	19

4.4.1.11 getFavour()	19
4.4.1.12 getFood()	20
4.4.1.13 getHP()	20
4.4.1.14 getMedical()	20
4.4.1.15 getName()	20
4.4.1.16 getTroops()	21
4.4.1.17 getWeapons()	21
4.4.1.18 increaseFavour()	21
4.4.1.19 increaseHP()	21
4.4.1.20 increasePower()	22
4.4.1.21 receiveDamage()	22
4.4.1.22 setFood()	22
4.4.1.23 setMaster()	23
4.4.1.24 setMedical()	23
4.4.1.25 setRaven()	23
4.4.1.26 setWeapons()	23
4.4.2 Member Data Documentation	25
4.4.2.1 assassin	25
4.4.2.2 damage	25
4.4.2.3 favour	25
4.4.2.4 HP	25
4.4.2.5 m	25
4.4.2.6 name	25
4.4.2.7 numFood	26
4.4.2.8 numMedical	26
4.4.2.9 numWeapons	26
4.4.2.10 ravenList	26
4.4.2.11 strategy	26
4.5 BattleField Class Reference	26
4.5.1 Detailed Description	27
4.5.2 Constructor & Destructor Documentation	27
4.5.2.1 BattleField()	27
4.5.3 Member Function Documentation	28
4.5.3.1 attack()	28
4.6 Class Class Reference	28
4.7 Commander Class Reference	28
4.7.1 Detailed Description	30
4.7.2 Constructor & Destructor Documentation	30
4.7.2.1 Commander()	30
4.7.3 Member Function Documentation	31
4.7.3.1 attach()	31
4.7.3.2 attack()	31

4.7.3.3 changeStrategy()	31
4.7.3.4 decreaseFavour()	32
4.7.3.5 decreaseFood()	32
4.7.3.6 decreaseMedical()	32
4.7.3.7 decreasePower()	32
4.7.3.8 decreaseWeapons()	33
4.7.3.9 detach()	33
4.7.3.10 getDamage()	33
4.7.3.11 getFavour()	33
4.7.3.12 getFood()	34
4.7.3.13 getHP()	34
4.7.3.14 getMedical()	34
4.7.3.15 getName()	34
4.7.3.16 getTroops()	35
4.7.3.17 getWeapons()	35
4.7.3.18 increaseFavour()	35
4.7.3.19 increaseHP()	35
4.7.3.20 increasePower()	36
4.7.3.21 receiveDamage()	36
4.7.3.22 setFood()	36
4.7.3.23 setMaster()	36
4.7.3.24 setMedical()	37
4.7.3.25 setRaven()	37
4.7.3.26 setStrategy()	37
4.7.3.27 setWeapons()	38
4.8 Conditions Class Reference	38
4.8.1 Detailed Description	39
4.8.2 Constructor & Destructor Documentation	39
4.8.2.1 Conditions()	39
4.8.3 Member Function Documentation	39
4.8.3.1 sendScout()	39
4.8.4 Member Data Documentation	40
4.8.4.1 myVenue	40
4.9 Conlterator Class Reference	40
4.9.1 Detailed Description	41
4.9.2 Constructor & Destructor Documentation	41
4.9.2.1 Conlterator()	41
4.9.3 Member Function Documentation	41
4.9.3.1 Current()	41
4.9.3.2 hasNext()	42
4.9.3.3 isActive()	42
4.9.3.4 next()	42

4.9.4 Member Data Documentation	43
4.9.4.1 armyList	43
4.9.4.2 it	43
4.10 Economy Class Reference	43
4.10.1 Detailed Description	44
4.10.2 Constructor & Destructor Documentation	44
4.10.2.1 Economy()	44
4.10.3 Member Function Documentation	44
4.10.3.1 getCurrency()	44
4.10.3.2 getState()	45
4.10.3.3 removeCurrency()	45
4.10.4 Member Data Documentation	45
4.10.4.1 currency	45
4.10.4.2 state	45
4.11 Factory Class Reference	46
4.11.1 Detailed Description	46
4.11.2 Member Function Documentation	46
4.11.2.1 getSupply()	47
4.11.2.2 make()	47
4.11.3 Member Data Documentation	47
4.11.3.1 supply	47
4.12 FailedState Class Reference	47
4.12.1 Detailed Description	48
4.12.2 Member Function Documentation	48
4.12.2.1 decreaseCurrency()	48
4.12.2.2 getState()	48
4.13 FoodFac Class Reference	49
4.13.1 Detailed Description	49
4.13.2 Member Function Documentation	49
4.13.2.1 make()	49
4.14 FoodSupp Class Reference	50
4.14.1 Detailed Description	50
4.14.2 Member Function Documentation	50
4.14.2.1 getAmount()	50
4.15 FoodWagon Class Reference	51
4.15.1 Detailed Description	51
4.15.2 Member Function Documentation	51
4.15.2.1 clone()	51
4.15.2.2 getSupplies()	52
4.15.2.3 setSup()	52
4.15.3 Member Data Documentation	52
4.15.3.1 supp	52

4.16 HealthyState Class Reference . . . . .	53
4.16.1 Detailed Description . . . . .	53
4.16.2 Member Function Documentation . . . . .	53
4.16.2.1 decreaseCurrency() . . . . .	54
4.16.2.2 getDemotionState() . . . . .	54
4.16.2.3 getState() . . . . .	54
4.17 Historian Class Reference . . . . .	54
4.17.1 Member Function Documentation . . . . .	55
4.17.1.1 restoreAlly() . . . . .	55
4.17.1.2 setAlly() . . . . .	55
4.17.1.3 Store() . . . . .	55
4.17.2 Member Data Documentation . . . . .	56
4.17.2.1 bannerman . . . . .	56
4.18 History Class Reference . . . . .	56
4.18.1 Constructor & Destructor Documentation . . . . .	56
4.18.1.1 History() . . . . .	56
4.18.2 Member Function Documentation . . . . .	57
4.18.2.1 getBannerman() . . . . .	57
4.18.3 Member Data Documentation . . . . .	57
4.18.3.1 bannerman . . . . .	57
4.19 HistoryBook Class Reference . . . . .	58
4.19.1 Member Function Documentation . . . . .	58
4.19.1.1 add() . . . . .	58
4.19.1.2 getAlly() . . . . .	58
4.19.2 Member Data Documentation . . . . .	58
4.19.2.1 defectedAllies . . . . .	59
4.20 Iterator Class Reference . . . . .	59
4.20.1 Detailed Description . . . . .	60
4.20.2 Member Function Documentation . . . . .	60
4.20.2.1 Current() . . . . .	60
4.20.2.2 hasNext() . . . . .	60
4.20.2.3 isActive() . . . . .	61
4.20.2.4 next() . . . . .	61
4.21 Kingdom Class Reference . . . . .	61
4.21.1 Detailed Description . . . . .	62
4.21.2 Constructor & Destructor Documentation . . . . .	62
4.21.2.1 Kingdom() . . . . .	62
4.21.3 Member Function Documentation . . . . .	62
4.21.3.1 add() . . . . .	62
4.21.3.2 getAlly() . . . . .	63
4.21.3.3 getKingdom() . . . . .	63
4.21.3.4 getSize() . . . . .	63

4.21.3.5 remove()	63
4.21.4 Member Data Documentation	64
4.21.4.1 bannerman	64
4.21.4.2 economy	64
4.22 Location Class Reference	64
4.22.1 Detailed Description	65
4.22.2 Member Function Documentation	65
4.22.2.1 sendScout()	65
4.23 MasterOfCoin Class Reference	65
4.23.1 Detailed Description	66
4.23.2 Constructor & Destructor Documentation	66
4.23.2.1 MasterOfCoin()	66
4.23.3 Member Function Documentation	67
4.23.3.1 notify()	67
4.24 MedicalFac Class Reference	67
4.24.1 Detailed Description	68
4.24.2 Member Function Documentation	68
4.24.2.1 make()	68
4.25 MedicalSupp Class Reference	68
4.25.1 Detailed Description	69
4.25.2 Member Function Documentation	69
4.25.2.1 getAmount()	69
4.26 MedicalWagon Class Reference	69
4.26.1 Detailed Description	70
4.26.2 Member Function Documentation	70
4.26.2.1 clone()	70
4.26.2.2 getSupplies()	70
4.26.2.3 setSup()	70
4.26.3 Member Data Documentation	71
4.26.3.1 supp	71
4.27 Raven Class Reference	71
4.27.1 Detailed Description	71
4.27.2 Member Function Documentation	72
4.27.2.1 update()	72
4.28 sendRaven Class Reference	72
4.28.1 Detailed Description	73
4.28.2 Constructor & Destructor Documentation	73
4.28.2.1 sendRaven()	73
4.28.3 Member Function Documentation	73
4.28.3.1 update()	74
4.29 Siege Class Reference	74
4.29.1 Detailed Description	75



4.29.2 Constructor & Destructor Documentation	75
4.29.2.1 Siege()	75
4.29.3 Member Function Documentation	75
4.29.3.1 attack()	76
4.29.4 Member Data Documentation	76
4.29.4.1 stealth	76
4.30 State Class Reference	76
4.30.1 Detailed Description	77
4.30.2 Member Function Documentation	77
4.30.2.1 decreaseCurrency()	77
4.30.2.2 getDemotionState()	78
4.30.2.3 getState()	78
4.30.2.4 setContext()	78
4.30.3 Member Data Documentation	78
4.30.3.1 context	78
4.31 Strategy Class Reference	79
4.31.1 Detailed Description	80
4.31.2 Constructor & Destructor Documentation	80
4.31.2.1 Strategy()	80
4.31.3 Member Function Documentation	80
4.31.3.1 attack()	81
4.31.3.2 getEnemyBannerman()	81
4.31.3.3 getMyBannerman()	81
4.31.3.4 getStrategyName()	82
4.31.4 Member Data Documentation	82
4.31.4.1 BookOfDura	82
4.31.4.2 defectedAllies	82
4.31.4.3 enemyBannerman	82
4.31.4.4 enemyKingdom	82
4.31.4.5 Greg	82
4.31.4.6 minFavour	83
4.31.4.7 minSupplies	83
4.31.4.8 myBannerman	83
4.31.4.9 myKingdom	83
4.31.4.10 strategy	83
4.32 SupplyWagon Class Reference	83
4.32.1 Detailed Description	84
4.32.2 Member Function Documentation	84
4.32.2.1 clone()	84
4.32.2.2 getSupplies()	84
4.32.2.3 setSup()	84
4.32.3 Member Data Documentation	85

4.32.3.1 supp	85
4.33 Topology Class Reference	85
4.33.1 Detailed Description	86
4.33.2 Member Function Documentation	86
4.33.2.1 sendScout()	86
4.34 Treasury Class Reference	86
4.34.1 Member Function Documentation	86
4.34.1.1 notify()	87
4.35 Troop Class Reference	87
4.35.1 Detailed Description	89
4.35.2 Constructor & Destructor Documentation	89
4.35.2.1 ~Troop()	89
4.35.3 Member Function Documentation	89
4.35.3.1 attach()	89
4.35.3.2 attack()	89
4.35.3.3 changeStrategy()	90
4.35.3.4 decreaseFavour()	90
4.35.3.5 decreaseFood()	90
4.35.3.6 decreaseMedical()	91
4.35.3.7 decreasePower()	91
4.35.3.8 decreaseWeapons()	91
4.35.3.9 detach()	91
4.35.3.10 getDamage()	92
4.35.3.11 getFavour()	92
4.35.3.12 getFood()	92
4.35.3.13 getHP()	92
4.35.3.14 getMedical()	93
4.35.3.15 getName()	93
4.35.3.16 getSize()	93
4.35.3.17 getTroops()	93
4.35.3.18 getWeapons()	94
4.35.3.19 increaseFavour()	94
4.35.3.20 increaseHP()	94
4.35.3.21 increasePower()	94
4.35.3.22 receiveDamage()	95
4.35.3.23 setFood()	95
4.35.3.24 setMaster()	95
4.35.3.25 setMedical()	96
4.35.3.26 setRaven()	96
4.35.3.27 setWeapons()	96
4.35.4 Member Data Documentation	96
4.35.4.1 damage	97

---

4.35.4.2 size	97
4.35.4.3 strategy	97
4.35.4.4 warZone	97
4.36 UnstableState Class Reference	97
4.36.1 Detailed Description	98
4.36.2 Member Function Documentation	98
4.36.2.1 decreaseCurrency()	98
4.36.2.2 getDemotionState()	98
4.36.2.3 getState()	99
4.37 WarIndicators Class Reference	99
4.37.1 Detailed Description	100
4.37.2 Constructor & Destructor Documentation	100
4.37.2.1 WarIndicators()	100
4.37.3 Member Function Documentation	100
4.37.3.1 setTreasury()	100
4.37.4 Member Data Documentation	101
4.37.4.1 m	101
4.38 WarTheatre Class Reference	101
4.38.1 Detailed Description	102
4.38.2 Member Function Documentation	102
4.38.2.1 decideVenue()	102
4.38.2.2 getDifficulty()	102
4.38.2.3 getStrategy()	103
4.38.2.4 getVenue()	103
4.38.2.5 sendScout()	103
4.38.2.6 setVenue()	103
4.38.3 Member Data Documentation	103
4.38.3.1 difficulty	104
4.38.3.2 location	104
4.38.3.3 strategy	104
4.38.3.4 venue	104
4.39 WeaponsFac Class Reference	104
4.39.1 Detailed Description	105
4.39.2 Member Function Documentation	105
4.39.2.1 make()	105
4.40 WeaponSupp Class Reference	105
4.40.1 Detailed Description	106
4.40.2 Member Function Documentation	106
4.40.2.1 getAmount()	106
4.41 WeaponWagon Class Reference	106
4.41.1 Detailed Description	107
4.41.2 Member Function Documentation	107

4.41.2.1 clone()	107
4.41.2.2 getSupplies()	107
4.41.2.3 setSup()	107
4.41.3 Member Data Documentation	108
4.41.3.1 supp	108
4.42 Weather Class Reference	108
4.42.1 Detailed Description	109
4.42.2 Constructor & Destructor Documentation	109
4.42.2.1 Weather()	109
4.42.3 Member Function Documentation	109
4.42.3.1 calcEffect()	109
4.42.3.2 sendScout()	110
4.42.3.3 setTemp()	110
4.42.3.4 setWindSpeed()	110
4.42.4 Member Data Documentation	110
4.42.4.1 rain	110
4.42.4.2 temp	110
4.42.4.3 windspeed	111
<b>5 File Documentation</b>	<b>113</b>
5.1 Ambush.h	113
5.2 ArmySupplies.h	113
5.3 Assassinate.h	113
5.4 Bannerman.h	114
5.5 BattleField.h	115
5.6 Class.h	115
5.7 Commander.h	115
5.8 Conditions.h	116
5.9 Conliterator.h	117
5.10 Economy.h	117
5.11 Factory.h	117
5.12 FailedState.h	118
5.13 FoodFac.h	118
5.14 FoodSupp.h	118
5.15 FoodWagon.h	118
5.16 HealthyState.h	119
5.17 Historian.h	119
5.18 History.h	119
5.19 HistoryBook.h	119
5.20 Iterator.h	120
5.21 Kingdom.h	120
5.22 Location.h	120

---

5.23 MasterOfCoin.h . . . . .	121
5.24 MedicalFac.h . . . . .	121
5.25 MedicalSupp.h . . . . .	121
5.26 MedicalWagon.h . . . . .	122
5.27 Raven.h . . . . .	122
5.28 sendRaven.h . . . . .	122
5.29 Siege.h . . . . .	122
5.30 State.h . . . . .	123
5.31 Strategy.h . . . . .	123
5.32 SupplyWagon.h . . . . .	124
5.33 Topology.h . . . . .	124
5.34 Treasury.h . . . . .	124
5.35 Troop.h . . . . .	124
5.36 UnstableState.h . . . . .	125
5.37 WarIndicators.h . . . . .	126
5.38 WarTheatre.h . . . . .	126
5.39 WeaponsFac.h . . . . .	126
5.40 WeaponSupp.h . . . . .	127
5.41 WeaponWagon.h . . . . .	127
5.42 Weather.h . . . . .	127
<b>Index</b>	<b>129</b>



# Chapter 1

## Hierarchical Index

### 1.1 Class Hierarchy

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

ArmySupplies . . . . .	11
FoodSupp . . . . .	50
MedicalSupp . . . . .	68
WeaponSupp . . . . .	105
Class . . . . .	28
Historian . . . . .	54
History . . . . .	56
HistoryBook . . . . .	58
Iterator . . . . .	59
ConIterator . . . . .	40
Kingdom . . . . .	61
Raven . . . . .	71
sendRaven . . . . .	72
State . . . . .	76
FailedState . . . . .	47
HealthyState . . . . .	53
UnstableState . . . . .	97
SupplyWagon . . . . .	83
FoodWagon . . . . .	51
MedicalWagon . . . . .	69
WeaponWagon . . . . .	106
Treasury . . . . .	86
MasterOfCoin . . . . .	65
WarIndicators . . . . .	99
Bannerman . . . . .	15
Commander . . . . .	28
Commander . . . . .	28
Troop . . . . .	87
Troop . . . . .	87
Economy . . . . .	43
Factory . . . . .	46
FoodFac . . . . .	49
MedicalFac . . . . .	67

WeaponsFac . . . . .	104
Strategy . . . . .	79
Ambush . . . . .	9
Assassinate . . . . .	12
BattleField . . . . .	26
Siege . . . . .	74
WarTheatre . . . . .	101
Conditions . . . . .	38
Topology . . . . .	85
Weather . . . . .	108
Location . . . . .	64



## Chapter 2

# Class Index

### 2.1 Class List

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

<a href="#">Ambush</a>	A concrete strategy class. A class that provides an alternative strategy for how <a href="#">Bannerman</a> will fight enemyBannerman . . . . .	9
<a href="#">ArmySupplies</a>	Abstract <a href="#">Class</a> containing the information on the supplies . . . . .	11
<a href="#">Assassinate</a>	A concrete strategy class. A class that provides an alternative strategy for how <a href="#">Bannerman</a> will fight enemyBannerman . . . . .	12
<a href="#">Bannerman</a>	. . . . .	15
<a href="#">BattleField</a>	A concrete strategy class. A class that provides an alternative strategy for how <a href="#">Bannerman</a> will fight enemyBannerman . . . . .	26
<a href="#">Class</a>	. . . . .	28
<a href="#">Commander</a>	A class that acts as a container/composite for bannerman objects as well as performing operations on and using various bannerman objects . . . . .	28
<a href="#">Conditions</a>	Decorator class. This is the class/object decorates the concreteComponent. Inherits from <a href="#">WarTheatre</a> class . . . . .	38
<a href="#">Conlterator</a>	Concrete iterator class Implements the interface for accessing and traversing bannerman elements in groundForces. Keeps track of the current position in the traversal of the aggregate(groundForces list) . . . . .	40
<a href="#">Economy</a>	Keeps all variables relating to the state of <a href="#">Economy</a> , is context of States. A class that provides an alternative strategy for how <a href="#">Bannerman</a> will fight enemyBannerman . . . . .	43
<a href="#">Factory</a>	<a href="#">Factory</a> class to serve as a base class for all factory concrete classes used to create supplies .	46
<a href="#">FailedState</a>	A concrete state class. A class that is one of the alternative concrete states for <a href="#">Economy</a> of <a href="#">Kingdom</a> . . . . .	47
<a href="#">FoodFac</a>	<a href="#">Factory</a> class to serve as a concrete creator class used to create Food supplies . . . . .	49
<a href="#">FoodSupp</a>	Concrete Product <a href="#">Class</a> containing the information on the Food supplies . . . . .	50

<a href="#">FoodWagon</a>	
Concrete Prototype class to serve as the wagon for Food	51
<a href="#">HealthyState</a>	
A concrete state class. A class that is one of the alternative concrete states for <a href="#">Economy</a> of <a href="#">Kingdom</a>	53
<a href="#">Historian</a>	54
<a href="#">History</a>	56
<a href="#">HistoryBook</a>	58
<a href="#">Iterator</a>	
<a href="#">Iterator</a> class Defines an interface for accessing and traversing bannerman elements in ground↔	
Forces	59
<a href="#">Kingdom</a>	
A <a href="#">Kingdom</a> class that has all the fighting <a href="#">Bannerman</a> and <a href="#">Economy</a>	61
<a href="#">Location</a>	
Concrete object. This is the class/object that will be decorated. Inherits from <a href="#">WarTheatre</a> class	64
<a href="#">MasterOfCoin</a>	
The Concrete class for the Mediator A class that allows for other classes to talk to when key changes are made	65
<a href="#">MedicalFac</a>	
<a href="#">Factory</a> class to serve as a concrete creator class used to create Medical supplies	67
<a href="#">MedicalSupp</a>	
Concrete Product <a href="#">Class</a> containing the information on the Medical supplies	68
<a href="#">MedicalWagon</a>	
Concrete Prototype class to serve as the wagon for Medical	69
<a href="#">Raven</a>	
The Abstract class for the Observer A class that the Concrete Observer class inherits from	71
<a href="#">sendRaven</a>	
The Concrete class for the Observer A class that watches the <a href="#">Bannerman</a> for any changes that are made	72
<a href="#">Siege</a>	
A concrete strategy class. A class that provides an alternative strategy for how <a href="#">Bannerman</a> will fight enemyBannerman	74
<a href="#">State</a>	
An Abstract state class. A class that provides an interface to the alternative concrete states for <a href="#">Economy</a> of <a href="#">Kingdom</a>	76
<a href="#">Strategy</a>	
An Abstract strategy class. A class that provides an interface to the alternative concrete strategies for how <a href="#">Bannerman</a> will fight enemyBannerman	79
<a href="#">SupplyWagon</a>	
Abstract Prototype class <a href="#">SupplyWagon</a> to serve as the base class for all derived concrete prototypes	83
<a href="#">Topology</a>	
Concrete decorator B. This is class that decorates the topology of the venue. Inherits from <a href="#">Conditions</a> class	85
<a href="#">Treasury</a>	86
<a href="#">Troop</a>	
A class that defines the primitive objects of the <a href="#">Bannerman</a> composition	87
<a href="#">UnstableState</a>	
A concrete state class. A class that is one of the alternative concrete states for <a href="#">Economy</a> of <a href="#">Kingdom</a>	97
<a href="#">WarIndicators</a>	
An interface class for all classes that communicate with each other through <a href="#">Treasury</a> mediator	99
<a href="#">WarTheatre</a>	
Abstract object. This is the Component participant in the decorator pattern	101
<a href="#">WeaponsFac</a>	
<a href="#">Factory</a> class to serve as a concrete creator class used to create Weapons supplies	104
<a href="#">WeaponSupp</a>	
Concrete Product <a href="#">Class</a> containing the information on the Weapon supplies	105

---

<a href="#">WeaponWagon</a>	
Concrete Prototype class to serve as the wagon for Weapons . . . . .	106
<a href="#">Weather</a>	
Concrete decorator A. This is the class implements climate effects to the war location. Inherits from <a href="#">Conditions</a> class . . . . .	108



## Chapter 3

# File Index

### 3.1 File List

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

<a href="#">Ambush.h</a>	??
<a href="#">ArmySupplies.h</a>	??
<a href="#">Assassinate.h</a>	??
<a href="#">Bannerman.h</a>	??
<a href="#">BattleField.h</a>	??
<a href="#">Class.h</a>	??
<a href="#">Commander.h</a>	??
<a href="#">Conditions.h</a>	??
<a href="#">Conlterator.h</a>	??
<a href="#">Economy.h</a>	??
<a href="#">Factory.h</a>	??
<a href="#">FailedState.h</a>	??
<a href="#">FoodFac.h</a>	??
<a href="#">FoodSupp.h</a>	??
<a href="#">FoodWagon.h</a>	??
<a href="#">HealthyState.h</a>	??
<a href="#">Historian.h</a>	??
<a href="#">History.h</a>	??
<a href="#">HistoryBook.h</a>	??
<a href="#">Iterator.h</a>	??
<a href="#">Kingdom.h</a>	??
<a href="#">Location.h</a>	??
<a href="#">MasterOfCoin.h</a>	??
<a href="#">MedicalFac.h</a>	??
<a href="#">MedicalSupp.h</a>	??
<a href="#">MedicalWagon.h</a>	??
<a href="#">Raven.h</a>	??
<a href="#">sendRaven.h</a>	??
<a href="#">Siege.h</a>	??
<a href="#">State.h</a>	??
<a href="#">Strategy.h</a>	??
<a href="#">SupplyWagon.h</a>	??
<a href="#">Topology.h</a>	??
<a href="#">Treasury.h</a>	??
<a href="#">Troop.h</a>	??

<a href="#">UnstableState.h</a>	??
<a href="#">WarIndicators.h</a>	??
<a href="#">WarTheatre.h</a>	??
<a href="#">WeaponsFac.h</a>	??
<a href="#">WeaponSupp.h</a>	??
<a href="#">WeaponWagon.h</a>	??
<a href="#">Weather.h</a>	??

## Chapter 4

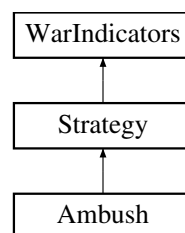
# Class Documentation

### 4.1 Ambush Class Reference

A concrete strategy class. A class that provides an alternative strategy for how [Bannerman](#) will fight enemy↔ Bannerman.

```
#include <Ambush.h>
```

Inheritance diagram for Ambush:



#### Public Member Functions

- [Ambush](#) (int [stealth](#), [Kingdom](#) \*[myKingdom](#), [Kingdom](#) \*[enemyKingdom](#), [Bannerman](#) \*[myBannerman](#), [Bannerman](#) \*[enemyBannerman](#), string name, int min, int [minFavour](#), [Historian](#) \*[h](#), [HistoryBook](#) \*[hb](#))  
*constructor. initializes stealth and calls base class constructor to initialize myKingdom, enemyKingdom, my↔ Bannerman, enemyBannerman, strategy, min and minFavour.*
- bool [attack](#) ([Bannerman](#) \*[myBannerman](#), [Bannerman](#) \*[enemyBannerman](#))  
*makes two bannerman from enemy kingdoms fight each other until one loses*
- ~[Ambush](#) ()  
*destructor.*

#### Private Attributes

- int [stealth](#)

## Additional Inherited Members

### 4.1.1 Detailed Description

A concrete strategy class. A class that provides an alternative strategy for how [Bannerman](#) will fight enemy↵ Bannerman.

#### Author

Morgan Bentley

#### Date

October 2022

### 4.1.2 Constructor & Destructor Documentation

#### 4.1.2.1 Ambush()

```
Ambush::Ambush (
    int stealth,
    Kingdom * myKingdom,
    Kingdom * enemyKingdom,
    Bannerman * myBannerman,
    Bannerman * enemyBannerman,
    string name,
    int min,
    int minFavour,
    Historian * h,
    HistoryBook * hb )
```

constructor. initializes stealth and calls base class constructor to initialize myKingdom, enemyKingdom, my↵ Bannerman, enemyBannerman, strategy, min and minFavour.

#### Parameters

<i>stealth</i>	- skill level of <a href="#">Bannerman</a> .
<i>myKingdom</i>	- <a href="#">Kingdom</a> pointer of attacking <a href="#">Bannerman's Kingdom</a> .
<i>enemyKingdom</i>	- <a href="#">Kingdom</a> pointer of enemyBannerman's <a href="#">Kingdom</a> .
<i>myBannerman</i>	- attacking <a href="#">Bannerman</a> .
<i>enemyBannerman</i>	- defending <a href="#">Bannerman</a> .
<i>name</i>	- name of concrete strategy.
<i>min</i>	- minimum supplies for food, weapons and medicine
<i>minFavour</i>	- minimum favour below which bannerman change allegiances
<i>h</i>	- <a href="#">Historian</a> (originator) to save defected bannemen.
<i>hb</i>	- Caretaker for memento implementation.



### 4.1.3 Member Function Documentation

#### 4.1.3.1 attack()

```
bool Ambush::attack (
    Bannerman * myBannerman,
    Bannerman * enemyBannerman ) [virtual]
```

makes two bannerman from enemy kingdoms fight each other until one loses

##### Parameters

<i>myBannerman</i>	- attacking bannerman object.
<i>enemyBannerman</i>	- <a href="#">Bannerman</a> object being attacked.

##### Returns

battle result as a boolean with true implying the attacking bannerman object won and false implying the opposite

Implements [Strategy](#).

### 4.1.4 Member Data Documentation

#### 4.1.4.1 stealth

```
int Ambush::stealth [private]
```

skill level of [Bannerman](#) using this strategy

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

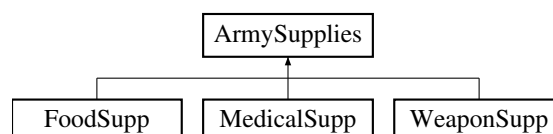
- Ambush.h
- Ambush.cpp

## 4.2 ArmySupplies Class Reference

Abstract [Class](#) containing the information on the supplies.

```
#include <ArmySupplies.h>
```

Inheritance diagram for ArmySupplies:



## Public Member Functions

- virtual int [getAmount](#) ()=0  
*[getAmount\(\)](#) is used to return the amount of supplies.*

### 4.2.1 Detailed Description

Abstract [Class](#) containing the information on the supplies.

Author

Ronin Brookes 19069686

### 4.2.2 Member Function Documentation

#### 4.2.2.1 [getAmount\(\)](#)

```
virtual int ArmySupplies::getAmount ( ) [pure virtual]
```

[getAmount\(\)](#) is used to return the amount of supplies.

Returns

an integer representation of the amount of supplies.

Implemented in [FoodSupp](#), [MedicalSupp](#), and [WeaponSupp](#).

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

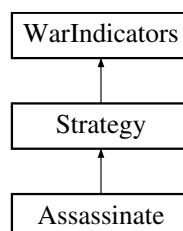
- [ArmySupplies.h](#)

## 4.3 Assassinate Class Reference

A concrete strategy class. A class that provides an alternative strategy for how [Bannerman](#) will fight enemy↵  
Bannerman.

```
#include <Assassinate.h>
```

Inheritance diagram for Assassinate:



## Public Member Functions

- `Assassinate` (int `stealth`, bool `alive`, Kingdom `*myKingdom`, Kingdom `*enemyKingdom`, Bannerman `*myBannerman`, Bannerman `*enemyBannerman`, string `name`, int `min`, int `minFavour`, Historian `*h`, HistoryBook `*hb`)  
*constructor. initializes stealth and calls base class constructor to initialize myKingdom, enemyKingdom, myBannerman, enemyBannerman, strategy, min and minFavour.*
- bool `attack` (Bannerman `*myBannerman`, Bannerman `*enemyBannerman`)  
*makes assassin of attacking Bannerman try to kill enemyBannerman*
- `~Assassinate` ()  
*destructor.*

## Private Attributes

- int `stealth`
- bool `alive`

## Additional Inherited Members

### 4.3.1 Detailed Description

A concrete strategy class. A class that provides an alternative strategy for how Bannerman will fight enemy Bannerman.

#### Author

Morgan Bentley

#### Date

October 2022

### 4.3.2 Constructor & Destructor Documentation

#### 4.3.2.1 Assassinate()

```
Assassinate::Assassinate (
    int stealth,
    bool alive,
    Kingdom * myKingdom,
    Kingdom * enemyKingdom,
    Bannerman * myBannerman,
    Bannerman * enemyBannerman,
    string name,
    int min,
    int minFavour,
    Historian * h,
    HistoryBook * hb )
```

constructor. initializes stealth and calls base class constructor to initialize myKingdom, enemyKingdom, myBannerman, enemyBannerman, strategy, min and minFavour.

## Parameters

<i>stealth</i>	- skill level of <a href="#">Bannerman</a> 's assassin.
<i>alive</i>	-life state of <a href="#">Bannerman</a> 's assassin, is false if assassin was killed.
<i>myKingdom</i>	- <a href="#">Kingdom</a> pointer of attacking <a href="#">Bannerman</a> 's <a href="#">Kingdom</a> .
<i>enemyKingdom</i>	- <a href="#">Kingdom</a> pointer of enemyBannerman's <a href="#">Kingdom</a> .
<i>myBannerman</i>	- attacking <a href="#">Bannerman</a> .
<i>enemyBannerman</i>	- defending <a href="#">Bannerman</a> .
<i>name</i>	- name of concrete strategy.
<i>min</i>	- minimum supplies for food,weapons and medicine
<i>minFavour</i>	- minimum favour below which bannerman change allegiances
<i>h</i>	- <a href="#">Historian</a> (originator) to save defected bannemen.
<i>hb</i>	- Caretaker for memento implementation.

### 4.3.3 Member Function Documentation

#### 4.3.3.1 attack()

```
bool Assassinate::attack (
    Bannerman * myBannerman,
    Bannerman * enemyBannerman ) [virtual]
```

makes assassin of attacking [Bannerman](#) try to kill enemyBannerman

## Parameters

<i>myBannerman</i>	- attacking <a href="#">Bannerman</a> object.
<i>enemyBannerman</i>	- <a href="#">Bannerman</a> object being attacked.

## Returns

battle result as a boolean with true implying the attacking [Bannerman](#) object won and false implying the opposite

Implements [Strategy](#).

### 4.3.4 Member Data Documentation

#### 4.3.4.1 alive

```
bool Assassinate::alive [private]
```

life state of [Bannerman](#)'s assassin, is false if assassin was killed

## 4.3.4.2 stealth

```
int Assassinate::stealth [private]
```

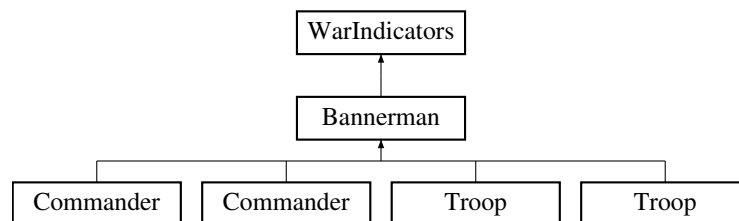
skill level of Assassin from [Bannerman](#) using this strategy

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

- Assassinate.h
- Assassinate.cpp

## 4.4 Bannerman Class Reference

Inheritance diagram for Bannerman:



## Public Member Functions

- **Bannerman ()**  
*Default constructor.*
- virtual void [increaseFavour](#) ()=0  
*Abstract. increases the favour of the bannerman.*
- virtual void [decreaseFavour](#) ()=0  
*Abstract. decreases the favour of the bannerman.*
- virtual void [attach](#) ([Raven](#) \*o)=0  
*Abstract. Attaches a [Raven](#) Observer on the bannerman.*
- virtual void [detach](#) ([Raven](#) \*o)=0  
*Abstract. detaches a [Raven](#) Observer from the bannerman.*
- virtual void [increaseHP](#) (int boost)=0  
*Abstract. increases the health points(HP) of the bannerman.*
- virtual void [changeStrategy](#) ([Strategy](#) \*strategy)=0  
*Abstract. changes the attack strategy of the bannerman.*
- virtual void [attack](#) ([Bannerman](#) \*myBannerman, [Bannerman](#) \*enemyBannerman)=0  
*Abstract. uses strategy to attack another kingdom.*
- virtual void [increasePower](#) (int boost)=0  
*Abstract. increases the damage capability of the bannerman.*
- virtual string [getName](#) ()=0  
*Accessor that returns the name of the bannerman.*
- virtual int [getHP](#) ()=0  
*Abstract. Accessor that returns the name of the component.*
- virtual int [getDamage](#) ()=0

- Abstract. Accessor that returns the damage capability of the bannerman.*

  - virtual void `receiveDamage` (int boost)=0
- Abstract. Increases the damage capability of the bannerman.*

  - virtual void `decreasePower` (int x)=0
- Abstract. decreases the damage capability of the bannerman.*

  - virtual void `decreaseWeapons` ()=0
- Abstract. Decreases the number of weapons that the bannerman has.*

  - virtual void `decreaseFood` ()=0
- Abstract. Decreases the number of food supplies that the bannerman has.*

  - virtual void `decreaseMedical` ()=0
- Abstract. Decreases the number of medical supplies that the bannerman has.*

  - virtual int `getWeapons` ()=0
- Abstract. Accessor that returns the number of weapons of the bannerman.*

  - virtual int `getFood` ()=0
- Abstract. Accessor that returns number of food supplies that the bannerman has.*

  - virtual int `getMedical` ()=0
- Abstract. Accessor that returns the number of medical supplies that the bannerman has.*

  - virtual void `setWeapons` (int numWeapons)=0
- Abstract. Sets the number of weapons of the bannerman.*

  - virtual void `setFood` (int numFood)=0
- Abstract. Sets the number of food supplies that the bannerman has.*

  - virtual void `setMedical` (int numMedical)=0
- Abstract. Sets the number of medical supplies that the bannerman has.*

  - virtual void `setRaven` (list< `Raven` \* > r)=0
- Abstract. Assigns a list of `Raven` Observers to the bannerman's ravenList.*

  - virtual void `setMaster` (`MasterOfCoin` \*m)=0
- Abstract. Assigns a MasterofCoin mediator to ensure that the army has the supplies it needs.*

  - virtual int `getFavour` ()=0
- Abstract. returns the favour of the bannerman.*

  - `~Bannerman` ()
- Default destructor.*

  - virtual list< `Bannerman` \* > `getTroops` ()=0
- Abstract. gets the list of troops under the commander.*

## Public Attributes

- bool `assassin`

## Protected Attributes

- string `name`
- int `favour`
- int `numWeapons`
- int `damage`
- int `numFood`
- int `numMedical`
- list< `Raven` \* > `ravenList`
- `MasterOfCoin` \* m
- `Strategy` \* strategy
- int `HP`

## 4.4.1 Member Function Documentation

### 4.4.1.1 attach()

```
virtual void Bannerman::attach (
    Raven * o ) [pure virtual]
```

Abstract. Attaches a [Raven](#) Observer on the bannerman.

#### Parameters

<i>o</i>	- the <a href="#">Raven</a> Observer to attach.
----------	---

Implemented in [Commander](#), and [Troop](#).

### 4.4.1.2 attack()

```
virtual void Bannerman::attack (
    Bannerman * myBannerman,
    Bannerman * enemyBannerman ) [pure virtual]
```

Abstract. uses strategy to attack another kingdom.

#### Parameters

<i>myBannerman</i>	- The attacking bannerman object.
<i>enemyBannerman</i>	- The bannerman object being attacked.

Implemented in [Commander](#), and [Troop](#).

### 4.4.1.3 changeStrategy()

```
virtual void Bannerman::changeStrategy (
    Strategy * strategy ) [pure virtual]
```

Abstract. changes the attack strategy of the bannerman.

#### Parameters

<i>strategy</i>	- the new attack strategy.
-----------------	----------------------------

Implemented in [Commander](#), and [Troop](#).

#### 4.4.1.4 decreaseFavour()

```
virtual void Bannerman::decreaseFavour ( ) [pure virtual]
```

Abstract. decreases the favour of the bannerman.

Implemented in [Commander](#), and [Troop](#).

#### 4.4.1.5 decreaseFood()

```
virtual void Bannerman::decreaseFood ( ) [pure virtual]
```

Abstract. Decreases the number of food supplies that the bannerman has.

Implemented in [Commander](#), and [Troop](#).

#### 4.4.1.6 decreaseMedical()

```
virtual void Bannerman::decreaseMedical ( ) [pure virtual]
```

Abstract. Decreases the number of medical supplies that the bannerman has.

Implemented in [Commander](#), and [Troop](#).

#### 4.4.1.7 decreasePower()

```
virtual void Bannerman::decreasePower (
    int x ) [pure virtual]
```

Abstract. decreases the damage capability of the bannerman.

##### Parameters

x	- the number by which to decrease damage.
---	---

Implemented in [Commander](#), and [Troop](#).



#### 4.4.1.8 decreaseWeapons()

```
virtual void Bannerman::decreaseWeapons ( ) [pure virtual]
```

Abstract. Decreases the number of weapons that the bannerman has.

Implemented in [Commander](#), and [Troop](#).

#### 4.4.1.9 detach()

```
virtual void Bannerman::detach (
    Raven * o ) [pure virtual]
```

Abstract. detaches a [Raven](#) Observer from the bannerman.

##### Parameters

<i>o</i>	- the <a href="#">Raven</a> Observer to detach.
----------	---

Implemented in [Commander](#), and [Troop](#).

#### 4.4.1.10 getDamage()

```
virtual int Bannerman::getDamage ( ) [pure virtual]
```

Abstract. Accessor that returns the damage capability of the bannerman.

##### Returns

The damage of the bannerman.

Implemented in [Commander](#), and [Troop](#).

#### 4.4.1.11 getFavour()

```
virtual int Bannerman::getFavour ( ) [pure virtual]
```

Abstract. returns the favour of the bannerman.

Implemented in [Commander](#), and [Troop](#).

#### 4.4.1.12 getFood()

```
virtual int Bannerman::getFood ( ) [pure virtual]
```

Abstract. Accessor that returns number of food supplies that the bannerman has.

##### Returns

The numFood of the bannerman.

Implemented in [Commander](#), and [Troop](#).

#### 4.4.1.13 getHP()

```
virtual int Bannerman::getHP ( ) [pure virtual]
```

Abstract. Accessor that returns the name of the component.

##### Returns

The name of the bannerman.

Implemented in [Commander](#), and [Troop](#).

#### 4.4.1.14 getMedical()

```
virtual int Bannerman::getMedical ( ) [pure virtual]
```

Abstract. Accessor that returns the number of medical supplies that the bannerman has.

##### Returns

The numMedical of the bannerman.

Implemented in [Commander](#), and [Troop](#).

#### 4.4.1.15 getName()

```
virtual string Bannerman::getName ( ) [pure virtual]
```

Accessor that returns the name of the bannerman.

##### Returns

The name of the bannerman.

Implemented in [Commander](#), and [Troop](#).

#### 4.4.1.16 getTroops()

```
virtual list< Bannerman * > Bannerman::getTroops ( ) [pure virtual]
```

Abstract. gets the list of troops under the commander.

##### Returns

the list of troops variable

Implemented in [Commander](#), and [Troop](#).

#### 4.4.1.17 getWeapons()

```
virtual int Bannerman::getWeapons ( ) [pure virtual]
```

Abstract. Accessor that returns the number of weapons of the bannerman.

##### Returns

The numWeapons of the bannerman.

Implemented in [Commander](#), and [Troop](#).

#### 4.4.1.18 increaseFavour()

```
virtual void Bannerman::increaseFavour ( ) [pure virtual]
```

Abstract. increases the favour of the bannerman.

Implemented in [Commander](#), and [Troop](#).

#### 4.4.1.19 increaseHP()

```
virtual void Bannerman::increaseHP (
    int boost ) [pure virtual]
```

Abstract. increases the health points(HP) of the bannerman.

##### Parameters

<i>boost</i>	- the number by which to increase HP.
--------------	---------------------------------------

Implemented in [Commander](#), and [Troop](#).

#### 4.4.1.20 increasePower()

```
virtual void Bannerman::increasePower (
    int boost ) [pure virtual]
```

Abstract. increases the damage capability of the bannerman.

##### Parameters

<i>boost</i>	- the number by which to increase damage.
--------------	---

Implemented in [Commander](#), and [Troop](#).

#### 4.4.1.21 receiveDamage()

```
virtual void Bannerman::receiveDamage (
    int boost ) [pure virtual]
```

Abstract. Increases the damage capability of the bannerman.

##### Parameters

<i>boost</i>	- the number by which to increase damage
--------------	--

Implemented in [Commander](#), and [Troop](#).

#### 4.4.1.22 setFood()

```
virtual void Bannerman::setFood (
    int numFood ) [pure virtual]
```

Abstract. Sets the number of food supplies that the bannerman has.

##### Parameters

<i>numFood</i>	- The new numFood the component should have
----------------	---

Implemented in [Commander](#), and [Troop](#).

#### 4.4.1.23 setMaster()

```
virtual void Bannerman::setMaster (
    MasterOfCoin * m ) [pure virtual]
```

Abstract. Assigns a MasterOfCoin mediator to ensure that the army has the supplies it needs.

##### Parameters

<i>m</i>	- the new <a href="#">MasterOfCoin</a> mediator.
----------	--

Implemented in [Commander](#), and [Troop](#).

#### 4.4.1.24 setMedical()

```
virtual void Bannerman::setMedical (
    int numMedical ) [pure virtual]
```

Abstract. Sets the number of medical supplies that the bannerman has.

##### Parameters

<i>numMedical</i>	- The new numMedical the component should have
-------------------	--

Implemented in [Commander](#), and [Troop](#).

#### 4.4.1.25 setRaven()

```
virtual void Bannerman::setRaven (
    list< Raven * > r ) [pure virtual]
```

Abstract. Assigns a list of [Raven](#) Observers to the bannerman's ravenList.

##### Parameters

<i>r</i>	- the list of <a href="#">Raven</a> Observer to attach.
----------	---

Implemented in [Commander](#), and [Troop](#).

#### 4.4.1.26 setWeapons()

```
virtual void Bannerman::setWeapons (
    int numWeapons ) [pure virtual]
```

Abstract. Sets the number of weapons of the bannerman.

## Parameters

<code>numWeapons</code>	- The new name the numWeapons should have
-------------------------	---

Implemented in [Commander](#), and [Troop](#).

## 4.4.2 Member Data Documentation

### 4.4.2.1 assassin

```
bool Bannerman::assassin
```

Indicates whether or not the bannerman is an assassin

### 4.4.2.2 damage

```
int Bannerman::damage [protected]
```

Damage capability the bannerman has

### 4.4.2.3 favour

```
int Bannerman::favour [protected]
```

Amount of favour the bannerman has

### 4.4.2.4 HP

```
int Bannerman::HP [protected]
```

The health points of the bannerman

### 4.4.2.5 m

```
MasterOfCoin* Bannerman::m [protected]
```

Mediator for ensuring that the bannerman/army has the supplies it needs.

### 4.4.2.6 name

```
string Bannerman::name [protected]
```

Name of the bannerman

#### 4.4.2.7 numFood

```
int Bannerman::numFood [protected]
```

Amount of food the bannerman has

#### 4.4.2.8 numMedical

```
int Bannerman::numMedical [protected]
```

Amount of medical supplies the bannerman has

#### 4.4.2.9 numWeapons

```
int Bannerman::numWeapons [protected]
```

Number of weapons the bannerman has

#### 4.4.2.10 ravenList

```
list<Raven*> Bannerman::ravenList [protected]
```

A list of [Raven](#) observers that have been attached to the bannerman object

#### 4.4.2.11 strategy

```
Strategy* Bannerman::strategy [protected]
```

The attack strategy the bannerman uses

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

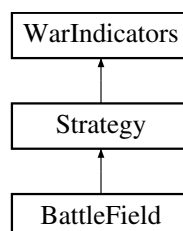
- Bannerman.h
- Bannerman.cpp

## 4.5 BattleField Class Reference

A concrete strategy class. A class that provides an alternative strategy for how [Bannerman](#) will fight enemy↵  
Bannerman.

```
#include <BattleField.h>
```

Inheritance diagram for BattleField:





## Public Member Functions

- [BattleField](#) ([Kingdom](#) \*myKingdom, [Kingdom](#) \*enemyKingdom, [Bannerman](#) \*myBannerman, [Bannerman](#) \*enemyBannerman, string name, int min, int minFavour, [Historian](#) \*h, [HistoryBook](#) \*hb)  
*constructor. initializes stealth and calls base class constructor to initialize myKingdom, enemyKingdom, myBannerman, enemyBannerman, strategy, min and minFavour.*
- bool [attack](#) ([Bannerman](#) \*myBannerman, [Bannerman](#) \*enemyBannerman)  
*makes two bannerman from enemy kingdoms fight each other until one loses*
- ~[BattleField](#) ()  
*destructor.*

## Additional Inherited Members

### 4.5.1 Detailed Description

A concrete strategy class. A class that provides an alternative strategy for how [Bannerman](#) will fight enemyBannerman.

#### Author

Morgan Bentley

#### Date

October 2022

### 4.5.2 Constructor & Destructor Documentation

#### 4.5.2.1 BattleField()

```
BattleField::BattleField (
    Kingdom * myKingdom,
    Kingdom * enemyKingdom,
    Bannerman * myBannerman,
    Bannerman * enemyBannerman,
    string name,
    int min,
    int minFavour,
    Historian * h,
    HistoryBook * hb )
```

constructor. initializes stealth and calls base class constructor to initialize myKingdom, enemyKingdom, myBannerman, enemyBannerman, strategy, min and minFavour.

#### Parameters

<i>myKingdom</i>	- <a href="#">Kingdom</a> pointer of attacking <a href="#">Bannerman</a> 's <a href="#">Kingdom</a> .
<i>enemyKingdom</i>	- <a href="#">Kingdom</a> pointer of enemyBannerman's <a href="#">Kingdom</a> .
<i>myBannerman</i>	- attacking <a href="#">Bannerman</a> .
<i>enemyBannerman</i>	- defending <a href="#">Bannerman</a> .
<i>name</i>	- name of concrete strategy.
<i>min</i>	- minimum supplies for food, weapons and medicine
<i>minFavour</i>	- minimum favour below which bannerman changes allegiance

### 4.5.3 Member Function Documentation

#### 4.5.3.1 attack()

```
bool Battlefield::attack (
    Bannerman * myBannerman,
    Bannerman * enemyBannerman ) [virtual]
```

makes two bannerman from enemy kingdoms fight each other until one loses

#### Parameters

<i>myBannerman</i>	- attacking bannerman object.
<i>enemyBannerman</i>	- <a href="#">Bannerman</a> object being attacked.

#### Returns

battle result as a boolean with true implying the attacking bannerman object won and false implying the opposite

Implements [Strategy](#).

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

- Battlefield.h
- Battlefield.cpp

## 4.6 Class Class Reference

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

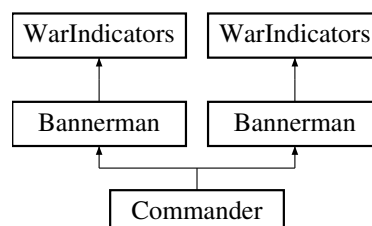
- Class.h

## 4.7 Commander Class Reference

A class that acts as a container/composite for bannerman objects as well as performing operations on and using various bannerman objects.

```
#include <Commander.h>
```

Inheritance diagram for Commander:



## Public Member Functions

- **Commander** (string *name*)  
*Commander* Constructor which takes in the name of a commander.
- **Iterator** \* **createIterator** ()  
*creates a conlterator object, which is a means to traverse groundForces sequentially.*
- void **removeBannerman** (**Bannerman** \*x)  
*removes a bannerman object from the groundforces list.*
- void **attack** (**Bannerman** \*myBannerman, **Bannerman** \*enemyBannerman) override  
*uses strategy to make all troops in groundForces to attack another kingdom.*
- void **addBannerman** (**Bannerman** \*b)  
*adds a bannerman object to the groundforces list.*
- list< **Bannerman** \* > **getTroops** ()
- int **getHP** () override  
*Accessor. Returns the HP.*
- int **getDamage** () override  
*Accessor. Returns the damage.*
- void **attach** (**Raven** \*o) override  
*Attaches a **Raven** observer to all the bannerman objects in the groundForces list.*
- void **detach** (**Raven** \*o) override  
*Detaches a **Raven** observer from all the bannerman objects in the groundForces list.*
- void **receiveDamage** (int x) override  
*increases the damage variables of the groundForces by x*
- void **decreaseWeapons** () override  
*decreases the numWeapons variables of the bannerman objects in groundForces by 1.*
- void **decreaseFood** () override  
*decreases the numFood variables of the bannerman objects in groundForces by 1.*
- void **decreaseMedical** () override  
*decreases the numMedical variables of the bannerman objects in groundForces by 1.*
- void **increasePower** (int boost) override  
*increases the damage variables of the bannerman objects in groundForces.*
- int **getWeapons** () override  
*Accessor. Returns numWeapons.*
- int **getFood** () override  
*Accessor. Returns the total groundForces amount of food.*
- int **getMedical** () override  
*Accessor. Returns the total groundForces amount of medical supplies.*
- void **increaseHP** (int boost) override  
*increases the HP variables of the bannerman objects in groundForces.*
- void **changeStrategy** (**Strategy** \*strategy) override  
*Changes the attack strategy variable of the bannerman objects in groundForces.*
- void **increaseFavour** () override  
*Increases the loyalty favour levels variable of the bannerman objects in groundForces by 1.*
- void **decreaseFavour** () override  
*Decreases the loyalty favour levels variable of the bannerman objects in groundForces by 1.*
- int **getFavour** () override  
*Returns the sum of the loyalty favour levels variable of the bannerman objects in groundForce.*
- void **setWeapons** (int numWeapons) override  
*Sets the number of weapon supplies of the bannerman objects in groundForces.*
- void **setFood** (int numFood) override  
*Sets the number of food supplies of the bannerman objects in groundForces.*

- void `setMedical` (int `numMedical`) override  
*Sets the number of medical supplies of the bannerman objects in groundForces.*
- void `decreasePower` (int x) override  
*Decreases the damage capability of all the bannerman objects in groundForces.*
- `~Commander` ()  
*Default destructor.*
- void `setRaven` (list< `Raven` \* > r) override  
*Assigns a `Raven` observer list to all the bannerman objects in the groundForces list.*
- void `setMaster` (`MasterOfCoin` \*m) override  
*Assigns a MasterofCoin mediator all the bannerman objects in the groundForces list to ensure that the army has the supplies it needs.*
- void `setStrategy` (`Strategy` \*s)  
*Sets the attack strategy variable of the bannerman objects in groundForces.*
- string `getName` ()  
*Returns the name of this commander object.*

## Private Attributes

- list< `Bannerman` \* > `groundForces`  
*list of bannerman components.*
- string `name`  
*name of the bannerman as a whole, represented by the commander speaking for the bannerman*

## Additional Inherited Members

### 4.7.1 Detailed Description

A class that acts as a container/composite for bannerman objects as well as performing operations on and using various bannerman objects.

#### Author

Thapelo Thoka

#### Date

October 2022

### 4.7.2 Constructor & Destructor Documentation

#### 4.7.2.1 Commander()

```
Commander::Commander (
    string name )
```

`Commander` Constructor which takes in the name of a commander.

## Parameters

<i>name</i>	- name of commander
-------------	---------------------

### 4.7.3 Member Function Documentation

#### 4.7.3.1 attach()

```
void Commander::attach (  
    Raven * o ) [override], [virtual]
```

Attaches a [Raven](#) observer to all the bannerman objects in the groundForces list.

## Parameters

<i>o</i>	- the <a href="#">Raven</a> observer object
----------	---

Implements [Bannerman](#).

#### 4.7.3.2 attack()

```
void Commander::attack (  
    Bannerman * myBannerman,  
    Bannerman * enemyBannerman ) [override], [virtual]
```

uses strategy to make all troops in groundForces to attack another kingdom.

## Parameters

<i>myBannerman</i>	- The attacking bannerman object.
<i>enemyBannerman</i>	- The bannerman object being attacked.

Implements [Bannerman](#).

#### 4.7.3.3 changeStrategy()

```
void Commander::changeStrategy (  
    Strategy * strategy ) [override], [virtual]
```

Changes the attack strategy variable of the bannerman objects in groundForces.

**Parameters**

<i>strategy</i>	- The new strategy bannerman objects in groundForces should have.
-----------------	---

Implements [Bannerman](#).

**4.7.3.4 decreaseFavour()**

```
void Commander::decreaseFavour ( ) [override], [virtual]
```

Decreases the loyalty favour levels variable of the bannerman objects in groundForces by 1.

Implements [Bannerman](#).

**4.7.3.5 decreaseFood()**

```
void Commander::decreaseFood ( ) [override], [virtual]
```

decreases the numFood variables of the bannerman objects in groundForces by 1.

Implements [Bannerman](#).

**4.7.3.6 decreaseMedical()**

```
void Commander::decreaseMedical ( ) [override], [virtual]
```

decreases the numMedical variables of the bannerman objects in groundForces by 1.

Implements [Bannerman](#).

**4.7.3.7 decreasePower()**

```
void Commander::decreasePower (
    int x ) [override], [virtual]
```

Decreases the damage capability of all the bannerman objects in groundForces.

**Parameters**

<i>x</i>	- the number by which to decrease damage.
----------	---

Implements [Bannerman](#).

#### 4.7.3.8 decreaseWeapons()

```
void Commander::decreaseWeapons ( ) [override], [virtual]
```

decreases the numWeapons variables of the bannerman objects in groundForces by 1.

Implements [Bannerman](#).

#### 4.7.3.9 detach()

```
void Commander::detach (
    Raven * o ) [override], [virtual]
```

Detaches a [Raven](#) observer from all the bannerman objects in the groundForces list.

##### Parameters

<i>o</i>	- the <a href="#">Raven</a> observer object
----------	---

Implements [Bannerman](#).

#### 4.7.3.10 getDamage()

```
int Commander::getDamage ( ) [override], [virtual]
```

Accessor. Returns the damage.

##### Returns

The total groundForces damage.

Implements [Bannerman](#).

#### 4.7.3.11 getFavour()

```
int Commander::getFavour ( ) [override], [virtual]
```

Returns the sum of the loyalty favour levels variable of the bannerman objects in groundForce.

##### Returns

favour

Implements [Bannerman](#).

#### 4.7.3.12 getFood()

```
int Commander::getFood ( ) [override], [virtual]
```

Accessor. Returns the total groundForces amount of food.

##### Returns

numFood.

Implements [Bannerman](#).

#### 4.7.3.13 getHP()

```
int Commander::getHP ( ) [override], [virtual]
```

Accessor. Returns the HP.

##### Returns

The total groundForces HP.

Implements [Bannerman](#).

#### 4.7.3.14 getMedical()

```
int Commander::getMedical ( ) [override], [virtual]
```

Accessor. Returns the total groundForces amount of medical supplies.

##### Returns

numMedical.

Implements [Bannerman](#).

#### 4.7.3.15 getName()

```
string Commander::getName ( ) [virtual]
```

Returns the name of this commander object.

##### Returns

name

Implements [Bannerman](#).



#### 4.7.3.16 getTroops()

```
list< Bannerman * > Commander::getTroops ( ) [virtual]
```

Returns the groundForces bannerman list

Implements [Bannerman](#).

#### 4.7.3.17 getWeapons()

```
int Commander::getWeapons ( ) [override], [virtual]
```

Accessor. Returns numWeapons.

##### Returns

The total groundForces number of Weapons.

Implements [Bannerman](#).

#### 4.7.3.18 increaseFavour()

```
void Commander::increaseFavour ( ) [override], [virtual]
```

Increases the loyalty favour levels variable of the bannerman objects in groundForces by 1.

Implements [Bannerman](#).

#### 4.7.3.19 increaseHP()

```
void Commander::increaseHP (
    int boost ) [override], [virtual]
```

increases the HP variables of the bannerman objects in groundForces.

##### Parameters

<i>boost</i>	- The number by which to increase the HPs of the bannerman objects in groundForces.
--------------	---

Implements [Bannerman](#).

#### 4.7.3.20 increasePower()

```
void Commander::increasePower (
    int boost ) [override], [virtual]
```

increases the damage variables of the bannerman objects in groundForces.

##### Parameters

<i>boost</i>	- The number by which to increase the damage of the bannerman objects in groundForces.
--------------	--

Implements [Bannerman](#).

#### 4.7.3.21 receiveDamage()

```
void Commander::receiveDamage (
    int x ) [override], [virtual]
```

increases the damage variables of the groundForces by x

##### Parameters

<i>x</i>	- the number by which to increase the damage variables of the bannerman objects in groundForces.
----------	--

Implements [Bannerman](#).

#### 4.7.3.22 setFood()

```
void Commander::setFood (
    int numFood ) [override], [virtual]
```

Sets the number of food supplies of the bannerman objects in groundForces.

##### Parameters

<i>numFood</i>	- The new numFood bannerman objects in groundForces should have.
----------------	--

Implements [Bannerman](#).

#### 4.7.3.23 setMaster()

```
void Commander::setMaster (
    MasterOfCoin * m ) [override], [virtual]
```

Assigns a MasterofCoin mediator all the bannerman objects in the groundForces list to ensure that the army has the supplies it needs.

#### Parameters

<i>m</i>	- the new <a href="#">MasterOfCoin</a> mediator.
----------	--

Implements [Bannerman](#).

#### 4.7.3.24 setMedical()

```
void Commander::setMedical (
    int numMedical ) [override], [virtual]
```

Sets the number of medical supplies of the bannerman objects in groundForces.

#### Parameters

<i>numMedical</i>	- The new numMedical bannerman objects in groundForces should have.
-------------------	---

Implements [Bannerman](#).

#### 4.7.3.25 setRaven()

```
void Commander::setRaven (
    list< Raven * > r ) [override], [virtual]
```

Assigns a [Raven](#) observer list to all the bannerman objects in the groundForces list.

#### Parameters

<i>r</i>	- the <a href="#">Raven</a> observer list to attach
----------	---

Implements [Bannerman](#).

#### 4.7.3.26 setStrategy()

```
void Commander::setStrategy (
    Strategy * s )
```

Sets the attack strategy variable of the bannerman objects in groundForces.

## Parameters

<i>strategy</i>	- The new strategy bannerman objects in groundForces should have.
-----------------	---

**4.7.3.27 setWeapons()**

```
void Commander::setWeapons (
    int numWeapons ) [override], [virtual]
```

Sets the number of weapon supplies of the bannerman objects in groundForces.

## Parameters

<i>numWeapons</i>	- The new numWeapons bannerman objects in groundForces should have.
-------------------	---

Implements [Bannerman](#).

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

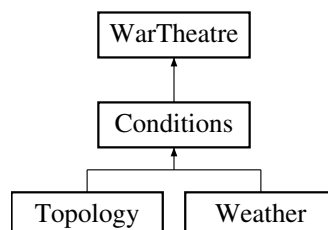
- Commander.h
- Commander.cpp

**4.8 Conditions Class Reference**

decorator class. This is the class/object decorates the concreteComponent. Inherits from [WarTheatre](#) class.

```
#include <Conditions.h>
```

Inheritance diagram for Conditions:

**Public Member Functions**

- void [sendScout](#) ()  
*decorator function for thr pattern*
- [Conditions](#) ([WarTheatre](#) \*myVenue)  
*constructor for the decorator class. merely used to assign variables and call the decorator function*
- virtual ~**Conditions** ()  
*destructor to delete the [WarTheatre](#) pointer*

## Private Attributes

- [WarTheatre](#) \* [myVenue](#)

### 4.8.1 Detailed Description

decorator class. This is the class/object decorates the concreteComponent. Inherits from [WarTheatre](#) class.

#### Author

Keabetswe Mothapo

#### Date

October 2022

### 4.8.2 Constructor & Destructor Documentation

#### 4.8.2.1 Conditions()

```
Conditions::Conditions (
    WarTheatre * myVenue )
```

constructor for the decorator class. merely used to assign variables and call the decorator function

#### Parameters

<i>myVenue</i>	- pointer to the object that is to be decoraed
----------------	--

### 4.8.3 Member Function Documentation

#### 4.8.3.1 sendScout()

```
void Conditions::sendScout ( ) [virtual]
```

decorator function for thr pattern

Implements [WarTheatre](#).

Reimplemented in [Topology](#), and [Weather](#).

## 4.8.4 Member Data Documentation

### 4.8.4.1 myVenue

`WarTheatre* Conditions::myVenue [private]`

Pointer to the decorated component

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

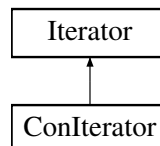
- Conditions.h
- Conditions.cpp

## 4.9 ConIterator Class Reference

Concrete iterator class Implements the interface for accessing and traversing bannerman elements in groundForces. Keeps track of the current position in the traversal of the aggregate(groundForces list).

```
#include <ConIterator.h>
```

Inheritance diagram for ConIterator:



### Public Member Functions

- `ConIterator` (list< `Bannerman` \* > X)
- `Bannerman` \* `Current` ()  
*Returns a pointer to the current bannerman object in groundForces being accessed, , primitive operations can be performed on the returned bannerman object.*
- `Bannerman` \* `next` ()  
*Sets the (\*it) pointer to the next bannerman object in groundForces to be accessed and returns it, , primitive operations can be performed on the returned bannerman object.*
- bool `hasNext` ()  
*Determines whether or not the current bannerman object which (\*it) currently points to is the last element in the list or not.*
- bool `isActive` ()  
*Returns whether or not the current item is active or not by determining whether HP>0 or not.*

### Private Attributes

- list< `Bannerman` \* > `armyList`
- list< `Bannerman` \* >::iterator `it`

### 4.9.1 Detailed Description

Concrete iterator class Implements the interface for accessing and traversing bannerman elements in groundForces. Keeps track of the current position in the traversal of the aggregate(groundForces list).

#### Author

Thapelo Thoka

#### Date

October 2022

### 4.9.2 Constructor & Destructor Documentation

#### 4.9.2.1 Conlterator()

```
ConIterator::ConIterator (
    list< Bannerman * > X )
```

[Conlterator](#) Constructor which takes in a list of bannerman objects.

#### Parameters

<i>X</i>	- the groundForces list which will be accessed sequentially.
----------	--

### 4.9.3 Member Function Documentation

#### 4.9.3.1 Current()

```
Bannerman * ConIterator::Current ( ) [virtual]
```

Returns a pointer to the current bannerman object in groundForces being accessed, , primitive operations can be performed on the returned bannerman object.

#### Returns

[Bannerman](#)\*

Implements [Iterator](#).

#### 4.9.3.2 hasNext()

```
bool ConIterator::hasNext ( ) [virtual]
```

Determines whether or not the current bannerman object which (\*it) currently points to is the last element in the list or not.

##### Returns

false  
true

##### Test

Implements [Iterator](#).

#### 4.9.3.3 isActive()

```
bool ConIterator::isActive ( ) [virtual]
```

Returns whether or not the current item is active or not by determining whether HP>0 or not.

##### Returns

true  
false

##### Test

Implements [Iterator](#).

#### 4.9.3.4 next()

```
Bannerman * ConIterator::next ( ) [virtual]
```

Sets the (\*it) pointer to the next bannerman object in groundForces to be accessed and returns it, , primitive operations can be performed on the returned bannerman object.

##### Returns

Bannerman\*

##### Test

Implements [Iterator](#).



## 4.9.4 Member Data Documentation

### 4.9.4.1 armyList

```
list<Bannerman*> ConIterator::armyList [private]
```

The list of bannerman objects to traverse

### 4.9.4.2 it

```
list<Bannerman*>::iterator ConIterator::it [private]
```

An iterator which is a pointer to the current bannerman object in [Commander](#)'s groundForces being accessed, primitive operations can be performed using this pointer.

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

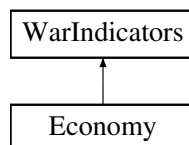
- ConIterator.h
- ConIterator.cpp

## 4.10 Economy Class Reference

keeps all variables relating to the state of [Economy](#), is context of States. A class that provides an alternative strategy for how [Bannerman](#) will fight enemyBannerman.

```
#include <Economy.h>
```

Inheritance diagram for Economy:



### Public Member Functions

- [Economy](#) ([State](#) \*state, int currency)  
*Constructor. initializes [State](#) pointer and currency.*
- void **SetState** ()  
*tells [State](#) to invoke its getDemotionState method.*
- [State](#) \* **getState** ()  
*returns [State](#) pointer.*
- void **decreaseCurrency** ()  
*tells [State](#) to invoke its decreaseCurrency method and then it invokes notify method of MasterofCoin passing itself as the argument.*
- int **getCurrency** ()  
*returns currency.*
- void **removeCurrency** (int i)  
*reduces currency by the passed in amount.*
- virtual ~**Economy** ()  
*destructor. deallocates [State](#).*

## Private Attributes

- [State](#) \* *state*
- int *currency*

## Additional Inherited Members

### 4.10.1 Detailed Description

keeps all variables relating to the state of [Economy](#), is context of States. A class that provides an alternative strategy for how [Bannerman](#) will fight enemyBannerman.

#### Author

Morgan Bentley

#### Date

October 2022

### 4.10.2 Constructor & Destructor Documentation

#### 4.10.2.1 Economy()

```
Economy::Economy (
    State * state,
    int currency )
```

Constructor. initializes [State](#) pointer and currency.

#### Parameters

<i>state</i>	- <a href="#">State</a> of <a href="#">Economy</a> .
<i>currency</i>	- Monetary reserves of the <a href="#">Kingdom</a> .

### 4.10.3 Member Function Documentation

#### 4.10.3.1 getCurrency()

```
int Economy::getCurrency ( )
```

returns currency.

**Returns**

currency of [Economy](#).

**4.10.3.2 getState()**

```
State * Economy::getState ( )
```

returns [State](#) pointer.

**Returns**

[State](#) of [Economy](#).

**4.10.3.3 removeCurrency()**

```
void Economy::removeCurrency (
    int i )
```

reduces currency by the passed in amount.

**Parameters**

<i>i</i>	- Expense of manufacturing required Supplies.
----------	---

**4.10.4 Member Data Documentation****4.10.4.1 currency**

```
int Economy::currency [private]
```

Monetary reserves of the [Kingdom](#)

**4.10.4.2 state**

```
State* Economy::state [private]
```

state of economy

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

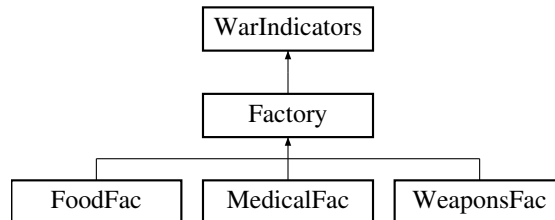
- Economy.h
- Economy.cpp

## 4.11 Factory Class Reference

[Factory](#) class to serve as a base class for all factory concrete classes used to create supplies.

```
#include <Factory.h>
```

Inheritance diagram for [Factory](#):



### Public Member Functions

- void **operation** ()  
*[operation\(\)](#) is the template Method that calls [make\(\)](#), indirectly manufacturing the supplies.*
- [ArmySupplies](#) \* **getSupply** ()  
*[getSupply\(\)](#) is the get Method used to get the supply variable.*
- virtual ~**Factory** ()  
*[~Factory\(\)](#) is the destructor for the [Factory](#) class. It is never implemented as its derived classes delete the supply pointer.*

### Protected Member Functions

- virtual [ArmySupplies](#) \* **make** ()=0  
*[make\(\)](#) is the factory Method called in [operation\(\)](#) to make the new supplies.*

### Private Attributes

- [ArmySupplies](#) \* **supply**

### Additional Inherited Members

#### 4.11.1 Detailed Description

[Factory](#) class to serve as a base class for all factory concrete classes used to create supplies.

Author

Ronin Brookes 19069686

#### 4.11.2 Member Function Documentation

#### 4.11.2.1 getSupply()

```
ArmySupplies * Factory::getSupply ( )
```

getSupply() is the get Method used to get the supply variable.

##### Returns

[ArmySupplies](#) pointer to the newly created supplies.

#### 4.11.2.2 make()

```
virtual ArmySupplies * Factory::make ( ) [protected], [pure virtual]
```

make() is the factory Method called in [operation\(\)](#) to make the new supplies.

##### Returns

[ArmySupplies](#) pointer to the newly created supplies.

Implemented in [FoodFac](#), [MedicalFac](#), and [WeaponsFac](#).

### 4.11.3 Member Data Documentation

#### 4.11.3.1 supply

```
ArmySupplies* Factory::supply [private]
```

pointer to the supplies

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

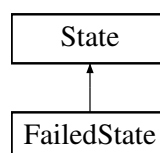
- [Factory.h](#)
- [Factory.cpp](#)

## 4.12 FailedState Class Reference

A concrete state class. A class that is one of the alternative concrete states for [Economy](#) of [Kingdom](#).

```
#include <FailedState.h>
```

Inheritance diagram for FailedState:



## Public Member Functions

- **FailedState** ()  
*Default constructor. initializes context to null.*
- void **decreaseCurrency** ()  
*prints out string stating that current [State](#) is [FailedState](#)*
- string **getState** ()  
*tells what the [State](#) of the [Economy](#) is.*

## Additional Inherited Members

### 4.12.1 Detailed Description

A concrete state class. A class that is one of the alternative concrete states for [Economy](#) of [Kingdom](#).

#### Author

Morgan Bentley

#### Date

October 2022

### 4.12.2 Member Function Documentation

#### 4.12.2.1 decreaseCurrency()

```
void FailedState::decreaseCurrency ( ) [virtual]
```

prints out string stating that current [State](#) is [FailedState](#)

Implements [State](#).

#### 4.12.2.2 getState()

```
string FailedState::getState ( ) [virtual]
```

tells what the [State](#) of the [Economy](#) is.

#### Returns

returns a string that says in text what the [State](#) is.

Implements [State](#).

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

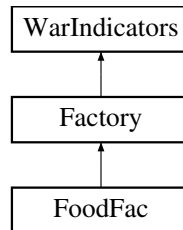
- FailedState.h
- FailedState.cpp

## 4.13 FoodFac Class Reference

[Factory](#) class to serve as a concrete creator class used to create Food supplies.

```
#include <FoodFac.h>
```

Inheritance diagram for FoodFac:



### Public Member Functions

- [ArmySupplies](#) \* [make](#) ()  
*[make\(\)](#) is the factory Method called in [operation\(\)](#) to make the Food supplies.*
- ~[FoodFac](#) ()  
*destructor for the [FoodFac](#) class to delete and free the supply variable.*

### Additional Inherited Members

#### 4.13.1 Detailed Description

[Factory](#) class to serve as a concrete creator class used to create Food supplies.

Author

Ronin Brookes 19069686

#### 4.13.2 Member Function Documentation

##### 4.13.2.1 [make\(\)](#)

```
ArmySupplies * FoodFac::make ( ) [virtual]
```

[make\(\)](#) is the factory Method called in [operation\(\)](#) to make the Food supplies.

Returns

[ArmySupplies](#) pointer to the newly created Food supplies.

Implements [Factory](#).

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

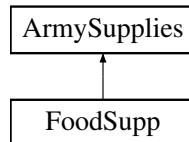
- FoodFac.h
- FoodFac.cpp

## 4.14 FoodSupp Class Reference

Concrete Product [Class](#) containing the information on the Food supplies.

```
#include <FoodSupp.h>
```

Inheritance diagram for FoodSupp:



### Public Member Functions

- int [getAmount](#) ()  
*[getAmount\(\)](#) is used to return the amount of Food supplies.*

#### 4.14.1 Detailed Description

Concrete Product [Class](#) containing the information on the Food supplies.

Author

Ronin Brookes 19069686

#### 4.14.2 Member Function Documentation

##### 4.14.2.1 [getAmount\(\)](#)

```
int FoodSupp::getAmount ( ) [virtual]
```

[getAmount\(\)](#) is used to return the amount of Food supplies.

Returns

an integer representation of the amount of Food supplies.

Implements [ArmySupplies](#).

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

- FoodSupp.h
- FoodSupp.cpp

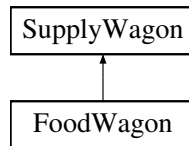


## 4.15 FoodWagon Class Reference

Concrete Prototype class to serve as the wagon for Food.

```
#include <FoodWagon.h>
```

Inheritance diagram for FoodWagon:



### Public Member Functions

- void [setSup](#) ([ArmySupplies](#) \*sup)  
*initialize the supp variable for the [FoodWagon](#).*
- [SupplyWagon](#) \* [clone](#) ()  
*Clone the Food Supply Wagon object.*
- [ArmySupplies](#) \* [getSupplies](#) ()  
*Return the supp variable for the Food Wagon.*
- ~**FoodWagon** ()  
*destructor for the Food Wagon class to delete and free the supply variable.*

### Public Attributes

- [ArmySupplies](#) \* [supp](#) =NULL

#### 4.15.1 Detailed Description

Concrete Prototype class to serve as the wagon for Food.

Author

Ronin Brookes 19069686

#### 4.15.2 Member Function Documentation

##### 4.15.2.1 clone()

```
SupplyWagon * FoodWagon::clone ( ) [virtual]
```

Clone the Food Supply Wagon object.

Returns

the new Food Supply Wagon clone.

Implements [SupplyWagon](#).

#### 4.15.2.2 getSupplies()

```
ArmySupplies * FoodWagon::getSupplies ( ) [virtual]
```

Return the supp variable for the Food Wagon.

##### Returns

returns the supp variable. Returns the necessary supplies.

Implements [SupplyWagon](#).

#### 4.15.2.3 setSup()

```
void FoodWagon::setSup (
    ArmySupplies * sup ) [virtual]
```

initialize the supp variable for the [FoodWagon](#).

##### Parameters

<i>sup</i>	is used to set the supp variable,
------------	-----------------------------------

Implements [SupplyWagon](#).

### 4.15.3 Member Data Documentation

#### 4.15.3.1 supp

```
ArmySupplies* FoodWagon::supp =NULL
```

pointer to the supplies in the Food wagon

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

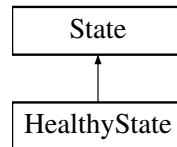
- FoodWagon.h
- FoodWagon.cpp

## 4.16 HealthyState Class Reference

A concrete state class. A class that is one of the alternative concrete states for [Economy](#) of [Kingdom](#).

```
#include <HealthyState.h>
```

Inheritance diagram for HealthyState:



### Public Member Functions

- **HealthyState ()**  
*Default constructor. initializes context to null.*
- void **decreaseCurrency ()**  
*decreases [Economy](#) currency and then checks if the conditions are sufficient for the economy to downgrade to a lower state*
- virtual **State \* getDemotionState ()**  
*gets lower level state of economy*
- string **getState ()**  
*tells what the [State](#) of the [Economy](#) is.*

### Additional Inherited Members

#### 4.16.1 Detailed Description

A concrete state class. A class that is one of the alternative concrete states for [Economy](#) of [Kingdom](#).

Author

Morgan Bentley

Date

October 2022

#### 4.16.2 Member Function Documentation

#### 4.16.2.1 decreaseCurrency()

```
void HealthyState::decreaseCurrency ( ) [virtual]
```

decreases [Economy](#) currency and then checks if the conditions are sufficient for the economy to downgrade to a lower state

Implements [State](#).

#### 4.16.2.2 getDemotionState()

```
State * HealthyState::getDemotionState ( ) [virtual]
```

gets lower level state of economy

##### Returns

concrete [State](#) of [Economy](#)

Reimplemented from [State](#).

#### 4.16.2.3 getState()

```
string HealthyState::getState ( ) [virtual]
```

tells what the [State](#) of the [Economy](#) is.

##### Returns

returns a string that says in text what the [State](#) is.

Implements [State](#).

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

- [HealthyState.h](#)
- [HealthyState.cpp](#)

## 4.17 Historian Class Reference

### Public Member Functions

- void [setAlly](#) ([Bannerman](#) \*b)  
*sets the bannerman variable*
- [History](#) \* [Store](#) ()  
*stores the bannerman in a memento object (history).*
- [Bannerman](#) \* [restoreAlly](#) ([History](#) \*h)  
*reverts the bannerman that was stored to original state for when they return.*
- [~Historian](#) ()  
*the destructor.*

## Private Attributes

- [Bannerman](#) \* `bannerman`

*The originator class in the memento design pattern.*

### 4.17.1 Member Function Documentation

#### 4.17.1.1 `restoreAlly()`

```
Bannerman * Historian::restoreAlly (
    History * h )
```

reverts the bannerman that was stored to original state for when they return.

##### Parameters

<i>h</i>	the history object being restored.
----------	------------------------------------

##### Returns

the bannerman that was restored from the stored history object.

#### 4.17.1.2 `setAlly()`

```
void Historian::setAlly (
    Bannerman * b )
```

sets the bannerman variable

##### Parameters

<i>b</i>	the bannerman being set to the class variable
----------	---

#### 4.17.1.3 `Store()`

```
History * Historian::Store ( )
```

stores the bannerman in a memento object (history).

##### Returns

the resulting history object with the bannerman.

## 4.17.2 Member Data Documentation

### 4.17.2.1 bannerman

```
Bannerman* Historian::bannerman [private]
```

The originator class in the memento design pattern.

#### Author

Julianna Venter.

#### Date

November 2022. a bannerman object that represents the current bannerman being stored - the one that has defected

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

- Historian.h
- Historian.cpp

## 4.18 History Class Reference

### Public Member Functions

- [History](#) ([Bannerman](#) \*b)  
*constructor that sets class variable.*
- [Bannerman](#) \* [getBannerman](#) ()  
*returns the stored bannerman*
- [~History](#) ()  
*the destructor*

### Private Attributes

- [Bannerman](#) \* [bannerman](#)  
*The memento class in the memento design pattern.*

## 4.18.1 Constructor & Destructor Documentation

### 4.18.1.1 History()

```
History::History (  
    Bannerman * b )
```

constructor that sets class variable.

## Parameters

<i>b</i>	parameter that class variable is set to.
----------	--

## 4.18.2 Member Function Documentation

### 4.18.2.1 getBannerman()

```
Bannerman * History::getBannerman ( )
```

returns the stored bannerman

## Returns

the stored bannerman

## 4.18.3 Member Data Documentation

### 4.18.3.1 bannerman

```
Bannerman* History::bannerman [private]
```

The memento class in the memento design pattern.

## Author

Julianna Venter

## Date

November 2022 a bannerman object that represents the current bannerman being stored - the one that has defected.

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

- History.h
- History.cpp

## 4.19 HistoryBook Class Reference

### Public Member Functions

- void `add (History *h)`  
*adds the passed in history object to the class list*
- `History * getAlly ()`  
*gets the required history object to be restored*

### Private Attributes

- `list< History * > defectedAllies`  
*The caretaker class in the memento design pattern.*

### 4.19.1 Member Function Documentation

#### 4.19.1.1 `add()`

```
void HistoryBook::add (  
    History * h )
```

adds the passed in history object to the class list

##### Parameters

<i>h</i>	the history object to be added
----------	--------------------------------

#### 4.19.1.2 `getAlly()`

```
History * HistoryBook::getAlly ( )
```

gets the required history object to be restored

##### Returns

the history object that will be restored

### 4.19.2 Member Data Documentation



#### 4.19.2.1 defectedAllies

```
list<History*> HistoryBook::defectedAllies [private]
```

The caretaker class in the memento design pattern.

##### Author

Julianna Venter

##### Date

November 2022 the list of history (memento) objects that represent bannermen that have defected

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

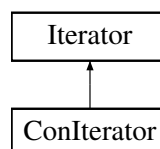
- HistoryBook.h
- HistoryBook.cpp

## 4.20 Iterator Class Reference

**Iterator** class Defines an interface for accessing and traversing bannerman elements in groundForces.

```
#include <Iterator.h>
```

Inheritance diagram for Iterator:



### Public Member Functions

- virtual **Bannerman** \* **Current** ()=0  
*Abstract. Returns a pointer to the current bannerman object in groundForces being accessed, , primitive operations can be performed on the returned bannerman object.*
- virtual bool **hasNext** ()=0  
*Abstract. determines whether or not the current bannerman object being accessed in groundForces is the last element in the list or not.*
- virtual **Bannerman** \* **next** ()=0  
*Abstract. Sets the current pointer to the next bannerman object in groundForces to be accessed and returns it, , primitive operations can be performed on the returned bannerman object.*
- virtual bool **isActive** ()=0  
*Abstract. Returns whether or not the current item is active or not.*

### 4.20.1 Detailed Description

[Iterator](#) class Defines an interface for accessing and traversing bannerman elements in groundForces.

#### Author

Thapelo Thoka

#### Date

October 2022

### 4.20.2 Member Function Documentation

#### 4.20.2.1 Current()

```
virtual Bannerman * Iterator::Current ( ) [pure virtual]
```

Abstract. Returns a pointer to the current bannerman object in groundForces being accessed, , primitive operations can be performed on the returned bannerman object.

#### Returns

Bannerman\*

Implemented in [Conliterator](#).

#### 4.20.2.2 hasNext()

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

Abstract. determines whether or not the current bannerman object being accessed in groundForces is the last element in the list or not.

#### Returns

false  
true

Implemented in [Conliterator](#).

#### 4.20.2.3 isActive()

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

Abstract. Returns whether or not the current item is active or not.

##### Returns

true  
false

Implemented in [ConIterator](#).

#### 4.20.2.4 next()

```
virtual Bannerman * Iterator::next ( ) [pure virtual]
```

Abstract. Sets the current pointer to the next bannerman object in groundForces to be accessed and returns it, , primitive operations can be performed on the returned bannerman object.

##### Returns

Bannerman\*

Implemented in [ConIterator](#).

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

- [Iterator.h](#)

## 4.21 Kingdom Class Reference

A [Kingdom](#) class that has all the fighting [Bannerman](#) and [Economy](#).

```
#include <Kingdom.h>
```

### Public Member Functions

- [Kingdom](#) ([Economy](#) \*economy)  
*Constructor. initializes [Economy](#) pointer and vector list.*
- void [remove](#) ([Bannerman](#) \*b)  
*removes specified [Bannerman](#) from vector list.*
- void [add](#) ([Bannerman](#) \*b)  
*adds specified [Bannerman](#) into vector list.*
- virtual ~[Kingdom](#) ()  
*destructor. deallocates all pointers of this class*
- int [getSize](#) ()  
*returns current number of [Bannerman](#) in the [Kingdom](#).*
- [Bannerman](#) \* [getAlly](#) (string n)  
*returns a particular allied [Bannerman](#) given a unique string name.*
- list< [Bannerman](#) \* > [getKingdom](#) ()  
*returns a list of [Bannerman](#) pointers in the [Kingdom](#).*

## Private Attributes

- list< [Bannerman](#) \* > [bannerman](#)
- [Economy](#) \* [economy](#)

### 4.21.1 Detailed Description

A [Kingdom](#) class that has all the fighting [Bannerman](#) and [Economy](#).

#### Author

Morgan Bentley

#### Date

October 2022

### 4.21.2 Constructor & Destructor Documentation

#### 4.21.2.1 Kingdom()

```
Kingdom::Kingdom (
    Economy * economy )
```

Constructor. initializes [Economy](#) pointer and vector list.

#### Parameters

<i>economy</i>	- <a href="#">Economy</a> pointer to player's <a href="#">Economy</a> object.
----------------	---

### 4.21.3 Member Function Documentation

#### 4.21.3.1 add()

```
void Kingdom::add (
    Bannerman * b )
```

adds specified [Bannerman](#) into vector list.

#### Parameters

<i>b</i>	- <a href="#">Bannerman</a> that has defected from enemyKingdom
----------	---

#### 4.21.3.2 getAlly()

```
Bannerman * Kingdom::getAlly (
    string n )
```

returns a particular allied [Bannerman](#) given a unique string name.

##### Parameters

<i>n</i>	- unique name of a particular <a href="#">Bannerman</a> .
----------	---

##### Returns

a particular allied [Bannerman](#).

#### 4.21.3.3 getKingdom()

```
list< Bannerman * > Kingdom::getKingdom ( )
```

returns a list of [Bannerman](#) pointers in the [Kingdom](#).

##### Returns

vector list of all [Bannerman](#) objects a [Kingdom](#) owns.

#### 4.21.3.4 getSize()

```
int Kingdom::getSize ( )
```

returns current number of [Bannerman](#) in the [Kingdom](#).

##### Returns

integer representing the number of [Bannerman](#) in [Kingdom](#).

#### 4.21.3.5 remove()

```
void Kingdom::remove (
    Bannerman * b )
```

removes specified [Bannerman](#) from vector list.

## Parameters

<i>b</i>	- <a href="#">Bannerman</a> that has lost a fight or defected to enemyKingdom.
----------	--

## 4.21.4 Member Data Documentation

### 4.21.4.1 bannerman

```
list<Bannerman*> Kingdom::bannerman [private]
```

vector list of all [Bannerman](#) objects a [Kingdom](#) owns

### 4.21.4.2 economy

```
Economy* Kingdom::economy [private]
```

pointer to the [Economy](#) of the [Kingdom](#)

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

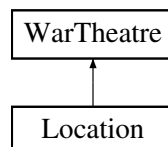
- Kingdom.h
- Kingdom.cpp

## 4.22 Location Class Reference

concrete object. This is the class/object that will be decorated. Inherits from [WarTheatre](#) class

```
#include <Location.h>
```

Inheritance diagram for Location:



## Public Member Functions

- **Location ()**  
*Default constructor.*
- void **sendScout ()**  
*the function that will be called to decorate the war venue*

### 4.22.1 Detailed Description

concrete object. This is the class/object that will be decorated. Inherits from [WarTheatre](#) class

#### Author

Keabetswe Mothapo

#### Date

October 2022

### 4.22.2 Member Function Documentation

#### 4.22.2.1 sendScout()

```
void Location::sendScout ( ) [virtual]
```

the function that will be called to decorate the war venue

Implements [WarTheatre](#).

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

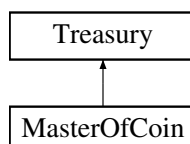
- Location.h
- Location.cpp

## 4.23 MasterOfCoin Class Reference

The Concrete class for the Mediator A class that allows for other classes to talk to when key changes are made.

```
#include <MasterOfCoin.h>
```

Inheritance diagram for MasterOfCoin:



## Public Member Functions

- **MasterOfCoin** (*Economy* \**economy*, *Raven* \**observer*, *Strategy* \**strategy*)  
*Constructor to set the variables in this class.*
- void **notify** (*WarIndicators* \**sender*)  
*This is the method that all objects will use to call the mediator when a change is made.*
- void **decreaseCurrency** ()  
*Will call the decreaseCurrency method of the *Economy*.*
- void **manufacture** ()  
*will call on the obsever to create more supplies*
- ~**MasterOfCoin** ()  
*This is the destructor used to deallocate all memory used in this class.*

## Private Attributes

- *Economy* \* **economy**  
*The *Economy* that the mediator uses.*
- *Raven* \* **observer**  
*The Observer that the mediator uses.*
- *Strategy* \* **strategy**  
*The *Strategy* the mediator uses.*

### 4.23.1 Detailed Description

The Concrete class for the Mediator A class that allows for other classes to talk to when key changes are made.

#### Author

Sameet Keshav u21479373

#### Date

October 2022

### 4.23.2 Constructor & Destructor Documentation

#### 4.23.2.1 MasterOfCoin()

```
MasterOfCoin::MasterOfCoin (
    Economy * economy,
    Raven * observer,
    Strategy * strategy )
```

Constructor to set the variables in this class.



## Parameters

<i>economy</i>	holds the <a href="#">Economy</a> that will be using the mediator
<i>observer</i>	holds the observer that will use the mediator
<i>strategy</i>	holds the strategy object that will use the mediator

### 4.23.3 Member Function Documentation

#### 4.23.3.1 notify()

```
void MasterOfCoin::notify (
    WarIndicators * sender ) [virtual]
```

This is the method that all objects will use to call the mediator when a change is made.

## Parameters

<i>sender</i>	is the variable that holds the object who sent the notify request
---------------	---

Implements [Treasury](#).

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

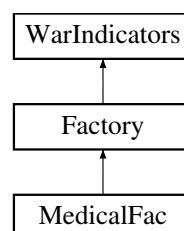
- MasterOfCoin.h
- MasterOfCoin.cpp

## 4.24 MedicalFac Class Reference

[Factory](#) class to serve as a concrete creator class used to create Medical supplies.

```
#include <MedicalFac.h>
```

Inheritance diagram for MedicalFac:



## Public Member Functions

- [ArmySupplies](#) \* [make](#) ()  
*[make\(\)](#) is the factory Method called in [operation\(\)](#) to make the Medical supplies.*
- [~MedicalFac](#) ()  
*destructor for the [MedicalFac](#) class to delete and free the supply variable.*

## Additional Inherited Members

### 4.24.1 Detailed Description

[Factory](#) class to serve as a concrete creator class used to create Medical supplies.

#### Author

Ronin Brookes 19069686

### 4.24.2 Member Function Documentation

#### 4.24.2.1 [make\(\)](#)

```
ArmySupplies * MedicalFac::make ( ) [virtual]
```

[make\(\)](#) is the factory Method called in [operation\(\)](#) to make the Medical supplies.

#### Returns

[ArmySupplies](#) pointer to the newly created Medical supplies.

Implements [Factory](#).

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

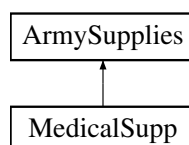
- [MedicalFac.h](#)
- [MedicalFac.cpp](#)

## 4.25 MedicalSupp Class Reference

Concrete Product [Class](#) containing the information on the Medical supplies.

```
#include <MedicalSupp.h>
```

Inheritance diagram for [MedicalSupp](#):



## Public Member Functions

- int [getAmount](#) ()  
*[getAmount\(\)](#) is used to return the amount of Medical supplies.*

### 4.25.1 Detailed Description

Concrete Product [Class](#) containing the information on the Medical supplies.

#### Author

Ronin Brookes 19069686

### 4.25.2 Member Function Documentation

#### 4.25.2.1 [getAmount\(\)](#)

```
int MedicalSupp::getAmount ( ) [virtual]
```

[getAmount\(\)](#) is used to return the amount of Medical supplies.

#### Returns

an integer representation of the amount of Medical supplies.

Implements [ArmySupplies](#).

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

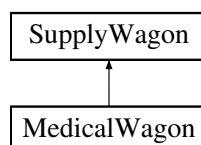
- MedicalSupp.h
- MedicalSupp.cpp

## 4.26 MedicalWagon Class Reference

Concrete Prototype class to serve as the wagon for Medical.

```
#include <MedicalWagon.h>
```

Inheritance diagram for MedicalWagon:



## Public Member Functions

- void [setSup](#) ([ArmySupplies](#) \*sup)  
*initialize the supp variable for the [MedicalWagon](#).*
- [SupplyWagon](#) \* [clone](#) ()  
*Clone the Medical Supply Wagon object.*
- [ArmySupplies](#) \* [getSupplies](#) ()  
*Return the supp variable for the Medical Wagon.*
- ~[MedicalWagon](#) ()  
*destructor for the Medical Wagon class to delete and free the supply variable.*

## Public Attributes

- [ArmySupplies](#) \* [supp](#)

### 4.26.1 Detailed Description

Concrete Prototype class to serve as the wagon for Medical.

Author

Ronin Brookes 19069686

### 4.26.2 Member Function Documentation

#### 4.26.2.1 clone()

```
SupplyWagon * MedicalWagon::clone ( ) [virtual]
```

Clone the Medical Supply Wagon object.

Returns

the new Medical Supply Wagon clone.

Implements [SupplyWagon](#).

#### 4.26.2.2 getSupplies()

```
ArmySupplies * MedicalWagon::getSupplies ( ) [virtual]
```

Return the supp variable for the Medical Wagon.

Returns

returns the supp variable. Returns the necessary supplies.

Implements [SupplyWagon](#).

#### 4.26.2.3 setSup()

```
void MedicalWagon::setSup (  
    ArmySupplies * sup ) [virtual]
```

initialize the supp variable for the [MedicalWagon](#).

## Parameters

<code>sup</code>	is used to set the supp variable,
------------------	-----------------------------------

Implements [SupplyWagon](#).

## 4.26.3 Member Data Documentation

### 4.26.3.1 `supp`

[ArmySupplies](#)\* `MedicalWagon::supp`

pointer to the supplies in the Medical wagon

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

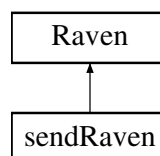
- `MedicalWagon.h`
- `MedicalWagon.cpp`

## 4.27 Raven Class Reference

The Abstract class for the Observer A class that the Concrete Observer class inherits from.

```
#include <Raven.h>
```

Inheritance diagram for Raven:



### Public Member Functions

- virtual void [update](#) ()=0

*The Pure Virtual function for the update method that the concrete class uses.*

### 4.27.1 Detailed Description

The Abstract class for the Observer A class that the Concrete Observer class inherits from.

#### Author

Sameet Keshav u21479373

#### Date

October 2022

## 4.27.2 Member Function Documentation

### 4.27.2.1 update()

```
virtual void Raven::update ( ) [pure virtual]
```

The Pure Virtual function for the update method that the concrete class uses.

Implemented in [sendRaven](#).

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

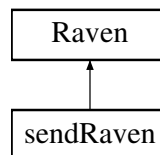
- [Raven.h](#)

## 4.28 sendRaven Class Reference

The Concrete class for the Observer A class that watches the [Bannerman](#) for any changes that are made.

```
#include <sendRaven.h>
```

Inheritance diagram for sendRaven:



### Public Member Functions

- [sendRaven](#) ([SupplyWagon](#) \*\*supplies, [Bannerman](#) \*subject)  
*constructor for the concrete [Raven](#) class*
- void [update](#) ()  
*This method will be called from outside the class, this method will get the amount of resources the subject has, then call the check method.*
- void **checkSupplies** ()  
*This method will check to see if there is too little supplies then call the appropriate Supply Wagon to fill it up.*
- ~**sendRaven** ()  
*The Destructor will be used to deference all pointer variables used by this class to stop all memory leaks.*

## Private Attributes

- `int numFood`  
*Amount of food the bannerman has.*
- `int numMedical`  
*Amount of medical supplies the bannerman has.*
- `int numWeapons`  
*Amount of medical supplies the bannerman has.*
- `SupplyWagon ** supplies`  
*The array of [SupplyWagon](#)'s used to hold the different types of supplies.*
- `Bannerman * subject`  
*The [Bannerman](#) subject the Observer watches.*

### 4.28.1 Detailed Description

The Concrete class for the Observer A class that watches the [Bannerman](#) for any changes that are made.

#### Author

Sameet Keshav u21479373

#### Date

October 2022

### 4.28.2 Constructor & Destructor Documentation

#### 4.28.2.1 sendRaven()

```
sendRaven::sendRaven (
    SupplyWagon ** supplies,
    Bannerman * subject )
```

constructor for the concrete [Raven](#) class

#### Parameters

<i>supplies</i>	is an array of pointer objects of type <a href="#">SupplyWagon</a>
<i>subject</i>	is a pointer for the <a href="#">Bannerman</a> subject that this Observer will observe

### 4.28.3 Member Function Documentation

#### 4.28.3.1 update()

```
void sendRaven::update ( ) [virtual]
```

This method will be called from outside the class, this method will get the amount of resources the subject has, then call the check method.

Implements [Raven](#).

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

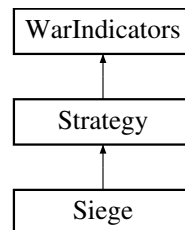
- [sendRaven.h](#)
- [sendRaven.cpp](#)

## 4.29 Siege Class Reference

A concrete strategy class. A class that provides an alternative strategy for how [Bannerman](#) will fight enemy↔ Bannerman.

```
#include <Siege.h>
```

Inheritance diagram for Siege:



### Public Member Functions

- [Siege](#) (int [stealth](#), [Kingdom](#) \*myKingdom, [Kingdom](#) \*enemyKingdom, [Bannerman](#) \*myBannerman, [Bannerman](#) \*enemyBannerman, string name, int min, int minFavour, [Historian](#) \*h, [HistoryBook](#) \*hb)  
*constructor. initializes stealth and calls base class constructor to initialize myKingdom, enemyKingdom, my↔ Bannerman, enemyBannerman, strategy, min and minFavour.*
- bool [attack](#) ([Bannerman](#) \*myBannerman, [Bannerman](#) \*enemyBannerman)  
*makes two bannerman from enemy kingdoms fight each other until one loses*
- ~[Siege](#) ()  
*destructor.*

### Private Attributes

- int [stealth](#)



## Additional Inherited Members

### 4.29.1 Detailed Description

A concrete strategy class. A class that provides an alternative strategy for how [Bannerman](#) will fight enemy↵  
Bannerman.

#### Author

Morgan Bentley

#### Date

October 2022

### 4.29.2 Constructor & Destructor Documentation

#### 4.29.2.1 Siege()

```
Siege::Siege (
    int stealth,
    Kingdom * myKingdom,
    Kingdom * enemyKingdom,
    Bannerman * myBannerman,
    Bannerman * enemyBannerman,
    string name,
    int min,
    int minFavour,
    Historian * h,
    HistoryBook * hb )
```

constructor. initializes stealth and calls base class constructor to initialize myKingdom, enemyKingdom, my↵  
Bannerman, enemyBannerman, strategy, min and minFavour.

#### Parameters

<i>stealth</i>	- skill level of <a href="#">Bannerman</a> .
<i>min</i>	- minimum supplies for food,weapons and medicine
<i>minFavour</i>	- minimum favour below which bannerman change allegiances

### 4.29.3 Member Function Documentation

#### 4.29.3.1 attack()

```
bool Siege::attack (
    Bannerman * myBannerman,
    Bannerman * enemyBannerman ) [virtual]
```

makes two bannerman from enemy kingdoms fight each other until one loses

##### Parameters

<i>myBannerman</i>	- attacking bannerman object.
<i>enemyBannerman</i>	- <a href="#">Bannerman</a> object being attacked.

##### Returns

battle result as a boolean with true implying the attacking bannerman object won and false implying the opposite

Implements [Strategy](#).

### 4.29.4 Member Data Documentation

#### 4.29.4.1 stealth

```
int Siege::stealth [private]
```

skill level of [Bannerman](#) using this strategy

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

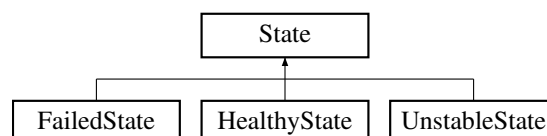
- Siege.h
- Siege.cpp

## 4.30 State Class Reference

An Abstract state class. A class that provides an interface to the alternative concrete states for [Economy](#) of [Kingdom](#).

```
#include <State.h>
```

Inheritance diagram for State:



## Public Member Functions

- **State** ()  
*Default constructor.*
- virtual void [setContext](#) ([Economy](#) \*context)  
*sets context variable.*
- virtual void [decreaseCurrency](#) ()=0  
*Abstract method.*
- virtual [State](#) \* [getDemotionState](#) ()  
*gets lower level state of [Economy](#) with null being returned if no lower [State](#) is possible*
- virtual ~**State** ()  
*destructor. deallocates context*
- virtual string [getState](#) ()=0  
*Abstract method.*

## Protected Attributes

- [Economy](#) \* context

### 4.30.1 Detailed Description

An Abstract state class. A class that provides an interface to the alternative concrete states for [Economy](#) of [Kingdom](#).

#### Author

Morgan Bentley

#### Date

October 2022

### 4.30.2 Member Function Documentation

#### 4.30.2.1 decreaseCurrency()

```
virtual void State::decreaseCurrency ( ) [pure virtual]
```

Abstract method.

Implemented in [FailedState](#), [HealthyState](#), and [UnstableState](#).

#### 4.30.2.2 getDemotionState()

```
State * State::getDemotionState ( ) [virtual]
```

gets lower level state of [Economy](#) with null being returned if no lower [State](#) is possible

##### Returns

concrete [State](#) of [Economy](#)

Reimplemented in [HealthyState](#), and [UnstableState](#).

#### 4.30.2.3 getState()

```
virtual string State::getState ( ) [pure virtual]
```

Abstract method.

##### Returns

name of [State](#) as a string.

Implemented in [FailedState](#), [HealthyState](#), and [UnstableState](#).

#### 4.30.2.4 setContext()

```
void State::setContext (
    Economy * context ) [virtual]
```

sets context variable.

##### Parameters

<i>context</i>	- <a href="#">Economy</a> pointer to player's <a href="#">Economy</a> object.
----------------	---

### 4.30.3 Member Data Documentation

#### 4.30.3.1 context

```
Economy* State::context [protected]
```

[Economy](#) pointer of player's [Kingdom](#)

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

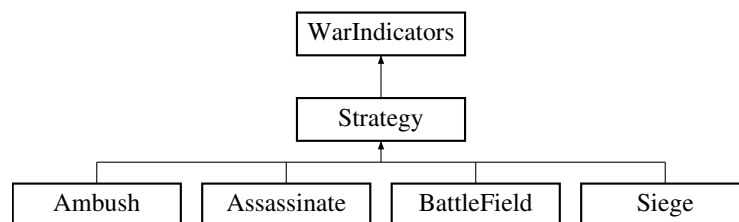
- State.h
- State.cpp

## 4.31 Strategy Class Reference

An Abstract strategy class. A class that provides an interface to the alternative concrete strategies for how [Bannerman](#) will fight enemyBannerman.

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

Inheritance diagram for Strategy:



### Public Member Functions

- [Strategy](#) ([Kingdom](#) \*myKingdom, [Kingdom](#) \*enemyKingdom, [Bannerman](#) \*myBannerman, [Bannerman](#) \*enemyBannerman, string name, int min, int minFavour, [Historian](#) \*h, [HistoryBook](#) \*hb)  
*constructor. initializes myKingdom, enemyKingdom, myBannerman, enemyBannerman, strategy, minSupplies and minFavour.*
- virtual bool [attack](#) ([Bannerman](#) \*myBannerman, [Bannerman](#) \*enemyBannerman)=0  
*Abstract method.*
- virtual string [getStrategyName](#) ()  
*gets strategy variable*
- virtual [Bannerman](#) \* [getMyBannerman](#) ()  
*gets attacking Bannerman*
- virtual [Bannerman](#) \* [getEnemyBannerman](#) ()  
*gets Bannerman being attacked*
- virtual ~**Strategy** ()  
*destructor. deallocates all pointers of this class*

### Protected Attributes

- [Kingdom](#) \* myKingdom
- [Kingdom](#) \* enemyKingdom
- [Bannerman](#) \* myBannerman
- [Bannerman](#) \* enemyBannerman
- string strategy
- int minFavour
- int minSupplies
- [HistoryBook](#) \* BookOfDura
- [Historian](#) \* Greg
- int defectedAllies

### 4.31.1 Detailed Description

An Abstract strategy class. A class that provides an interface to the alternative concrete strategies for how [Bannerman](#) will fight enemyBannerman.

Author

Morgan Bentley

Date

October 2022

### 4.31.2 Constructor & Destructor Documentation

#### 4.31.2.1 Strategy()

```
Strategy::Strategy (
    Kingdom * myKingdom,
    Kingdom * enemyKingdom,
    Bannerman * myBannerman,
    Bannerman * enemyBannerman,
    string name,
    int min,
    int minFavour,
    Historian * h,
    HistoryBook * hb )
```

constructor. initializes myKingdom, enemyKingdom, myBannerman, enemyBannerman, strategy, minSupplies and minFavour.

Parameters

<i>myKingdom</i>	- <a href="#">Kingdom</a> pointer of attacking <a href="#">Bannerman's Kingdom</a> .
<i>enemyKingdom</i>	- <a href="#">Kingdom</a> pointer of enemyBannerman's <a href="#">Kingdom</a> .
<i>myBannerman</i>	- attacking <a href="#">Bannerman</a> .
<i>enemyBannerman</i>	- defending <a href="#">Bannerman</a> .
<i>name</i>	- name of concrete strategy.
<i>min</i>	- minimum supplies for food, weapons and medicine
<i>minFavour</i>	- minimum favour below which bannerman change allegiances
<i>h</i>	- <a href="#">Historian</a> (originator) to save defected bannemen.
<i>hb</i>	- Caretaker for memento implementation.

### 4.31.3 Member Function Documentation

#### 4.31.3.1 attack()

```
virtual bool Strategy::attack (
    Bannerman * myBannerman,
    Bannerman * enemyBannerman ) [pure virtual]
```

Abstract method.

##### Parameters

<i>myBannerman</i>	- attacking <a href="#">Bannerman</a> object.
<i>enemyBannerman</i>	- <a href="#">Bannerman</a> object being attacked.

##### Returns

battle result as a boolean with true implying the attacking bannerman object won and false implying the opposite

Implemented in [Ambush](#), [Assassinate](#), [BattleField](#), and [Siege](#).

#### 4.31.3.2 getEnemyBannerman()

```
Bannerman * Strategy::getEnemyBannerman ( ) [virtual]
```

gets [Bannerman](#) being attacked

##### Returns

enemyBannerman pointer

#### 4.31.3.3 getMyBannerman()

```
Bannerman * Strategy::getMyBannerman ( ) [virtual]
```

gets attacking [Bannerman](#)

##### Returns

myBannerman pointer

#### 4.31.3.4 `getStrategyName()`

```
string Strategy::getStrategyName ( ) [virtual]
```

gets strategy variable

##### Returns

strategy variable

### 4.31.4 Member Data Documentation

#### 4.31.4.1 `BookOfDura`

```
HistoryBook* Strategy::BookOfDura [protected]
```

Caretaker for memento implementation

#### 4.31.4.2 `defectedAllies`

```
int Strategy::defectedAllies [protected]
```

amount of defected allies

#### 4.31.4.3 `enemyBannerman`

```
Bannerman* Strategy::enemyBannerman [protected]
```

defending [Bannerman](#)

#### 4.31.4.4 `enemyKingdom`

```
Kingdom* Strategy::enemyKingdom [protected]
```

[Kingdom](#) pointer of enemyBannerman's [Kingdom](#)

#### 4.31.4.5 `Greg`

```
Historian* Strategy::Greg [protected]
```

[Historian](#) (originator) to save defected bannemen



#### 4.31.4.6 minFavour

```
int Strategy::minFavour [protected]
```

minimum favour below which [Bannerman](#) change allegiances

#### 4.31.4.7 minSupplies

```
int Strategy::minSupplies [protected]
```

minimum supplies for food, weapons and medicine

#### 4.31.4.8 myBannerman

```
Bannerman* Strategy::myBannerman [protected]
```

attacking [Bannerman](#)

#### 4.31.4.9 myKingdom

```
Kingdom* Strategy::myKingdom [protected]
```

[Kingdom](#) pointer of attacking [Bannerman](#)'s [Kingdom](#)

#### 4.31.4.10 strategy

```
string Strategy::strategy [protected]
```

name of concrete strategy

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

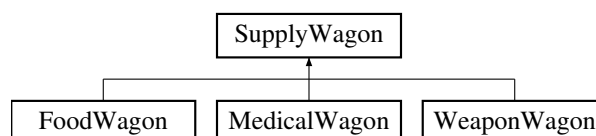
- Strategy.h
- Strategy.cpp

## 4.32 SupplyWagon Class Reference

Abstract Prototype class [SupplyWagon](#) to serve as the base class for all derived concrete prototypes.

```
#include <SupplyWagon.h>
```

Inheritance diagram for SupplyWagon:



## Public Member Functions

- virtual void [setSup](#) ([ArmySupplies](#) \*sup)=0  
*initialize the supp variable.*
- virtual [SupplyWagon](#) \* [clone](#) ()=0  
*Clone the Supply Wagon object.*
- virtual [ArmySupplies](#) \* [getSupplies](#) ()=0  
*Return the supp variable.*

## Public Attributes

- [ArmySupplies](#) \* [supp](#)

### 4.32.1 Detailed Description

Abstract Prototype class [SupplyWagon](#) to serve as the base class for all derived concrete prototypes.

Author

Ronin Brookes 19069686

### 4.32.2 Member Function Documentation

#### 4.32.2.1 clone()

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

Clone the Supply Wagon object.

Returns

the new Supply Wagon clone.

Implemented in [FoodWagon](#), [MedicalWagon](#), and [WeaponWagon](#).

#### 4.32.2.2 getSupplies()

```
virtual ArmySupplies * SupplyWagon::getSupplies ( ) [pure virtual]
```

Return the supp variable.

Returns

returns the supp variable. Returns the necessary supplies.

Implemented in [FoodWagon](#), [MedicalWagon](#), and [WeaponWagon](#).

#### 4.32.2.3 setSup()

```
virtual void SupplyWagon::setSup (  
    ArmySupplies * sup ) [pure virtual]
```

initialize the supp variable.

## Parameters

<code>sup</code>	used to set the supp variable.
------------------	--------------------------------

Implemented in [FoodWagon](#), [MedicalWagon](#), and [WeaponWagon](#).

### 4.32.3 Member Data Documentation

#### 4.32.3.1 `supp`

`ArmySupplies* SupplyWagon::supp`

pointer to the supplies in the wagon

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

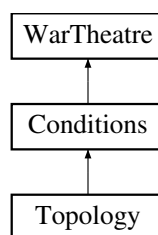
- `SupplyWagon.h`

## 4.33 Topology Class Reference

concrete decorator B. This is class that decorates the topology of the venue. Inherits from [Conditions](#) class

```
#include <Topology.h>
```

Inheritance diagram for Topology:



### Public Member Functions

- `void setTopology ()`  
*setter for the topology of the war venue uses the inherited venue variable to adjust difficulty of teh terrian*
- `Topology (WarTheatre *myTheatre)`  
*default constructor that calls the decorator function*
- `void sendScout ()`  
*implements the decorator function*

### 4.33.1 Detailed Description

concrete decorator B. This is class that decorates the topology of the venue. Inherits from [Conditions](#) class

#### Author

Keabetswe Mothapo

#### Date

October 2022

### 4.33.2 Member Function Documentation

#### 4.33.2.1 sendScout()

```
void Topology::sendScout ( ) [virtual]
```

implements the decorator function

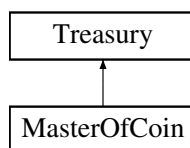
Reimplemented from [Conditions](#).

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

- Topology.h
- Topology.cpp

## 4.34 Treasury Class Reference

Inheritance diagram for Treasury:



### Public Member Functions

- virtual void [notify](#) ([WarIndicators](#) \*sender)=0  
*The pure virtual function for the notify function.*

### 4.34.1 Member Function Documentation

## 4.34.1.1 notify()

```
virtual void Treasury::notify (
    WarIndicators * sender ) [pure virtual]
```

The pure virtual function for the notify function.

Implemented in [MasterOfCoin](#).

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

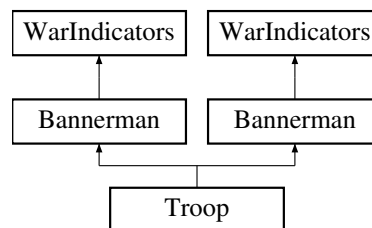
- [Treasury.h](#)

## 4.35 Troop Class Reference

A class that defines the primitive objects of the [Bannerman](#) composition.

```
#include <Troop.h>
```

Inheritance diagram for Troop:



## Public Member Functions

- **Troop** (string [name](#), int favor, int [numFood](#), int Medical, int [HP](#), [WarTheatre](#) \*warZone, [Strategy](#) \*strategy, [MasterOfCoin](#) \*m, bool [assassin](#), int [size](#))  
*Constructor. Initializes the name; favour; numFood; Medical; HP; warZone; strategy; m; assassin and size variables of the troop.*
- int [getHP](#) () override  
*Returns the troop's HP.*
- int [getSize](#) ()  
*Returns the troop's size.*
- void [attach](#) ([Raven](#) \*o) override  
*Attaches a [Raven](#) observer to the troop.*
- void [detach](#) ([Raven](#) \*o) override  
*Detaches a [Raven](#) observer to the troop.*
- void [increasePower](#) (int boost) override  
*increases the damage variables of the troop.*
- void [attack](#) ([Bannerman](#) \*myBannerman, [Bannerman](#) \*enemyBannerman) override  
*uses strategy to make a troop attack another kingdom.*
- void [decreaseWeapons](#) () override  
*decreases the numWeapons variable of the troop by 1.*
- void [decreaseFood](#) () override

- decreases the numFood variable of the troop by 1.*
- void `decreaseMedical ()` override
  - decreases the numMedical variable of the troop by 1.*
- void `changeStrategy (Strategy *strategy)` override
  - Changes the attack strategy of the troop.*
- void `increaseFavour ()` override
  - Increases the loyalty favour level of the troop by 1.*
- void `decreaseFavour ()` override
  - Decreases the loyalty favour level of the troop by 1.*
- int `getFavour ()` override
  - Returns the the loyalty favour levels variable of the `Troop` object.*
- void `increaseHP (int boost)` override
  - increases the HP of the troop.*
- int `getDamage ()` override
  - Accessor. Returns the damage capacity of the troop.*
- void `receiveDamage (int X)` override
  - increases the damage capacity of the troop.*
- int `getWeapons ()` override
  - Accessor. Returns the number of weapons the troop has.*
- int `getFood ()` override
  - Accessor. Returns the amount of food the troop has.*
- int `getMedical ()` override
  - Accessor. Returns the number of medical supplies the troop has.*
- void `setWeapons (int numWeapons)` override
  - Sets the number of weapon supplies the troop has.*
- void `setFood (int numFood)` override
  - Sets the number of food supplies the troop has.*
- void `decreasePower (int x)` override
  - Decreases the damage capability of the troop.*
- void `setMedical (int numMedical)` override
  - Sets the number of medical supplies the troop has.*
- void `setRaven (list< Raven * > r)` override
  - Assigns a list of `Raven` Observers to the troop's ravenList.*
- void `setMaster (MasterOfCoin *m)` override
  - Assigns a MasterofCoin mediator to ensure that the troop has the supplies it needs.*
- `~Troop ()`
- string `getName ()` override
  - Returns the name of this `Troop` object.*
- list< `Bannerman * >` `getTroops ()` override
  - `Bannerman` pure virtual method, not used by troop.*

## Private Attributes

- int `size`
- `Strategy * strategy`
- int `damage`
- `WarTheatre * warZone`

## Additional Inherited Members

### 4.35.1 Detailed Description

A class that defines the primitive objects of the [Bannerman](#) composition.

#### Author

Thapelo Thoka

#### Date

October 2022

### 4.35.2 Constructor & Destructor Documentation

#### 4.35.2.1 `~Troop()`

```
Troop::~~Troop ( )
```

Default destructor.

### 4.35.3 Member Function Documentation

#### 4.35.3.1 `attach()`

```
void Troop::attach (
    Raven * o ) [override], [virtual]
```

Attaches a [Raven](#) observer to the troop.

#### Parameters

<code>o</code>	- the <a href="#">Raven</a> observer object to attach
----------------	---

Implements [Bannerman](#).

#### 4.35.3.2 `attack()`

```
void Troop::attack (
```

```
Bannerman * myBannerman,  
Bannerman * enemyBannerman ) [override], [virtual]
```

uses strategy to make a troop attack another kingdom.

#### Parameters

<i>myBannerman</i>	- The attacking bannerman object.
<i>enemyBannerman</i>	- The bannerman object being attacked.

Implements [Bannerman](#).

#### 4.35.3.3 changeStrategy()

```
void Troop::changeStrategy (  
    Strategy * strategy ) [override], [virtual]
```

Changes the attack strategy of the troop.

#### Parameters

<i>strategy</i>	- The new strategy the troop should have.
-----------------	---

Implements [Bannerman](#).

#### 4.35.3.4 decreaseFavour()

```
void Troop::decreaseFavour ( ) [override], [virtual]
```

Decreases the loyalty favour level of the troop by 1.

Implements [Bannerman](#).

#### 4.35.3.5 decreaseFood()

```
void Troop::decreaseFood ( ) [override], [virtual]
```

decreases the numFood variable of the troop by 1.

Implements [Bannerman](#).



#### 4.35.3.6 decreaseMedical()

```
void Troop::decreaseMedical ( ) [override], [virtual]
```

decreases the numMedical variable of the troop by 1.

Implements [Bannerman](#).

#### 4.35.3.7 decreasePower()

```
void Troop::decreasePower (
    int x ) [override], [virtual]
```

Decreases the damage capability of the troop.

##### Parameters

<i>x</i>	- the number by which to decrease damage.
----------	---

Implements [Bannerman](#).

#### 4.35.3.8 decreaseWeapons()

```
void Troop::decreaseWeapons ( ) [override], [virtual]
```

decreases the numWeapons variable of the troop by 1.

Implements [Bannerman](#).

#### 4.35.3.9 detach()

```
void Troop::detach (
    Raven * o ) [override], [virtual]
```

Detaches a [Raven](#) observer to the troop.

##### Parameters

<i>o</i>	- the <a href="#">Raven</a> observer object to detach
----------	---

Implements [Bannerman](#).

#### 4.35.3.10 `getDamage()`

```
int Troop::getDamage ( ) [override], [virtual]
```

Accessor. Returns the damage capacity of the troop.

##### Returns

damage.

Implements [Bannerman](#).

#### 4.35.3.11 `getFavour()`

```
int Troop::getFavour ( ) [override], [virtual]
```

Returns the the loyalty favour levels variable of the [Troop](#) object.

##### Returns

favour

Implements [Bannerman](#).

#### 4.35.3.12 `getFood()`

```
int Troop::getFood ( ) [override], [virtual]
```

Accessor. Returns the amount of food the troop has.

##### Returns

numFood.

Implements [Bannerman](#).

#### 4.35.3.13 `getHP()`

```
int Troop::getHP ( ) [override], [virtual]
```

Returns the troop's HP.

##### Returns

HP.

Implements [Bannerman](#).

#### 4.35.3.14 getMedical()

```
int Troop::getMedical ( ) [override], [virtual]
```

Accessor. Returns the number of medical supplies the troop has.

##### Returns

numMedical.

Implements [Bannerman](#).

#### 4.35.3.15 getName()

```
string Troop::getName ( ) [override], [virtual]
```

Returns the name of this [Troop](#) object.

##### Returns

name

Implements [Bannerman](#).

#### 4.35.3.16 getSize()

```
int Troop::getSize ( )
```

Returns the troop's size.

##### Returns

size.

#### 4.35.3.17 getTroops()

```
list< Bannerman * > Troop::getTroops ( ) [override], [virtual]
```

[Bannerman](#) pure virtual method, not used by troop.

##### Returns

null

Implements [Bannerman](#).

#### 4.35.3.18 getWeapons()

```
int Troop::getWeapons ( ) [override], [virtual]
```

Accessor. Returns the number of weapons the troop has.

##### Returns

numWeapons.

Implements [Bannerman](#).

#### 4.35.3.19 increaseFavour()

```
void Troop::increaseFavour ( ) [override], [virtual]
```

Increases the loyalty favour level of the troop by 1.

Implements [Bannerman](#).

#### 4.35.3.20 increaseHP()

```
void Troop::increaseHP (
    int boost ) [override], [virtual]
```

increases the HP of the troop.

##### Parameters

<i>boost</i>	- The number by which to increase the HP of the troop.
--------------	--

Implements [Bannerman](#).

#### 4.35.3.21 increasePower()

```
void Troop::increasePower (
    int boost ) [override], [virtual]
```

increases the damage variables of the troop.

##### Parameters

<i>boost</i>	- The number by which to increase the damage of the troop.
--------------	--

Implements [Bannerman](#).

#### 4.35.3.22 receiveDamage()

```
void Troop::receiveDamage (
    int X ) [override], [virtual]
```

increases the damage capacity of the troop.

##### Parameters

<i>X</i>	- the number by which to increase the damage capacity of the troop.
----------	---

Implements [Bannerman](#).

#### 4.35.3.23 setFood()

```
void Troop::setFood (
    int numFood ) [override], [virtual]
```

Sets the number of food supplies the troop has.

##### Parameters

<i>numFood</i>	- The new numFood the troop should have.
----------------	--

Implements [Bannerman](#).

#### 4.35.3.24 setMaster()

```
void Troop::setMaster (
    MasterOfCoin * m ) [override], [virtual]
```

Assigns a MasterofCoin mediator to ensure that the troop has the supplies it needs.

##### Parameters

<i>m</i>	- the new <a href="#">MasterOfCoin</a> mediator to assign to the troop.
----------	---

Implements [Bannerman](#).

#### 4.35.3.25 setMedical()

```
void Troop::setMedical (
    int numMedical ) [override], [virtual]
```

Sets the number of medical supplies the troop has.

##### Parameters

<i>numMedical</i>	- The new numMedical the troop should have.
-------------------	---

Implements [Bannerman](#).

#### 4.35.3.26 setRaven()

```
void Troop::setRaven (
    list< Raven * > r ) [override], [virtual]
```

Assigns a list of [Raven](#) Observers to the troop's ravenList.

##### Parameters

<i>r</i>	- the list of <a href="#">Raven</a> Observers to attach to the troop.
----------	---

Implements [Bannerman](#).

#### 4.35.3.27 setWeapons()

```
void Troop::setWeapons (
    int numWeapons ) [override], [virtual]
```

Sets the number of weapon supplies the troop has.

##### Parameters

<i>numWeapons</i>	- The new numWeapons the troop should have.
-------------------	---

Implements [Bannerman](#).

### 4.35.4 Member Data Documentation

#### 4.35.4.1 damage

```
int Troop::damage [private]
```

The damage capability which the troop has

#### 4.35.4.2 size

```
int Troop::size [private]
```

The number of soldiers in the troop

#### 4.35.4.3 strategy

```
Strategy* Troop::strategy [private]
```

The attack strategy used by the troop in battle

#### 4.35.4.4 warZone

```
WarTheatre* Troop::warZone [private]
```

The war theatre at which this troop engages in battle

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

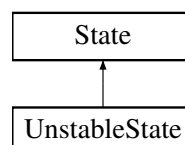
- Troop.h
- Troop.cpp

## 4.36 UnstableState Class Reference

A concrete state class. A class that is one of the alternative concrete states for [Economy](#) of [Kingdom](#).

```
#include <UnstableState.h>
```

Inheritance diagram for UnstableState:



## Public Member Functions

- **UnstableState** ()  
*Default constructor. initializes context to null.*
- void **decreaseCurrency** ()  
*decreases [Economy](#) currency and then checks if the conditions are sufficient for the economy to downgrade to a lower state*
- **State** \* **getDemotionState** ()  
*gets lower level state of economy*
- string **getState** ()  
*tells what the [State](#) of the [Economy](#) is.*

## Additional Inherited Members

### 4.36.1 Detailed Description

A concrete state class. A class that is one of the alternative concrete states for [Economy](#) of [Kingdom](#).

#### Author

Morgan Bentley

#### Date

October 2022

### 4.36.2 Member Function Documentation

#### 4.36.2.1 decreaseCurrency()

```
void UnstableState::decreaseCurrency ( ) [virtual]
```

decreases [Economy](#) currency and then checks if the conditions are sufficient for the economy to downgrade to a lower state

Implements [State](#).

#### 4.36.2.2 getDemotionState()

```
State * UnstableState::getDemotionState ( ) [virtual]
```

gets lower level state of economy

#### Returns

concrete [State](#) of [Economy](#)

Reimplemented from [State](#).



## 4.36.2.3 getState()

```
string UnstableState::getState ( ) [virtual]
```

tells what the [State](#) of the [Economy](#) is.

## Returns

returns a string that says in text what the [State](#) is.

Implements [State](#).

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

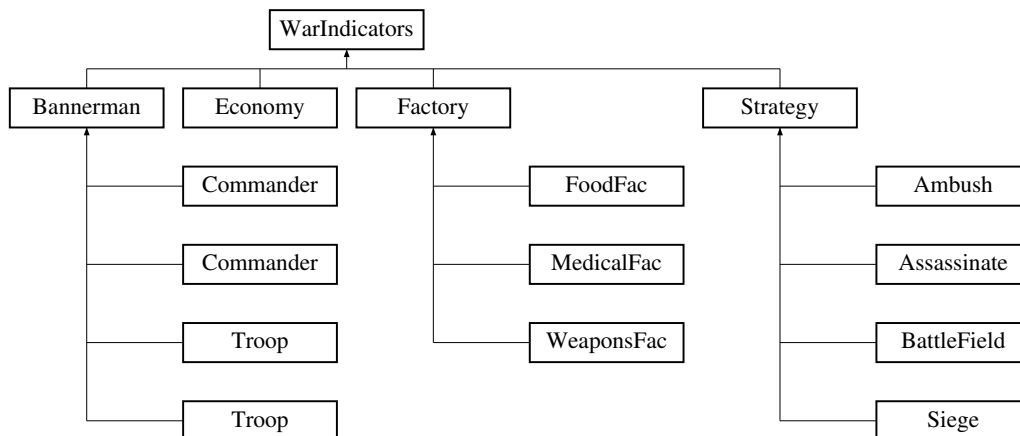
- UnstableState.h
- UnstableState.cpp

## 4.37 WarIndicators Class Reference

An interface class for all classes that communicate with each other through [Treasury](#) mediator.

```
#include <WarIndicators.h>
```

Inheritance diagram for WarIndicators:



## Public Member Functions

- **WarIndicators ()**  
*Default constructor. initializes m to null.*
- **WarIndicators (Treasury \*m)**  
*constructor. initializes m to the passed in Treasury object.*
- void **setTreasury (Treasury \*m)**  
*sets m (Treasury mediator pointer) to the passed in Treasury pointer.*
- virtual **~WarIndicators ()**  
*destructor. deallocates Treasury pointer of this class*

## Protected Attributes

- [Treasury](#) \* m

### 4.37.1 Detailed Description

An interface class for all classes that communicate with each other through [Treasury](#) mediator.

#### Author

Morgan Bentley

#### Date

October 2022

### 4.37.2 Constructor & Destructor Documentation

#### 4.37.2.1 WarIndicators()

```
WarIndicators::WarIndicators (
    Treasury * m ) [inline]
```

constructor. initializes m to the passed in [Treasury](#) object.

#### Parameters

<i>m</i>	- mediator which is of <a href="#">Treasury</a> pointer type
----------	--

### 4.37.3 Member Function Documentation

#### 4.37.3.1 setTreasury()

```
void WarIndicators::setTreasury (
    Treasury * m ) [inline]
```

sets m ([Treasury](#) mediator pointer) to the passed in [Treasury](#) pointer.

#### Parameters

<i>m</i>	- mediator which is of <a href="#">Treasury</a> pointer type
----------	--

### 4.37.4 Member Data Documentation

#### 4.37.4.1 m

`Treasury* WarIndicators::m [protected]`

`Treasury` mediator pointer

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

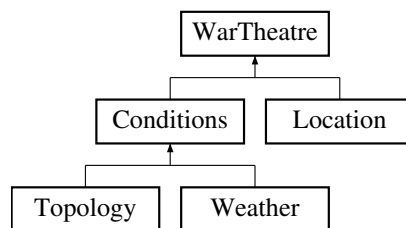
- WarIndicators.h

## 4.38 WarTheatre Class Reference

Abstract object. This is the Component participant in the decorator pattern.

```
#include <WarTheatre.h>
```

Inheritance diagram for WarTheatre:



### Public Member Functions

- **WarTheatre** ()  
*Constructor.*
- `Strategy * getStrategy` ()  
*returns the strategy the location is based on*
- virtual void `sendScout` ()=0  
*Abstract. Decorates the location of the battle.*
- char `decideVenue` (Strategy \*strategy)  
*function that uses the parameter to decide the venue of the war*
- char `getVenue` ()  
*returns the venue the battle is based in*
- void **setStrategy** (Strategy \*myStrat)  
*sets the strategy the location is based on*
- void **setLocation** (string loc)  
*sets the location of the battle*
- void **setDifficulty** (int num)  
*sets the difficulty variable*
- int `getDifficulty` ()  
*returns the difficulty value of the theatre*
- void `setVenue` (char v)  
*sets the venue variable.*

## Private Attributes

- [Strategy](#) \* [strategy](#)
- char [venue](#)
- string [location](#)
- int [difficulty](#) = 1

### 4.38.1 Detailed Description

Abstract object. This is the Component participant in the decorator pattern.

#### Author

Keabetswe Mothapo

#### Date

October 2022

### 4.38.2 Member Function Documentation

#### 4.38.2.1 decideVenue()

```
char WarTheatre::decideVenue (  
    Strategy * strategy )
```

function that uses the parameter to decide the venue of the war

#### Parameters

<i>strategy</i>	- a pointer to the strategy object used in the battle
-----------------	---

#### Returns

char variable of the location. options are a, b, c and d only

#### 4.38.2.2 getDifficulty()

```
int WarTheatre::getDifficulty ( )
```

returns the difficulty value of the theatre

#### Returns

the difficulty value

#### 4.38.2.3 getStrategy()

```
Strategy * WarTheatre::getStrategy ( )
```

returns the strategy the location is based on

##### Returns

the strategy pointer

#### 4.38.2.4 getVenue()

```
char WarTheatre::getVenue ( )
```

returns the venue the battle is based in

##### Returns

the venue variable

#### 4.38.2.5 sendScout()

```
virtual void WarTheatre::sendScout ( ) [pure virtual]
```

Abstract. Decorates the location of the battle.

Implemented in [Conditions](#), [Location](#), [Topology](#), and [Weather](#).

#### 4.38.2.6 setVenue()

```
void WarTheatre::setVenue (
    char v )
```

sets the venue variable.

##### Parameters

<code>v</code>	- the char variable used to set the venue.
----------------	--

### 4.38.3 Member Data Documentation

#### 4.38.3.1 difficulty

```
int WarTheatre::difficulty = 1 [private]
```

Integer representing the difficulty of the battle at the decorated venue

#### 4.38.3.2 location

```
string WarTheatre::location [private]
```

The location of the venue as a string

#### 4.38.3.3 strategy

```
Strategy* WarTheatre::strategy [private]
```

Name of [Strategy](#) as it determines the venue of the battle

#### 4.38.3.4 venue

```
char WarTheatre::venue [private]
```

Char representing the venue. options a to d only

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

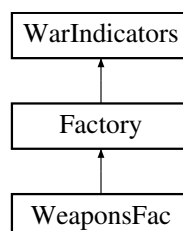
- WarTheatre.h
- WarTheatre.cpp

## 4.39 WeaponsFac Class Reference

[Factory](#) class to serve as a concrete creator class used to create Weapons supplies.

```
#include <WeaponsFac.h>
```

Inheritance diagram for WeaponsFac:



## Public Member Functions

- [ArmySupplies](#) \* `make` ()  
*`make()` is the factory Method called in [operation\(\)](#) to make the Weapon supplies.*
- `~WeaponsFac` ()  
*destructor for the [WeaponsFac](#) class to delete and free the supply variable.*

## Additional Inherited Members

### 4.39.1 Detailed Description

[Factory](#) class to serve as a concrete creator class used to create Weapons supplies.

#### Author

Ronin Brookes 19069686

### 4.39.2 Member Function Documentation

#### 4.39.2.1 `make()`

```
ArmySupplies * WeaponsFac::make ( ) [virtual]
```

`make()` is the factory Method called in [operation\(\)](#) to make the Weapon supplies.

#### Returns

[ArmySupplies](#) pointer to the newly created Weapon supplies.

Implements [Factory](#).

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

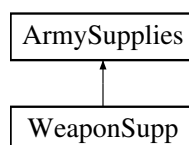
- `WeaponsFac.h`
- `WeaponsFac.cpp`

## 4.40 WeaponSupp Class Reference

Concrete Product [Class](#) containing the information on the Weapon supplies.

```
#include <WeaponSupp.h>
```

Inheritance diagram for `WeaponSupp`:



## Public Member Functions

- int [getAmount](#) ()  
*[getAmount\(\)](#) is used to return the amount of Weapon supplies.*

### 4.40.1 Detailed Description

Concrete Product [Class](#) containing the information on the Weapon supplies.

#### Author

Ronin Brookes 19069686

### 4.40.2 Member Function Documentation

#### 4.40.2.1 [getAmount\(\)](#)

```
int WeaponSupp::getAmount ( ) [virtual]
```

[getAmount\(\)](#) is used to return the amount of Weapon supplies.

#### Returns

an integer representation of the amount of Weapon supplies.

Implements [ArmySupplies](#).

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

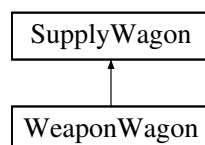
- [WeaponSupp.h](#)
- [WeaponSupp.cpp](#)

## 4.41 WeaponWagon Class Reference

Concrete Prototype class to serve as the wagon for Weapons.

```
#include <WeaponWagon.h>
```

Inheritance diagram for WeaponWagon:





## Public Member Functions

- void [setSup](#) ([ArmySupplies](#) \*sup)  
*initialize the supp variable for the [WeaponWagon](#).*
- [SupplyWagon](#) \* [clone](#) ()  
*Clone the Weapon Supply Wagon object.*
- [ArmySupplies](#) \* [getSupplies](#) ()  
*Return the supp variable for the Weapon Wagon.*
- ~[WeaponWagon](#) ()  
*destructor for the Weapon Wagon class to delete and free the supply variable.*

## Public Attributes

- [ArmySupplies](#) \* [supp](#)

### 4.41.1 Detailed Description

Concrete Prototype class to serve as the wagon for Weapons.

Author

Ronin Brookes 19069686

### 4.41.2 Member Function Documentation

#### 4.41.2.1 clone()

```
SupplyWagon * WeaponWagon::clone ( ) [virtual]
```

Clone the Weapon Supply Wagon object.

Returns

the new Weapon Supply Wagon clone.

Implements [SupplyWagon](#).

#### 4.41.2.2 getSupplies()

```
ArmySupplies * WeaponWagon::getSupplies ( ) [virtual]
```

Return the supp variable for the Weapon Wagon.

Returns

returns the supp variable. Returns the necessary supplies.

Implements [SupplyWagon](#).

#### 4.41.2.3 setSup()

```
void WeaponWagon::setSup (  
    ArmySupplies * sup ) [virtual]
```

initialize the supp variable for the [WeaponWagon](#).

## Parameters

<code>sup</code>	is used to set the supp variable,
------------------	-----------------------------------

Implements [SupplyWagon](#).

### 4.41.3 Member Data Documentation

#### 4.41.3.1 `supp`

[ArmySupplies](#)\* `WeaponWagon::supp`

pointer to the supplies in the Weapon wagon

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

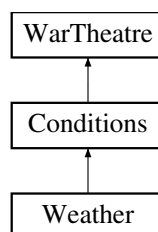
- `WeaponWagon.h`
- `WeaponWagon.cpp`

## 4.42 Weather Class Reference

concrete decorator A. This is the class implements climate effects to the war location. Inherits from [Conditions](#) class

```
#include <Weather.h>
```

Inheritance diagram for Weather:



### Public Member Functions

- [Weather](#) ([WarTheatre](#) \*myTheatre)  
*Costructor of the weather object.*
- int [calcEffect](#) ()  
*the function that uses the weather components to create a wholistic effect on bannermen*
- void [setTemp](#) (double t)  
*setter for the temperature variable*
- void [makeltRain](#) ()  
*setter for the rain variable makes the boolean variable true*
- void [setWindSpeed](#) (int SP)  
*setter for the windspeed variable*
- void [weatherReport](#) ()  
*displays the weather conditions of the location*
- void [sendScout](#) ()  
*calls the calEffect function and affects bannermen*

## Private Attributes

- double `temp`
- bool `rain`
- int `windspeed`

### 4.42.1 Detailed Description

concrete decorator A. This is the class implements climate effects to the war location. Inherits from [Conditions](#) class

#### Author

Keabetswe Mothapo

#### Date

October 2022

### 4.42.2 Constructor & Destructor Documentation

#### 4.42.2.1 Weather()

```
Weather::Weather (
    WarTheatre * myTheatre )
```

Costructor of the weather object.

#### Parameters

<code>val</code>	- recieves a generated value which will be used for setters in this class
------------------	---

### 4.42.3 Member Function Documentation

#### 4.42.3.1 calcEffect()

```
int Weather::calcEffect ( )
```

the function that uses the weather components to create a wholistic effect on bannermen

#### Returns

returns the adjusted difficulty value

#### 4.42.3.2 sendScout()

```
void Weather::sendScout ( ) [virtual]
```

calls the calEffect function and affects bannermen

Reimplemented from [Conditions](#).

#### 4.42.3.3 setTemp()

```
void Weather::setTemp (
    double t )
```

setter for the temperature variable

##### Parameters

<i>t</i>	- the value used to set the temp
----------	----------------------------------

#### 4.42.3.4 setWindSpeed()

```
void Weather::setWindSpeed (
    int SP )
```

setter for the windspeed variable

##### Parameters

<i>SP</i>	- the value used to set the wind speed
-----------	--

### 4.42.4 Member Data Documentation

#### 4.42.4.1 rain

```
bool Weather::rain [private]
```

Boolean to indicate whether it is raining or not

#### 4.42.4.2 temp

```
double Weather::temp [private]
```

Value of the temperature at the venue

#### 4.42.4.3 windspeed

```
int Weather::windspeed [private]
```

Value of the wind speed at the venue

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

- Weather.h
- Weather.cpp



## Chapter 5

# File Documentation

### 5.1 Ambush.h

```
1 #ifndef AMBUSH_H
2 #define AMBUSH_H
3 #include "Strategy.h"
4
10 class Ambush :public Strategy {
11 private:
13     int stealth;
14 public:
28     Ambush(int stealth, Kingdom* myKingdom, Kingdom* enemyKingdom, Bannerman* myBannerman, Bannerman*
        enemyBannerman, string name, int min, int minFavour, Historian* h, HistoryBook* hb);
34     bool attack(Bannerman* myBannerman, Bannerman* enemyBannerman);
36     ~Ambush();
37 };
38
39 #endif
```

### 5.2 ArmySupplies.h

```
1 #ifndef ARMYSUPPLIES_H
2 #define ARMYSUPPLIES_H
3
6 class ArmySupplies {
7
8 public:
9
12     virtual int getAmount() = 0;
13 };
14
15 #endif
```

### 5.3 Assassinate.h

```
1 #ifndef ASSASSINATE_H
2 #define ASSASSINATE_H
3 #include "Strategy.h"
4
10 class Assassinate :public Strategy {
11 private:
14     int stealth;
16     bool alive;
17 public:
32     Assassinate(int stealth, bool alive, Kingdom* myKingdom, Kingdom* enemyKingdom, Bannerman* myBannerman,
        Bannerman* enemyBannerman, string name, int min, int minFavour, Historian* h, HistoryBook* hb);
38     bool attack(Bannerman* myBannerman, Bannerman* enemyBannerman);
40     ~Assassinate();
41 };
42
43 #endif
```

## 5.4 Bannerman.h

```

1  #ifndef BANNERMAN_H
2  #define BANNERMAN_H
3  #include "Strategy.h"
4  #include "MasterOfCoin.h"
5  #include "Raven.h"
6  #include "WarIndicators.h"
7  #include "WarTheatre.h"
8  #include <string>
9  #include <list>
10
11 class MasterOfCoin;
12
13 class Strategy;
14
15 class Bannerman: public WarIndicators {
16
17 protected:
18
19     string name;
20
21     int favour;
22
23     int numWeapons;
24
25     int damage;
26
27     int numFood;
28
29     int numMedical;
30
31     list<Raven*> ravenList;
32
33     MasterOfCoin* m;
34
35     Strategy* strategy;
36
37     int HP;
38
39 public:
40     bool assassin;
41     Bannerman();
42
43     virtual void increaseFavour() = 0;
44
45     virtual void decreaseFavour() = 0;
46
47     virtual void attach(Raven* o)=0;
48
49     virtual void detach(Raven* o)=0;
50
51     virtual void increaseHP(int boost) = 0; //implement
52
53     virtual void changeStrategy(Strategy* strategy) = 0;
54
55     virtual void attack(Bannerman* myBannerman, Bannerman* enemyBannerman) = 0;
56
57     virtual void increasePower(int boost) = 0;
58
59     virtual string getName() = 0;
60
61     virtual int getHP() = 0;
62
63     virtual int getDamage() = 0;
64
65     virtual void receiveDamage(int boost) = 0;
66
67     virtual void decreasePower(int x) = 0;
68
69     virtual void decreaseWeapons() = 0;
70
71     virtual void decreaseFood() = 0;
72
73     virtual void decreaseMedical() = 0;
74
75     virtual int getWeapons() = 0;
76
77     virtual int getFood() = 0;
78
79     virtual int getMedical() = 0;
80
81     virtual void setWeapons(int numWeapons) = 0;
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157

```



```

161     virtual void setFood(int numFood) = 0;
162
166     virtual void setMedical(int numMedical) = 0;
167
171     virtual void setRaven(list<Raven*> r) = 0;
172
176     virtual void setMaster(MasterOfCoin* m) = 0;
177
180     virtual int getFavour() = 0;
184     ~Bannerman();
185
188     virtual list<Bannerman*> getTroops() = 0;
189 };
190
191 #endif

```

## 5.5 BattleField.h

```

1 #ifndef BATTLEFIELD_H
2 #define BATTLEFIELD_H
3 #include "Strategy.h"
4
10 class BattleField :public Strategy {
11
12 public:
25     BattleField(Kingdom* myKingdom, Kingdom* enemyKingdom, Bannerman* myBannerman, Bannerman*
        enemyBannerman, string name, int min, int minFavour, Historian* h, HistoryBook* hb);
31     bool attack(Bannerman* myBannerman, Bannerman* enemyBannerman);
33     ~BattleField();
34 };
35
36 #endif

```

## 5.6 Class.h

```

1 #ifndef CLASS_H
2 #define CLASS_H
3
4 class Class {
5 };
6
7 #endif

```

## 5.7 Commander.h

```

1 #ifndef COMMANDER_H
2 #define COMMANDER_H
3
4 #include "Bannerman.h"
5 #include "ConIterator.h"
6 #include <list>
7 class Iterator;
8 using namespace std;
9
15 class Commander : public Bannerman
16 {
17 private:
20     list<Bannerman *> groundForces;
21
23     string name;
24
25 public:
29     Commander(string name);
30
33     Iterator *createIterator();
34
37     void removeBannerman(Bannerman *x);
38
43     void attack(Bannerman *myBannerman, Bannerman *enemyBannerman) override;
44
47     void addBannerman(Bannerman *b);
48
50     list<Bannerman *> getTroops();
51
57     int getHP() override;

```

```

58
64     int getDamage() override;
68     void attach(Raven *o) override;
69
73     void detach(Raven *o) override;
74
79     void receiveDamage(int x) override;
82     void decreaseWeapons() override;
85     void decreaseFood() override;
88     void decreaseMedical() override;
89
93     void increasePower(int boost) override;
94
100     int getWeapons() override;
101
107     int getFood() override;
108
114     int getMedical() override;
115
119     void increaseHP(int boost) override;
120
124     void changeStrategy(Strategy *strategy) override;
125
128     void increaseFavour() override;
129
132     void decreaseFavour() override;
133
137     int getFavour() override;
138
139
143     void setWeapons(int numWeapons) override;
144
148     void setFood(int numFood) override;
149
153     void setMedical(int numMedical) override;
154
158     void decreasePower(int x) override;
159
162     ~Commander();
163
164     // Julianna added:
165
169     void setRaven(list<Raven *> r) override;
170
175     void setMaster(MasterOfCoin *m) override;
176
180     void setStrategy(Strategy *s);
181
185 string getName();
186 };
187
188 #endif

```

## 5.8 Conditions.h

```

1 #ifndef CONDITIONS_H
2 #define CONDITIONS_H
3 #include <cstdlib>
4 #include "WarTheatre.h"
5 // #include "Weather.h"
6 // #include "Topology.h"
7 #include "Location.h"
8
15 class Conditions : public WarTheatre {
16
17 private:
19     WarTheatre* myVenue;
20
21 public:
22
24     void sendScout() ;
25
30     Conditions(WarTheatre* myVenue);
31
33     virtual ~Conditions();
34 };
35
36 #endif

```

## 5.9 Conlterator.h

```

1  #ifndef CONITERATOR_H
2  #define CONITERATOR_H
3  #include "Bannerman.h"
4  #include "Iterator.h"
5  #include <list>
6  using namespace std;
7
14 class ConIterator : public Iterator {
15 private:
18     list<Bannerman*> armyList;
19
23     list<Bannerman*>::iterator it;
24 public:
29     ConIterator(list<Bannerman*> X);
30
36     Bannerman* Current();
37
38
44     Bannerman* next();
45
51     bool hasNext();
52
57     bool isActive();
58 };
59
60 #endif

```

## 5.10 Economy.h

```

1  #ifndef ECONOMY_H
2  #define ECONOMY_H
3  #include "WarIndicators.h"
4
5  #include <iostream>
6  #include "State.h"
7
8  using namespace std;
9
10 class State;
16 class Economy :public WarIndicators {
17
18 private:
20     State* state;
22     int currency;
23
24 public:
29     Economy(State* state,int currency);
31     void SetState();
34     State* getState();
37     void decreaseCurrency();
40     int getCurrency();
43     void removeCurrency(int i);
45     virtual ~Economy();
46 };
47 #endif

```

## 5.11 Factory.h

```

1
2  #ifndef FACTORY_H
3  #define FACTORY_H
4  #include "WarIndicators.h"
5  #include "ArmySupplies.h"
6  #include <iostream>
7  using namespace std;
10 class Factory : public WarIndicators {
11
12 protected:
15     virtual ArmySupplies* make() = 0;
16 public:
18     void operation();
21     ArmySupplies* getSupply();
23     virtual ~Factory();
24 private:
26     ArmySupplies* supply;
27 };
28
29 #endif

```

## 5.12 FailedState.h

```
1 #ifndef FAILEDSTATE_H
2 #define FAILEDSTATE_H
3 #include "State.h"
4 #include "Economy.h"
5
11 class FailedState :public State {
12
13
14 public:
17     FailedState();
19     void decreaseCurrency();
22     string getState();
23 };
24
25 #endif
```

## 5.13 FoodFac.h

```
1 #ifndef FOODFAC_H
2 #define FOODFAC_H
3 #include "Factory.h"
4 #include "ArmySupplies.h"
5 #include "FoodSupp.h"
6
9 class FoodFac : public Factory {
10
11
12 public:
15     ArmySupplies* make();
17     ~FoodFac();
18 };
19
20 #endif
```

## 5.14 FoodSupp.h

```
1 #ifndef FOODSUPP_H
2 #define FOODSUPP_H
3 #include "ArmySupplies.h"
4
7 class FoodSupp : public ArmySupplies {
8
9 public:
12     int getAmount();
13 };
14
15 #endif
16
```

## 5.15 FoodWagon.h

```
1 #ifndef FOODWAGON_H
2 #define FOODWAGON_H
3 #include "Factory.h"
4 #include "SupplyWagon.h"
5 #include "ArmySupplies.h"
6 #include "FoodSupp.h"
7
10 class FoodWagon : public SupplyWagon {
11
12 public:
15     void setSup(ArmySupplies* sup);
18     SupplyWagon* clone();
21     ArmySupplies* getSupplies();
23     ArmySupplies* supp=NULL;
25     ~FoodWagon();
26 };
27
28 #endif
```

## 5.16 HealthyState.h

```

1 #ifndef HEALTHYSTATE_H
2 #define HEALTHYSTATE_H
3 #include "State.h"
4 #include "Economy.h"
5 #include "UnstableState.h"
6
12 class HealthyState :public State {
13
14
15 public:
16     HealthyState();
21     void decreaseCurrency();
24     virtual State* getDemotionState();
27     string getState();
28 };
29
30 #endif

```

## 5.17 Historian.h

```

1 #ifndef HISTORIAN_H
2 #define HISTORIAN_H
3 #include "Bannerman.h"
4 #include "History.h"
5 class History;
6 class Historian {
11 private:
12     Bannerman* bannerman;
13
14
15 public:
16     void setAlly(Bannerman* b);
20
21     History* Store();
25
26     Bannerman* restoreAlly(History* h);
32
33     ~Historian(){};
34 };
35
36 #endif

```

## 5.18 History.h

```

1 #ifndef HISTORY_H
2 #define HISTORY_H
3 #include "Bannerman.h"
4 #include <iostream>
5 using namespace std;
6 class Bannerman;
7 class History { //memento
12 private:
13     Bannerman* bannerman;
14
15
16 public:
17     History(Bannerman* b);
21
22     Bannerman* getBannerman();
27
28     ~History(){};
29 };
30
31 #endif

```

## 5.19 HistoryBook.h

```

1 #ifndef HISTORYBOOK_H
2 #define HISTORYBOOK_H
3 #include "History.h"
4 #include <list>
5 class History;
6 class HistoryBook {
11 private:
12     list<History*> defectedAllies;

```

```

14
15 public:
20     void add(History* h);
21
26     History* getAlly();
27 };
28
29 #endif

```

## 5.20 Iterator.h

```

1 #ifndef ITERATOR_H
2 #define ITERATOR_H
3 #include "Bannerman.h"
4
11 class Iterator {
12
13
14 public:
20     virtual Bannerman* Current()=0;
26     virtual bool hasNext() = 0;
27
33     virtual Bannerman* next() = 0;
34
39     virtual bool isActive() = 0;
40 };
41
42 #endif

```

## 5.21 Kingdom.h

```

1 #ifndef KINGDOM_H
2 #define KINGDOM_H
3
4 #include <list>
5 #include "Bannerman.h"
6 #include "Economy.h"
7
8 using namespace std;
9
10 class Bannerman;
11
16 class Kingdom {
17
18 private:
20     list<Bannerman*> bannerman;
22     Economy* economy;
23
24 public:
28     Kingdom(Economy* economy);
32     void remove(Bannerman* b);
36     void add(Bannerman* b);
38     virtual ~Kingdom();
41     int getSize();
45     Bannerman* getAlly(string n);
48     list<Bannerman*> getKingdom();
49 };
50
51 #endif

```

## 5.22 Location.h

```

1 #ifndef LOCATION_H
2 #define LOCATION_H
3 #include <iostream>
4 #include "WarTheatre.h"
5 #include "Conditions.h" //causes circular dependency?
6
7 using namespace std;
8
15 class Location : public WarTheatre {
16
17
18 public:
19

```

```
21     Location();
22
24     void sendScout();
25
26
27 };
28
29 #endif
```

## 5.23 MasterOfCoin.h

```
1 #ifndef MASTEROFCOIN_H
2 #define MASTEROFCOIN_H
3
4 #include "Economy.h"
5 #include "Raven.h"
6 #include "Strategy.h"
7 #include "Treasury.h"
8 #include "WarIndicators.h"
9
10 class Strategy;
11 class MasterOfCoin : public Treasury {
12
13 private:
14     Economy* economy;
15
16     Raven* observer;
17
18     Strategy* strategy;
19
20 public:
21     MasterOfCoin(Economy* economy, Raven* observer, Strategy* strategy);
22
23     void notify(WarIndicators* sender);
24
25     void decreaseCurrency();
26
27     void manufacture();
28
29     ~MasterOfCoin();
30 };
31
32 #endif
```

## 5.24 MedicalFac.h

```
1 #ifndef MEDICALFAC_H
2 #define MEDICALFAC_H
3 #include "Factory.h"
4 #include "ArmySupplies.h"
5 #include "MedicalSupp.h"
6
7 class MedicalFac : public Factory {
8
9 public:
10     ArmySupplies* make();
11     ~MedicalFac();
12 };
13
14 #endif
```

## 5.25 MedicalSupp.h

```
1 #ifndef MEDICALSUPP_H
2 #define MEDICALSUPP_H
3 #include "ArmySupplies.h"
4
5 class MedicalSupp : public ArmySupplies {
6
7 public:
8     int getAmount();
9 };
10
11 #endif
```

## 5.26 MedicalWagon.h

```
1 #ifndef MEDICALWAGON_H
2 #define MEDICALWAGON_H
3 #include "Factory.h"
4 #include "SupplyWagon.h"
5 #include "ArmySupplies.h"
6 #include "MedicalSupp.h"
7
10 class MedicalWagon : public SupplyWagon {
11
12 public:
15     void setSup(ArmySupplies* sup);
18     SupplyWagon* clone();
21     ArmySupplies* getSupplies();
23     ArmySupplies* supp;
25     ~MedicalWagon();
26 };
27
28 #endif
```

## 5.27 Raven.h

```
1 #ifndef RAVEN_H
2 #define RAVEN_H
3
9 class Raven {
10
11
12 public:
14     virtual void update() = 0;
15 };
16
17 #endif
```

## 5.28 sendRaven.h

```
1 #ifndef SENDRAVEN_H
2 #define SENDRAVEN_H
3
4 #include "SupplyWagon.h"
5 #include "Commander.h"
6 #include "MasterOfCoin.h"
7 #include "Raven.h"
8 #include "Bannerman.h"
9
16 class sendRaven : public Raven {
17
18 private:
19
21     int numFood;
22
24     int numMedical;
25
27     int numWeapons;
28
30     SupplyWagon** supplies;
31
33     Bannerman* subject;
34
35 public:
39     sendRaven(SupplyWagon** supplies, Bannerman* subject);
40
42     void update();
43
45     void checkSupplies();
46
48     ~sendRaven();
49 };
50
51 #endif
```

## 5.29 Siege.h

```
1 #ifndef SIEGE_H
```



```

2 #define SIEGE_H
3 #include "Strategy.h"
4
10 class Siege :public Strategy {
11 private:
12     int stealth;
13 public:
21     Siege(int stealth, Kingdom* myKingdom, Kingdom* enemyKingdom, Bannerman* myBannerman, Bannerman*
        enemyBannerman, string name, int min, int minFavour, Historian* h, HistoryBook* hb);
27     bool attack(Bannerman* myBannerman, Bannerman* enemyBannerman);
29     ~Siege();
30 };
31
32 #endif

```

## 5.30 State.h

```

1 #ifndef STATE_H
2 #define STATE_H
3 #include "Economy.h"
4 class Economy;
5 using namespace std;
6
12 class State {
13
14 protected:
15     Economy* context;
16
17 public:
21     State();
24     virtual void setContext(Economy* context);
26     virtual void decreaseCurrency()=0;
29     virtual State* getDemotionState();
31     virtual ~State();
34     virtual string getState()=0;
35 };
36
37 #endif

```

## 5.31 Strategy.h

```

1 #ifndef STRATEGY_H
2 #define STRATEGY_H
3
4 #include <string>
5 #include <ctime>
6 #include "Bannerman.h"
7 #include "WarIndicators.h"
8 #include "MasterOfCoin.h"
9 #include "Kingdom.h"
10 #include "HistoryBook.h"
11 #include "Historian.h"
12
13 using namespace std;
14 class Historian;
15 class HistoryBook;
16 class Bannerman;
17 class Kingdom;
24 class Strategy : public WarIndicators {
25
26 protected:
28     Kingdom* myKingdom;
30     Kingdom* enemyKingdom;
32     Bannerman* myBannerman;
34     Bannerman* enemyBannerman;
36     string strategy;
38     int minFavour;
40     int minSupplies;
42     HistoryBook* BookOfDura;
44     Historian* Greg;
46     int defectedAllies;
47 public:
60     Strategy(Kingdom* myKingdom, Kingdom* enemyKingdom, Bannerman* myBannerman, Bannerman*
        enemyBannerman, string name, int min, int minFavour, Historian* h, HistoryBook* hb);
66     virtual bool attack(Bannerman* myBannerman, Bannerman* enemyBannerman)=0;
69     virtual string getStrategyName();
72     virtual Bannerman* getMyBannerman();
75     virtual Bannerman* getEnemyBannerman();
77     virtual ~Strategy();
78 };
79 #endif

```

## 5.32 SupplyWagon.h

```
1 #ifndef SUPPLYWAGON_H
2 #define SUPPLYWAGON_H
3 #include "ArmySupplies.h"
4
5
6
7 class SupplyWagon {
8
9 public:
10
11
12
13     virtual void setSup(ArmySupplies* sup)=0;
14     virtual SupplyWagon* clone() = 0;
15     virtual ArmySupplies* getSupplies()=0;
16     ArmySupplies* supp;
17     //virtual ~SupplyWagon();
18
19 };
20
21 #endif
```

## 5.33 Topology.h

```
1 #ifndef TOPOLOGY_H
2 #define TOPOLOGY_H
3 #include "Conditions.h"
4
5
6
7 class Topology : public Conditions {
8
9 public:
10
11
12
13     void setTopology();
14
15     Topology(WarTheatre* myTheatre);
16
17     void sendScout();
18 };
19
20 #endif
```

## 5.34 Treasury.h

```
1 #ifndef TREASURY_H
2 #define TREASURY_H
3 #include "WarIndicators.h"
4
5 class WarIndicators;
6
7 class Treasury {
8
9 public:
10     virtual void notify(WarIndicators* sender) = 0;
11 };
12
13 #endif
```

## 5.35 Troop.h

```
1 #ifndef TROOP_H
2 #define TROOP_H
3 #include "Kingdom.h"
4 #include "Strategy.h"
5 #include "Bannerman.h"
6
7
8
9 class Troop : public Bannerman {
10
11 private:
12
13     int size;
14
15     Strategy* strategy;
16
17     int damage;
18
19     WarTheatre* warZone;
20 public:
```

```

32     Troop(string name, int favor, int numFood, int Medical, int
HP,WarTheatre*warZone,Strategy*strategy,MasterOfCoin* m, bool assassin, int size);
33
39     int getHP() override;
40
46     int getSize();
47
51     void attach(Raven* o) override;
52
56     void detach(Raven* o) override;
57
61     void increasePower(int boost) override;
62
67     void attack(Bannerman* myBannerman, Bannerman* enemyBannerman) override;
68
71     void decreaseWeapons() override;
72
75     void decreaseFood() override;
76
79     void decreaseMedical() override;
80
84     void changeStrategy(Strategy* strategy) override;
85
88     void increaseFavour() override;
89
92     void decreaseFavour() override;
93
97     int getFavour() override;
98
102    void increaseHP(int boost) override;
108    int getDamage() override;
109
114    void receiveDamage(int X) override;
115
121    int getWeapons() override;
122
128    int getFood() override;
129
135    int getMedical() override;
136
140    void setWeapons(int numWeapons) override;
141
145    void setFood(int numFood) override;
146
147
148
152    void decreasePower(int x) override;
153
154
158    void setMedical(int numMedical) override;
159
160
164    void setRaven(list<Raven*> r) override;
165
169    void setMaster(MasterOfCoin* m) override;
170
173    ~Troop();
177    string getName() override;
182    list<Bannerman*> getTroops() override;
183 };
184
185 #endif

```

## 5.36 UnstableState.h

```

1 #ifndef UNSTABLESTATE_H
2 #define UNSTABLESTATE_H
3 #include "State.h"
4 #include "Economy.h"
5 #include "FailedState.h"
6
12 class UnstableState :public State {
13
14
15 public:
18     UnstableState();
21     void decreaseCurrency();
24     State* getDemotionState();
27     string getState();
28 };
29
30 #endif

```

## 5.37 WarIndicators.h

```

1 #ifndef WARINDICATORS_H
2 #define WARINDICATORS_H
3 #include "Treasury.h"
4 class Treasury;
10 class WarIndicators {
11
12 protected:
14     Treasury* m;
15
16 public:
19     WarIndicators(){
20         m = nullptr;
21     }
25     WarIndicators(Treasury* m){
26         this->m=m;
27     }
31     void setTreasury(Treasury* m){
32         this->m=m;
33     }
34
36     virtual ~WarIndicators(){
37         //delete m;
38     }
39
40
41 };
42
43 #endif

```

## 5.38 WarTheatre.h

```

1 #ifndef WARTHEATRE_H
2 #define WARTHEATRE_H
3 #include "Strategy.h"
4 #include <string>
5 #include <iostream>
6
7 using namespace std;
8
9 class Strategy;
16 class WarTheatre {
17
18 private:
20     Strategy* strategy;
21
23     char venue;
24
26     string location;
27
29     int difficulty = 1;
30
31
32 public:
34     WarTheatre();
35
38     Strategy* getStrategy(); //may need to make sendScout the virtual one instead
39
41     virtual void sendScout() = 0;
42
47     char decideVenue(Strategy* strategy);
48
51     char getVenue();
52
54     void setStrategy(Strategy* myStrat);
55
57     void setLocation(string loc);
58
60     void setDifficulty(int num);
61
64     int getDifficulty();
68     void setVenue(char v);
69 };
70
71 #endif

```

## 5.39 WeaponsFac.h

```

1 #ifndef WEAPONSFAC_H

```

```

2 #define WEAPONSFAC_H
3 #include "Factory.h"
4 #include "ArmySupplies.h"
5 #include "WeaponSupp.h"
6
9 class WeaponsFac : public Factory {
10
11 public:
12
15     ArmySupplies* make();
16
18     ~WeaponsFac();
19 };
20
21 #endif

```

## 5.40 WeaponSupp.h

```

1 #ifndef WEAPONSUPP_H
2 #define WEAPONSUPP_H
3 #include "ArmySupplies.h"
4
7 class WeaponSupp : public ArmySupplies {
8
9 public:
12     int getAmount();
13 };
14
15 #endif

```

## 5.41 WeaponWagon.h

```

1 #ifndef WEAPONWAGON_H
2 #define WEAPONWAGON_H
3 #include "Factory.h"
4 #include "SupplyWagon.h"
5 #include "ArmySupplies.h"
6 #include "WeaponSupp.h"
7
10 class WeaponWagon : public SupplyWagon {
11
12 public:
13
16     void setSup(ArmySupplies* sup);
19     SupplyWagon* clone();
22     ArmySupplies* getSupplies();
24     ArmySupplies* supp;
26     ~WeaponWagon();
27 };
28
29 #endif

```

## 5.42 Weather.h

```

1 #ifndef WEATHER_H
2 #define WEATHER_H
3 #include "Conditions.h"
4 #include <ctime>
5
12 class Weather:public Conditions{
13
14 private:
16     double temp;
17
19     bool rain;
20
22     int windspeed;
23
24 public:
27     Weather(WarTheatre* myTheatre);
28
31     int calcEffect();
32
35     void setTemp(double t);
36

```

```
39     void makeItRain();
40
43     void setWindSpeed(int SP);
44
46     void weatherReport(); //ask Jules
47
49     void sendScout();
50 };
51
52 #endif
```

# Index

- ~Troop
  - Troop, [89](#)
- add
  - HistoryBook, [58](#)
  - Kingdom, [62](#)
- alive
  - Assassinate, [14](#)
- Ambush, [9](#)
  - Ambush, [10](#)
  - attack, [11](#)
  - stealth, [11](#)
- armyList
  - Conlterator, [43](#)
- ArmySupplies, [11](#)
  - getAmount, [12](#)
- assassin
  - Bannerman, [25](#)
- Assassinate, [12](#)
  - alive, [14](#)
  - Assassinate, [13](#)
  - attack, [14](#)
  - stealth, [14](#)
- attach
  - Bannerman, [17](#)
  - Commander, [31](#)
  - Troop, [89](#)
- attack
  - Ambush, [11](#)
  - Assassinate, [14](#)
  - Bannerman, [17](#)
  - BattleField, [28](#)
  - Commander, [31](#)
  - Siege, [75](#)
  - Strategy, [80](#)
  - Troop, [89](#)
- Bannerman, [15](#)
  - assassin, [25](#)
  - attach, [17](#)
  - attack, [17](#)
  - changeStrategy, [17](#)
  - damage, [25](#)
  - decreaseFavour, [18](#)
  - decreaseFood, [18](#)
  - decreaseMedical, [18](#)
  - decreasePower, [18](#)
  - decreaseWeapons, [18](#)
  - detach, [19](#)
  - favour, [25](#)
  - getDamage, [19](#)
  - getFavour, [19](#)
  - getFood, [19](#)
  - getHP, [20](#)
  - getMedical, [20](#)
  - getName, [20](#)
  - getTroops, [20](#)
  - getWeapons, [21](#)
  - HP, [25](#)
  - increaseFavour, [21](#)
  - increaseHP, [21](#)
  - increasePower, [22](#)
  - m, [25](#)
  - name, [25](#)
  - numFood, [25](#)
  - numMedical, [26](#)
  - numWeapons, [26](#)
  - ravenList, [26](#)
  - receiveDamage, [22](#)
  - setFood, [22](#)
  - setMaster, [22](#)
  - setMedical, [23](#)
  - setRaven, [23](#)
  - setWeapons, [23](#)
  - strategy, [26](#)
- bannerman
  - Historian, [56](#)
  - History, [57](#)
  - Kingdom, [64](#)
- BattleField, [26](#)
  - attack, [28](#)
  - BattleField, [27](#)
- BookOfDura
  - Strategy, [82](#)
- calcEffect
  - Weather, [109](#)
- changeStrategy
  - Bannerman, [17](#)
  - Commander, [31](#)
  - Troop, [90](#)
- Class, [28](#)
- clone
  - FoodWagon, [51](#)
  - MedicalWagon, [70](#)
  - SupplyWagon, [84](#)
  - WeaponWagon, [107](#)
- Commander, [28](#)
  - attach, [31](#)
  - attack, [31](#)

- changeStrategy, 31
- Commander, 30
- decreaseFavour, 32
- decreaseFood, 32
- decreaseMedical, 32
- decreasePower, 32
- decreaseWeapons, 33
- detach, 33
- getDamage, 33
- getFavour, 33
- getFood, 33
- getHP, 34
- getMedical, 34
- getName, 34
- getTroops, 34
- getWeapons, 35
- increaseFavour, 35
- increaseHP, 35
- increasePower, 35
- receiveDamage, 36
- setFood, 36
- setMaster, 36
- setMedical, 37
- setRaven, 37
- setStrategy, 37
- setWeapons, 38
- Conditions, 38
  - Conditions, 39
  - myVenue, 40
  - sendScout, 39
- ConIterator, 40
  - armyList, 43
  - ConIterator, 41
  - Current, 41
  - hasNext, 41
  - isActive, 42
  - it, 43
  - next, 42
- context
  - State, 78
- currency
  - Economy, 45
- Current
  - ConIterator, 41
  - Iterator, 60
- damage
  - Bannerman, 25
  - Troop, 96
- decideVenue
  - WarTheatre, 102
- decreaseCurrency
  - FailedState, 48
  - HealthyState, 53
  - State, 77
  - UnstableState, 98
- decreaseFavour
  - Bannerman, 18
  - Commander, 32
  - Troop, 90
- decreaseFood
  - Bannerman, 18
  - Commander, 32
  - Troop, 90
- decreaseMedical
  - Bannerman, 18
  - Commander, 32
  - Troop, 90
- decreasePower
  - Bannerman, 18
  - Commander, 32
  - Troop, 91
- decreaseWeapons
  - Bannerman, 18
  - Commander, 33
  - Troop, 91
- defectedAllies
  - HistoryBook, 58
  - Strategy, 82
- detach
  - Bannerman, 19
  - Commander, 33
  - Troop, 91
- difficulty
  - WarTheatre, 103
- Economy, 43
  - currency, 45
  - Economy, 44
  - getCurrency, 44
  - getState, 45
  - removeCurrency, 45
  - state, 45
- economy
  - Kingdom, 64
- enemyBannerman
  - Strategy, 82
- enemyKingdom
  - Strategy, 82
- Factory, 46
  - getSupply, 46
  - make, 47
  - supply, 47
- FailedState, 47
  - decreaseCurrency, 48
  - getState, 48
- favour
  - Bannerman, 25
- FoodFac, 49
  - make, 49
- FoodSupp, 50
  - getAmount, 50
- FoodWagon, 51
  - clone, 51
  - getSupplies, 51
  - setSup, 52
  - supp, 52



getAlly  
    HistoryBook, 58  
    Kingdom, 63  
getAmount  
    ArmySupplies, 12  
    FoodSupp, 50  
    MedicalSupp, 69  
    WeaponSupp, 106  
getBannerman  
    History, 57  
getCurrency  
    Economy, 44  
getDamage  
    Bannerman, 19  
    Commander, 33  
    Troop, 91  
getDemotionState  
    HealthyState, 54  
    State, 77  
    UnstableState, 98  
getDifficulty  
    WarTheatre, 102  
getEnemyBannerman  
    Strategy, 81  
getFavour  
    Bannerman, 19  
    Commander, 33  
    Troop, 92  
getFood  
    Bannerman, 19  
    Commander, 33  
    Troop, 92  
getHP  
    Bannerman, 20  
    Commander, 34  
    Troop, 92  
getKingdom  
    Kingdom, 63  
getMedical  
    Bannerman, 20  
    Commander, 34  
    Troop, 92  
getMyBannerman  
    Strategy, 81  
getName  
    Bannerman, 20  
    Commander, 34  
    Troop, 93  
getSize  
    Kingdom, 63  
    Troop, 93  
getState  
    Economy, 45  
    FailedState, 48  
    HealthyState, 54  
    State, 78  
    UnstableState, 98  
getStrategy  
    WarTheatre, 102  
getStrategyName  
    Strategy, 81  
getSupplies  
    FoodWagon, 51  
    MedicalWagon, 70  
    SupplyWagon, 84  
    WeaponWagon, 107  
getSupply  
    Factory, 46  
getTroops  
    Bannerman, 20  
    Commander, 34  
    Troop, 93  
getVenue  
    WarTheatre, 103  
getWeapons  
    Bannerman, 21  
    Commander, 35  
    Troop, 93  
Greg  
    Strategy, 82  
hasNext  
    Conlterator, 41  
    Iterator, 60  
HealthyState, 53  
    decreaseCurrency, 53  
    getDemotionState, 54  
    getState, 54  
Historian, 54  
    bannerman, 56  
    restoreAlly, 55  
    setAlly, 55  
    Store, 55  
History, 56  
    bannerman, 57  
    getBannerman, 57  
    History, 56  
HistoryBook, 58  
    add, 58  
    defectedAllies, 58  
    getAlly, 58  
HP  
    Bannerman, 25  
increaseFavour  
    Bannerman, 21  
    Commander, 35  
    Troop, 94  
increaseHP  
    Bannerman, 21  
    Commander, 35  
    Troop, 94  
increasePower  
    Bannerman, 22  
    Commander, 35  
    Troop, 94  
isActive

- Conlterator, 42
- Iterator, 60
- it
  - Conlterator, 43
- Iterator, 59
  - Current, 60
  - hasNext, 60
  - isActive, 60
  - next, 61
- Kingdom, 61
  - add, 62
  - bannerman, 64
  - economy, 64
  - getAlly, 63
  - getKingdom, 63
  - getSize, 63
  - Kingdom, 62
  - remove, 63
- Location, 64
  - sendScout, 65
- location
  - WarTheatre, 104
- m
  - Bannerman, 25
  - WarIndicators, 101
- make
  - Factory, 47
  - FoodFac, 49
  - MedicalFac, 68
  - WeaponsFac, 105
- MasterOfCoin, 65
  - MasterOfCoin, 66
  - notify, 67
- MedicalFac, 67
  - make, 68
- MedicalSupp, 68
  - getAmount, 69
- MedicalWagon, 69
  - clone, 70
  - getSupplies, 70
  - setSup, 70
  - supp, 71
- minFavour
  - Strategy, 82
- minSupplies
  - Strategy, 83
- myBannerman
  - Strategy, 83
- myKingdom
  - Strategy, 83
- myVenue
  - Conditions, 40
- name
  - Bannerman, 25
- next
  - Conlterator, 42
  - Iterator, 61
- notify
  - MasterOfCoin, 67
  - Treasury, 86
- numFood
  - Bannerman, 25
- numMedical
  - Bannerman, 26
- numWeapons
  - Bannerman, 26
- rain
  - Weather, 110
- Raven, 71
  - update, 72
- ravenList
  - Bannerman, 26
- receiveDamage
  - Bannerman, 22
  - Commander, 36
  - Troop, 95
- remove
  - Kingdom, 63
- removeCurrency
  - Economy, 45
- restoreAlly
  - Historian, 55
- sendRaven, 72
  - sendRaven, 73
  - update, 73
- sendScout
  - Conditions, 39
  - Location, 65
  - Topology, 86
  - WarTheatre, 103
  - Weather, 109
- setAlly
  - Historian, 55
- setContext
  - State, 78
- setFood
  - Bannerman, 22
  - Commander, 36
  - Troop, 95
- setMaster
  - Bannerman, 22
  - Commander, 36
  - Troop, 95
- setMedical
  - Bannerman, 23
  - Commander, 37
  - Troop, 95
- setRaven
  - Bannerman, 23
  - Commander, 37
  - Troop, 96
- setStrategy

- Commander, [37](#)
- setSup
  - FoodWagon, [52](#)
  - MedicalWagon, [70](#)
  - SupplyWagon, [84](#)
  - WeaponWagon, [107](#)
- setTemp
  - Weather, [110](#)
- setTreasury
  - WarIndicators, [100](#)
- setVenue
  - WarTheatre, [103](#)
- setWeapons
  - Bannerman, [23](#)
  - Commander, [38](#)
  - Troop, [96](#)
- setWindSpeed
  - Weather, [110](#)
- Siege, [74](#)
  - attack, [75](#)
  - Siege, [75](#)
  - stealth, [76](#)
- size
  - Troop, [97](#)
- State, [76](#)
  - context, [78](#)
  - decreaseCurrency, [77](#)
  - getDemotionState, [77](#)
  - getState, [78](#)
  - setContext, [78](#)
- state
  - Economy, [45](#)
- stealth
  - Ambush, [11](#)
  - Assassinate, [14](#)
  - Siege, [76](#)
- Store
  - Historian, [55](#)
- Strategy, [79](#)
  - attack, [80](#)
  - BookOfDura, [82](#)
  - defectedAllies, [82](#)
  - enemyBannerman, [82](#)
  - enemyKingdom, [82](#)
  - getEnemyBannerman, [81](#)
  - getMyBannerman, [81](#)
  - getStrategyName, [81](#)
  - Greg, [82](#)
  - minFavour, [82](#)
  - minSupplies, [83](#)
  - myBannerman, [83](#)
  - myKingdom, [83](#)
  - Strategy, [80](#)
  - strategy, [83](#)
- strategy
  - Bannerman, [26](#)
  - Strategy, [83](#)
  - Troop, [97](#)
- WarTheatre, [104](#)
- supp
  - FoodWagon, [52](#)
  - MedicalWagon, [71](#)
  - SupplyWagon, [85](#)
  - WeaponWagon, [108](#)
- supply
  - Factory, [47](#)
  - SupplyWagon, [83](#)
    - clone, [84](#)
    - getSupplies, [84](#)
    - setSup, [84](#)
    - supp, [85](#)
- temp
  - Weather, [110](#)
- Topology, [85](#)
  - sendScout, [86](#)
- Treasury, [86](#)
  - notify, [86](#)
- Troop, [87](#)
  - ~Troop, [89](#)
  - attach, [89](#)
  - attack, [89](#)
  - changeStrategy, [90](#)
  - damage, [96](#)
  - decreaseFavour, [90](#)
  - decreaseFood, [90](#)
  - decreaseMedical, [90](#)
  - decreasePower, [91](#)
  - decreaseWeapons, [91](#)
  - detach, [91](#)
  - getDamage, [91](#)
  - getFavour, [92](#)
  - getFood, [92](#)
  - getHP, [92](#)
  - getMedical, [92](#)
  - getName, [93](#)
  - getSize, [93](#)
  - getTroops, [93](#)
  - getWeapons, [93](#)
  - increaseFavour, [94](#)
  - increaseHP, [94](#)
  - increasePower, [94](#)
  - receiveDamage, [95](#)
  - setFood, [95](#)
  - setMaster, [95](#)
  - setMedical, [95](#)
  - setRaven, [96](#)
  - setWeapons, [96](#)
  - size, [97](#)
  - strategy, [97](#)
  - warZone, [97](#)
- UnstableState, [97](#)
  - decreaseCurrency, [98](#)
  - getDemotionState, [98](#)
  - getState, [98](#)
- update

- Raven, [72](#)
  - sendRaven, [73](#)
- venue
  - WarTheatre, [104](#)
- WarIndicators, [99](#)
  - m, [101](#)
  - setTreasury, [100](#)
  - WarIndicators, [100](#)
- WarTheatre, [101](#)
  - decideVenue, [102](#)
  - difficulty, [103](#)
  - getDifficulty, [102](#)
  - getStrategy, [102](#)
  - getVenue, [103](#)
  - location, [104](#)
  - sendScout, [103](#)
  - setVenue, [103](#)
  - strategy, [104](#)
  - venue, [104](#)
- warZone
  - Troop, [97](#)
- WeaponsFac, [104](#)
  - make, [105](#)
- WeaponSupp, [105](#)
  - getAmount, [106](#)
- WeaponWagon, [106](#)
  - clone, [107](#)
  - getSupplies, [107](#)
  - setSup, [107](#)
  - supp, [108](#)
- Weather, [108](#)
  - calcEffect, [109](#)
  - rain, [110](#)
  - sendScout, [109](#)
  - setTemp, [110](#)
  - setWindSpeed, [110](#)
  - temp, [110](#)
  - Weather, [109](#)
  - windspeed, [110](#)
- windspeed
  - Weather, [110](#)