

COS214PROJECT

1st

Generated by Doxygen 1.9.5

1 Todo List	1
2 Hierarchical Index	3
2.1 Class Hierarchy	3
3 Class Index	5
3.1 Class List	5
4 File Index	7
4.1 File List	7
5 Class Documentation	9
5.1 Ambush Class Reference	9
5.1.1 Detailed Description	10
5.1.2 Constructor & Destructor Documentation	10
5.1.2.1 Ambush()	10
5.1.2.2 ~Ambush()	10
5.1.3 Member Function Documentation	11
5.1.3.1 attack()	11
5.1.4 Member Data Documentation	11
5.1.4.1 stealth	11
5.2 ArmySupplies Class Reference	11
5.3 Assassinate Class Reference	12
5.3.1 Detailed Description	12
5.3.2 Constructor & Destructor Documentation	13
5.3.2.1 Assassinate()	13
5.3.2.2 ~Assassinate()	13
5.3.3 Member Function Documentation	13
5.3.3.1 attack()	13
5.3.4 Member Data Documentation	14
5.3.4.1 alive	14
5.3.4.2 stealth	14
5.4 Bannerman Class Reference	14
5.4.1 Member Function Documentation	16
5.4.1.1 attach()	16
5.4.1.2 attack()	16
5.4.1.3 changeStrategy()	17
5.4.1.4 decreaseFavour()	17
5.4.1.5 decreaseFood()	17
5.4.1.6 decreaseMedical()	17
5.4.1.7 decreasePower()	18
5.4.1.8 decreaseWeapons()	18
5.4.1.9 detach()	18
5.4.1.10 getDamage()	18

5.4.1.11 getFavour()	19
5.4.1.12 getFood()	19
5.4.1.13 getHP()	19
5.4.1.14 getMedical()	19
5.4.1.15 getName()	20
5.4.1.16 getWeapons()	20
5.4.1.17 increaseFavour()	20
5.4.1.18 increaseHP()	20
5.4.1.19 increasePower()	21
5.4.1.20 receiveDamage()	21
5.4.1.21 setFood()	21
5.4.1.22 setMaster()	22
5.4.1.23 setMedical()	22
5.4.1.24 setRaven()	22
5.4.1.25 setWeapons()	22
5.4.2 Member Data Documentation	24
5.4.2.1 assassin	24
5.4.2.2 damage	24
5.4.2.3 favour	24
5.4.2.4 HP	24
5.4.2.5 m	24
5.4.2.6 name	24
5.4.2.7 numFood	25
5.4.2.8 numMedical	25
5.4.2.9 numWeapons	25
5.4.2.10 ravenList	25
5.4.2.11 strategy	25
5.5 BattleField Class Reference	25
5.5.1 Detailed Description	26
5.5.2 Constructor & Destructor Documentation	26
5.5.2.1 BattleField()	26
5.5.2.2 ~BattleField()	27
5.5.3 Member Function Documentation	27
5.5.3.1 attack()	27
5.6 Class Class Reference	27
5.7 Commander Class Reference	28
5.7.1 Detailed Description	29
5.7.2 Constructor & Destructor Documentation	30
5.7.2.1 Commander()	30
5.7.3 Member Function Documentation	30
5.7.3.1 attach()	30
5.7.3.2 attack()	30

5.7.3.3 changeStrategy()	31
5.7.3.4 decreaseFavour()	31
5.7.3.5 decreaseFood()	31
5.7.3.6 decreaseMedical()	31
5.7.3.7 decreasePower()	31
5.7.3.8 decreaseWeapons()	32
5.7.3.9 detach()	32
5.7.3.10 getDamage()	32
5.7.3.11 getFavour()	33
5.7.3.12 getFood()	33
5.7.3.13 getHP()	33
5.7.3.14 getMedical()	33
5.7.3.15 getTroops()	34
5.7.3.16 getWeapons()	34
5.7.3.17 increaseFavour()	34
5.7.3.18 increaseHP()	34
5.7.3.19 increasePower()	34
5.7.3.20 receiveDamage()	35
5.7.3.21 setFood()	35
5.7.3.22 setMaster()	35
5.7.3.23 setMedical()	36
5.7.3.24 setRaven()	36
5.7.3.25 setStrategy()	36
5.7.3.26 setWeapons()	37
5.8 Conditions Class Reference	37
5.8.1 Detailed Description	38
5.8.2 Constructor & Destructor Documentation	38
5.8.2.1 Conditions()	38
5.8.3 Member Function Documentation	38
5.8.3.1 sendScout()	38
5.8.4 Member Data Documentation	38
5.8.4.1 myVenue	39
5.9 Conlterator Class Reference	39
5.9.1 Detailed Description	39
5.9.2 Constructor & Destructor Documentation	40
5.9.2.1 Conlterator()	40
5.9.3 Member Function Documentation	40
5.9.3.1 Current()	40
5.9.3.2 hasNext()	40
5.9.3.3 isActive()	41
5.9.3.4 next()	41
5.9.4 Member Data Documentation	41

5.9.4.1 armyList	41
5.9.4.2 it	41
5.10 Economy Class Reference	42
5.10.1 Member Data Documentation	42
5.10.1.1 state	42
5.11 Factory Class Reference	42
5.11.1 Constructor & Destructor Documentation	43
5.11.1.1 ~Factory()	43
5.11.2 Member Function Documentation	43
5.11.2.1 getSupply()	43
5.11.2.2 make()	44
5.11.2.3 operation()	44
5.12 FailedState Class Reference	44
5.12.1 Detailed Description	45
5.12.2 Member Function Documentation	45
5.12.2.1 decreaseCurrency()	45
5.13 FoodFac Class Reference	45
5.13.1 Member Function Documentation	46
5.13.1.1 make()	46
5.14 FoodSupp Class Reference	46
5.14.1 Member Function Documentation	46
5.14.1.1 getAmount()	47
5.15 FoodWagon Class Reference	47
5.16 HealthyState Class Reference	47
5.16.1 Detailed Description	48
5.16.2 Member Function Documentation	48
5.16.2.1 decreaseCurrency()	48
5.16.2.2 getDemotionState()	48
5.17 Historian Class Reference	49
5.17.1 Constructor & Destructor Documentation	49
5.17.1.1 Historian()	49
5.17.2 Member Function Documentation	49
5.17.2.1 restoreAlly()	49
5.17.2.2 setAlly()	50
5.18 History Class Reference	50
5.18.1 Constructor & Destructor Documentation	50
5.18.1.1 History()	50
5.18.2 Member Function Documentation	51
5.18.2.1 getBannerman()	51
5.19 HistoryBook Class Reference	51
5.19.1 Member Function Documentation	51
5.19.1.1 add()	51

5.19.1.2 restoreAlly()	52
5.20 Iterator Class Reference	52
5.20.1 Detailed Description	53
5.20.2 Member Function Documentation	53
5.20.2.1 Current()	53
5.20.2.2 hasNext()	53
5.20.2.3 isActive()	54
5.20.2.4 next()	54
5.21 Kingdom Class Reference	54
5.21.1 Detailed Description	55
5.21.2 Constructor & Destructor Documentation	55
5.21.2.1 Kingdom()	55
5.21.2.2 ~Kingdom()	55
5.21.3 Member Function Documentation	55
5.21.3.1 add()	56
5.21.3.2 remove()	56
5.21.4 Member Data Documentation	56
5.21.4.1 bannerman	56
5.21.4.2 economy	56
5.22 Location Class Reference	57
5.22.1 Detailed Description	57
5.22.2 Member Function Documentation	57
5.22.2.1 sendScout()	57
5.23 MasterOfCoin Class Reference	58
5.23.1 Detailed Description	58
5.23.2 Constructor & Destructor Documentation	59
5.23.2.1 MasterOfCoin()	59
5.23.3 Member Function Documentation	59
5.23.3.1 notify()	59
5.24 MedicalFac Class Reference	59
5.24.1 Member Function Documentation	60
5.24.1.1 make()	60
5.25 MedicalSupp Class Reference	60
5.25.1 Member Function Documentation	61
5.25.1.1 getAmount()	61
5.26 MedicalWagon Class Reference	61
5.27 Raven Class Reference	61
5.27.1 Detailed Description	62
5.27.2 Member Function Documentation	62
5.27.2.1 update()	62
5.28 sendRaven Class Reference	62
5.28.1 Detailed Description	63

5.28.2 Constructor & Destructor Documentation	63
5.28.2.1 sendRaven()	63
5.28.3 Member Function Documentation	64
5.28.3.1 update()	64
5.29 Siege Class Reference	64
5.29.1 Detailed Description	65
5.29.2 Constructor & Destructor Documentation	65
5.29.2.1 Siege()	65
5.29.2.2 ~Siege()	66
5.29.3 Member Function Documentation	66
5.29.3.1 attack()	66
5.30 State Class Reference	66
5.30.1 Detailed Description	67
5.30.2 Constructor & Destructor Documentation	67
5.30.2.1 ~State()	67
5.30.3 Member Function Documentation	67
5.30.3.1 decreaseCurrency()	67
5.30.3.2 getDemotionState()	68
5.30.3.3 setContext()	68
5.30.4 Member Data Documentation	68
5.30.4.1 context	68
5.31 Strategy Class Reference	69
5.31.1 Detailed Description	69
5.31.2 Constructor & Destructor Documentation	70
5.31.2.1 Strategy()	70
5.31.2.2 ~Strategy()	70
5.31.3 Member Function Documentation	70
5.31.3.1 attack()	70
5.31.3.2 getEnemyBannerman()	71
5.31.3.3 getMyBannerman()	71
5.31.3.4 getStrategyName()	71
5.31.4 Member Data Documentation	72
5.31.4.1 enemyBannerman	72
5.31.4.2 enemyKingdom	72
5.31.4.3 minFavour	72
5.31.4.4 minSupplies	72
5.31.4.5 myBannerman	72
5.31.4.6 myKingdom	72
5.31.4.7 strategy	73
5.32 SupplyWagon Class Reference	73
5.32.1 Member Function Documentation	73
5.32.1.1 clone()	73

5.32.1.2 getSupplies()	74
5.32.1.3 setSup()	74
5.33 Topology Class Reference	74
5.33.1 Detailed Description	75
5.33.2 Member Function Documentation	75
5.33.2.1 getTopology()	75
5.33.2.2 sendScout()	75
5.34 Treasury Class Reference	76
5.34.1 Detailed Description	76
5.35 Troop Class Reference	76
5.35.1 Detailed Description	78
5.35.2 Constructor & Destructor Documentation	78
5.35.2.1 ~Troop()	78
5.35.3 Member Function Documentation	78
5.35.3.1 attach()	78
5.35.3.2 attack()	79
5.35.3.3 changeStrategy()	79
5.35.3.4 decreaseFavour()	79
5.35.3.5 decreaseFood()	80
5.35.3.6 decreaseMedical()	80
5.35.3.7 decreasePower()	80
5.35.3.8 decreaseWeapons()	80
5.35.3.9 detach()	80
5.35.3.10 getDamage()	81
5.35.3.11 getFavour()	81
5.35.3.12 getFood()	81
5.35.3.13 getHP()	82
5.35.3.14 getMedical()	82
5.35.3.15 getSize()	82
5.35.3.16 getWeapons()	82
5.35.3.17 increaseFavour()	83
5.35.3.18 increaseHP()	83
5.35.3.19 increasePower()	83
5.35.3.20 receiveDamage()	83
5.35.3.21 setFood()	84
5.35.3.22 setMaster()	84
5.35.3.23 setMedical()	84
5.35.3.24 setRaven()	85
5.35.3.25 setWeapons()	85
5.35.4 Member Data Documentation	85
5.35.4.1 damage	85
5.35.4.2 size	85

5.35.4.3 strategy	86
5.36 UnstableState Class Reference	86
5.36.1 Detailed Description	86
5.36.2 Member Function Documentation	87
5.36.2.1 decreaseCurrency()	87
5.36.2.2 getDemotionState()	87
5.37 WarIndicators Class Reference	87
5.37.1 Detailed Description	88
5.37.2 Constructor & Destructor Documentation	88
5.37.2.1 WarIndicators()	88
5.37.2.2 ~WarIndicators()	88
5.37.3 Member Data Documentation	89
5.37.3.1 m	89
5.38 WarTheatre Class Reference	89
5.38.1 Detailed Description	90
5.38.2 Constructor & Destructor Documentation	90
5.38.2.1 WarTheatre()	90
5.38.3 Member Function Documentation	90
5.38.3.1 decideVenue()	90
5.38.3.2 getStrategy()	91
5.38.3.3 sendScout()	91
5.38.4 Member Data Documentation	91
5.38.4.1 difficulty	91
5.38.4.2 location	91
5.38.4.3 strategy	91
5.38.4.4 venue	92
5.39 WeaponsFac Class Reference	92
5.39.1 Member Function Documentation	92
5.39.1.1 make()	92
5.40 WeaponSupp Class Reference	93
5.40.1 Member Function Documentation	93
5.40.1.1 getAmount()	93
5.41 WeaponWagon Class Reference	93
5.42 Weather Class Reference	94
5.42.1 Detailed Description	94
5.42.2 Constructor & Destructor Documentation	95
5.42.2.1 Weather()	95
5.42.3 Member Function Documentation	95
5.42.3.1 calcEffect()	95
5.42.3.2 setTemp()	95
5.42.3.3 setWindSpeed()	96
5.42.4 Member Data Documentation	96

5.42.4.1 rain	96
5.42.4.2 temp	96
5.42.4.3 windspeed	96
6 File Documentation	97
6.1 Ambush.h	97
6.2 ArmySupplies.h	97
6.3 Assassinate.h	97
6.4 Bannerman.h	98
6.5 BattleField.h	99
6.6 Class.h	99
6.7 Commander.h	99
6.8 Conditions.h	100
6.9 Conlterator.h	100
6.10 Economy.h	101
6.11 Factory.h	101
6.12 FailedState.h	101
6.13 FoodFac.h	102
6.14 FoodSupp.h	102
6.15 FoodWagon.h	102
6.16 HealthyState.h	102
6.17 Historian.h	103
6.18 History.h	103
6.19 HistoryBook.h	103
6.20 Iterator.h	104
6.21 Kingdom.h	104
6.22 Location.h	104
6.23 MasterOfCoin.h	104
6.24 MedicalFac.h	105
6.25 MedicalSupp.h	105
6.26 MedicalWagon.h	105
6.27 Raven.h	106
6.28 sendRaven.h	106
6.29 Siege.h	106
6.30 State.h	106
6.31 Strategy.h	107
6.32 SupplyWagon.h	107
6.33 Topology.h	107
6.34 Treasury.h	108
6.35 Troop.h	108
6.36 UnstableState.h	109
6.37 WarIndicators.h	109

6.38 WarTheatre.h	109
6.39 WeaponsFac.h	110
6.40 WeaponSupp.h	110
6.41 WeaponWagon.h	110
6.42 Weather.h	111
Index	113

Chapter 1

Todo List

Member `Ambush::Ambush` (int stealth, `Kingdom` *myKingdom, `Kingdom` *enemyKingdom, `Bannerman` *myBannerman, `Bannerman` *enemyBannerman, string name, int min, int minFavour)

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

Member `Ambush::attack` (`Bannerman` *myBannerman, `Bannerman` *enemyBannerman)

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

Member `Ambush::~~Ambush` ()

destructor.

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

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

Member `Assassinate::attack` (`Bannerman` *myBannerman, `Bannerman` *enemyBannerman)

makes assassin of attacking Bannerman try to kill enemyBannerman

Member `Assassinate::~~Assassinate` ()

destructor.

Member `BattleField::attack` (`Bannerman` *myBannerman, `Bannerman` *enemyBannerman)

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

Member `BattleField::BattleField` (`Kingdom` *myKingdom, `Kingdom` *enemyKingdom, `Bannerman` *myBannerman, `Bannerman` *enemyBannerman, string name, int min, int minFavour)

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

Member `BattleField::~~BattleField` ()

destructor.

Member `FailedState::decreaseCurrency` ()

decreases `Economy` currency and then checks if the conditions are sufficient for the economy to downgrade to a lower state

Member `HealthyState::decreaseCurrency` ()

decreases `Economy` currency and then checks if the conditions are sufficient for the economy to downgrade to a lower state

Member `HealthyState::getDemotionState` ()

gets lower level state of economy

Member `Kingdom::~~Kingdom ()`

destructor. deallocates all pointers of this class

Member `Siege::attack (Bannerman *myBannerman, Bannerman *enemyBannerman)`

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

Member `Siege::Siege (int stealth, Kingdom *myKingdom, Kingdom *enemyKingdom, Bannerman *myBannerman, Bannerman *enemyBannerman, string name, int min, int minFavour)`

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

Member `Siege::~~Siege ()`

destructor.

Member `State::decreaseCurrency ()=0`

Abstract method.

Member `State::getDemotionState ()`

gets lower level state of economy with null being returned if no lower state is possible

Member `State::setContext (Economy *context)`

sets context variable.

Member `State::~~State ()`

destructor. deallocates context

Member `Strategy::attack (Bannerman *myBannerman, Bannerman *enemyBannerman)=0`

Abstract method

Member `Strategy::getEnemyBannerman ()`

gets Bannerman being attacked

Member `Strategy::getMyBannerman ()`

gets attacking Bannerman

Member `Strategy::getStrategyName ()`

gets strategy variable

Member `Strategy::Strategy (Kingdom *myKingdom, Kingdom *enemyKingdom, Bannerman *myBannerman, Bannerman *enemyBannerman, string name, int min, int minFavour)`

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

Member `Strategy::~~Strategy ()`

destructor. deallocates all pointers of this class

Member `UnstableState::decreaseCurrency ()`

decreases Economy currency and then checks if the conditions are sufficient for the economy to downgrade to a lower state

Member `UnstableState::getDemotionState ()`

gets lower level state of economy

Member `WarIndicators::~~WarIndicators ()`

destructor. deallocates Treasury pointer of this class

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

ArmySupplies	11
FoodSupp	46
MedicalSupp	60
WeaponSupp	93
Class	27
Historian	49
History	50
HistoryBook	51
Iterator	52
ConIterator	39
Kingdom	54
Raven	61
sendRaven	62
State	66
FailedState	44
HealthyState	47
UnstableState	86
SupplyWagon	73
FoodWagon	47
MedicalWagon	61
WeaponWagon	93
Treasury	76
MasterOfCoin	58
WarIndicators	87
Bannerman	14
Commander	28
Commander	28
Troop	76
Troop	76
Economy	42
Factory	42
FoodFac	45
MedicalFac	59

WeaponsFac	92
Strategy	69
Ambush	9
Assassinate	12
BattleField	25
Siege	64
WarTheatre	89
Conditions	37
Topology	74
Weather	94
Location	57

Chapter 3

Class Index

3.1 Class List

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

Ambush	A concrete strategy class. A class that provides an alternative strategy for how Bannerman will fight enemyBannerman	9
ArmySupplies		11
Assassinate	A concrete strategy class. A class that provides an alternative strategy for how Bannerman will fight enemyBannerman	12
Bannerman		14
BattleField	A concrete strategy class. A class that provides an alternative strategy for how Bannerman will fight enemyBannerman	25
Class		27
Commander	A class that acts as a container/composite for bannerman objects as well as performing operations on and using various bannerman objects	28
Conditions	Decorator class. This is the class/object decorates the concreteComponent. Inherits from WarTheatre class	37
Conlterator		39
Economy		42
Factory		42
FailedState	A concrete state class. A class that is one of the alternative concrete states for Economy of Kingdom	44
FoodFac		45
FoodSupp		46
FoodWagon		47
HealthyState	A concrete state class. A class that is one of the alternative concrete states for Economy of Kingdom	47
Historian		49
History		50
HistoryBook		51
Iterator		52
Kingdom	A Kingdom class that has all the fighting Bannerman and Economy	54

Location	
Concrete object. This is the class/object that will be decorated. Inherits from WarTheatre class	57
MasterOfCoin	
The Concrete class for the Mediator A class that allows for other classes to talk to when key changes are made	58
MedicalFac	59
MedicalSupp	60
MedicalWagon	61
Raven	
The Abstract class for the Observer A class that the Concrete Observer class inherits from	61
sendRaven	
The Concrete class for the Observer A class that watches the Bannerman for any changes that are made	62
Siege	
A concrete strategy class. A class that provides an alternative strategy for how Bannerman will fight enemyBannerman	64
State	
An Abstract state class. A class that provides an interface to the alternative concrete states for Economy of Kingdom	66
Strategy	
An Abstract strategy class. A class that provides an interface to the alternative concrete strategies for how Bannerman will fight enemyBannerman	69
SupplyWagon	73
Topology	
Concrete decorator B. This is class that decorates the topology of the venue. Inherits from Conditions class	74
Treasury	
The Abstract class for the Mediator A class that the Concrete Mediator class inherits from	76
Troop	
A class that defines the primitive objects of the Bannerman composition	76
UnstableState	
A concrete state class. A class that is one of the alternative concrete states for Economy of Kingdom	86
WarIndicators	
An interface class for all classes that communicate with each other through Treasury mediator	87
WarTheatre	
Abstract object. This is the Component participant in the decorator pattern	89
WeaponsFac	92
WeaponSupp	93
WeaponWagon	93
Weather	
Concrete decorator A. This is the class implements climate effects to the war location. Inherits from Conditions class	94

Chapter 4

File Index

4.1 File List

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

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

UnstableState.h	??
WarIndicators.h	??
WarTheatre.h	??
WeaponsFac.h	??
WeaponSupp.h	??
WeaponWagon.h	??
Weather.h	??

Chapter 5

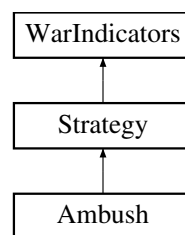
Class Documentation

5.1 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](#))
- bool [attack](#) (Bannerman *[myBannerman](#), Bannerman *[enemyBannerman](#))
- [~Ambush](#) ()

Private Attributes

- int [stealth](#)

Additional Inherited Members

5.1.1 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

5.1.2 Constructor & Destructor Documentation

5.1.2.1 Ambush()

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

Todo 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 Bannerman .
<i>min</i>	- minimum supplies for food, weapons and medicine
<i>minFavour</i>	- minimum favour below which bannerman change allegiances

5.1.2.2 ~Ambush()

```
Ambush::~~Ambush ( )
```

Todo destructor.

5.1.3 Member Function Documentation

5.1.3.1 attack()

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

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

Parameters

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

Returns

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

Implements [Strategy](#).

5.1.4 Member Data Documentation

5.1.4.1 stealth

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

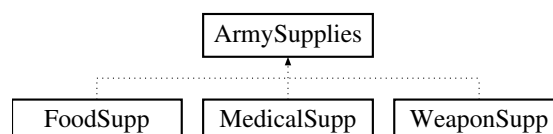
skill level of [Bannerman](#) using this strategy

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

- Ambush.h
- Ambush.cpp

5.2 ArmySupplies Class Reference

Inheritance diagram for ArmySupplies:



Public Member Functions

- virtual int **getAmount** ()=0

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

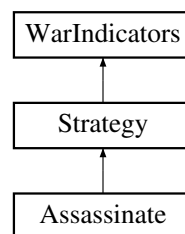
- ArmySupplies.h

5.3 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](#))
- bool [attack](#) (Bannerman *[myBannerman](#), Bannerman *[enemyBannerman](#))
- [~Assassinate](#) ()

Private Attributes

- int [stealth](#)
- bool [alive](#)

Additional Inherited Members

5.3.1 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

5.3.2 Constructor & Destructor Documentation

5.3.2.1 Assassinate()

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

Todo 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 Bannerman's assassin.
<i>min</i>	- minimum supplies for food, weapons and medicine
<i>minFavour</i>	- minimum favour below which Bannerman change allegiances

5.3.2.2 ~Assassinate()

```
Assassinate::~Assassinate ( )
```

Todo destructor.

5.3.3 Member Function Documentation

5.3.3.1 attack()

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

Todo makes assassin of attacking Bannerman try to kill enemyBannerman

Parameters

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

Returns

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

Implements [Strategy](#).

5.3.4 Member Data Documentation

5.3.4.1 alive

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

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

5.3.4.2 stealth

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

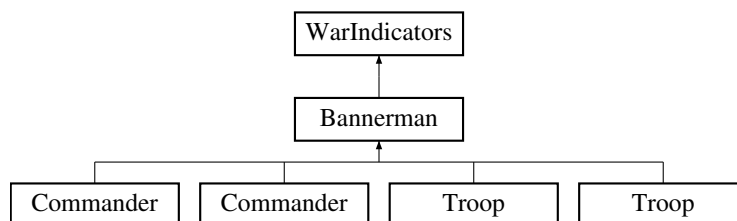
skill level of Assassin from [Bannerman](#) using this strategy

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

- Assassinate.h
- Assassinate.cpp

5.4 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.
- string **getName** ()
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.

Protected Attributes

- string [name](#)
- int [favour](#)
- int [numWeapons](#)
- int [damage](#)
- int [numFood](#)
- int [numMedical](#)
- list< [Raven](#) * > [ravenList](#)
- bool [assassin](#)
- [MasterOfCoin](#) * [m](#)
- [Strategy](#) * [strategy](#)
- int [HP](#)

5.4.1 Member Function Documentation

5.4.1.1 [attach\(\)](#)

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

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

Parameters

<i>o</i>	- the Raven Observer to attach.
----------	---

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

5.4.1.2 [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](#).

5.4.1.3 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](#).

5.4.1.4 decreaseFavour()

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

Abstract. decreases the favour of the bannerman.

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

5.4.1.5 decreaseFood()

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

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

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

5.4.1.6 decreaseMedical()

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

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

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

5.4.1.7 decreasePower()

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

Abstract. decreases the damage capability of the bannerman.

Parameters

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

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

5.4.1.8 decreaseWeapons()

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

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

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

5.4.1.9 detach()

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

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

Parameters

<i>o</i>	- the Raven Observer to detach.
----------	---

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

5.4.1.10 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](#).

5.4.1.11 getFavour()

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

Abstract. returns the favour of the bannerman.

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

5.4.1.12 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](#).

5.4.1.13 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](#).

5.4.1.14 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](#).

5.4.1.15 getName()

```
string Bannerman::getName ( )
```

Accessor that returns the name of the bannerman.

Returns

The name of the bannerman.

5.4.1.16 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](#).

5.4.1.17 increaseFavour()

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

Abstract. increases the favour of the bannerman.

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

5.4.1.18 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](#).

5.4.1.19 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](#).

5.4.1.20 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](#).

5.4.1.21 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](#).

5.4.1.22 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 MasterOfCoin mediator.
----------	--

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

5.4.1.23 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](#).

5.4.1.24 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 Raven Observer to attach.
----------	---

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

5.4.1.25 setWeapons()

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

Abstract. Sets the number of weapons of the bannerman.

Parameters

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

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

5.4.2 Member Data Documentation

5.4.2.1 assassin

```
bool Bannerman::assassin [protected]
```

Indicates whether or not the bannerman is an assassin

5.4.2.2 damage

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

Damage capability the bannerman has

5.4.2.3 favour

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

Amount of favour the bannerman has

5.4.2.4 HP

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

The health points of the bannerman

5.4.2.5 m

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

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

5.4.2.6 name

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

Name of the bannerman

5.4.2.7 numFood

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

Amount of food the bannerman has

5.4.2.8 numMedical

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

Amount of medical supplies the bannerman has

5.4.2.9 numWeapons

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

Number of weapons the bannerman has

5.4.2.10 ravenList

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

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

5.4.2.11 strategy

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

The attack strategy the bannerman uses

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

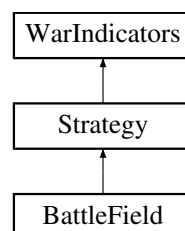
- Bannerman.h
- Bannerman.cpp

5.5 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)
- bool [attack](#) ([Bannerman](#) *myBannerman, [Bannerman](#) *enemyBannerman)
- ~[BattleField](#) ()

Additional Inherited Members

5.5.1 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

5.5.2 Constructor & Destructor Documentation

5.5.2.1 BattleField()

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

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

Parameters

<i>min</i>	- minimum supplies for food, weapons and medicine
<i>minFavour</i>	- minimum favour below which bannerman change allegiances

5.5.2.2 ~BattleField()

```
BattleField::~~BattleField ( )
```

Todo destructor.

5.5.3 Member Function Documentation

5.5.3.1 attack()

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

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

Parameters

<i>myBannerman</i>	- attacking bannerman object.
<i>enemyBannerman</i>	- Bannerman 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

5.6 Class Class Reference

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

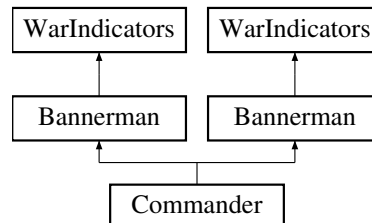
- Class.h

5.7 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` ()
Abstract. returns the favour of the bannerman.
- 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)
Decreases the damage capability of all the bannerman objects in groundForces.
- ~**Commander** ()
Default destructor.
- void `setRaven` (list< **Raven** * > r)
*Assigns a **Raven** observer list to all the bannerman objects in the groundForces list.*
- void `setMaster` (MasterOfCoin *m)
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.

Private Attributes

- list< **Bannerman** * > **groundForces**
list of bannerman components.
- string **name**

Additional Inherited Members

5.7.1 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

5.7.2 Constructor & Destructor Documentation

5.7.2.1 Commander()

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

[Commander](#) Constructor which takes in the name of a commander.

Parameters

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

5.7.3 Member Function Documentation

5.7.3.1 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 Raven observer object
----------	---

Implements [Bannerman](#).

5.7.3.2 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](#).

5.7.3.3 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](#).

5.7.3.4 decreaseFavour()

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

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

Implements [Bannerman](#).

5.7.3.5 decreaseFood()

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

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

Implements [Bannerman](#).

5.7.3.6 decreaseMedical()

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

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

Implements [Bannerman](#).

5.7.3.7 decreasePower()

```
void Commander::decreasePower (
    int x ) [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](#).

5.7.3.8 decreaseWeapons()

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

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

Implements [Bannerman](#).

5.7.3.9 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 Raven observer object
----------	---

Implements [Bannerman](#).

5.7.3.10 getDamage()

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

Accessor. Returns the damage.

Returns

The total groundForces damage.

Implements [Bannerman](#).

5.7.3.11 getFavour()

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

Abstract. returns the favour of the bannerman.

Implements [Bannerman](#).

5.7.3.12 getFood()

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

Accessor. Returns the total groundForces amount of food.

Returns

numFood.

Implements [Bannerman](#).

5.7.3.13 getHP()

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

Accessor. Returns the HP.

Returns

The total groundForces HP.

Implements [Bannerman](#).

5.7.3.14 getMedical()

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

Accessor. Returns the total groundForces amount of medical supplies.

Returns

numMedical.

Implements [Bannerman](#).

5.7.3.15 getTroops()

```
list< Bannerman * > Commander::getTroops ( )
```

Returns the groundForces bannerman list

5.7.3.16 getWeapons()

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

Accessor. Returns numWeapons.

Returns

The total groundForces number of Weapons.

Implements [Bannerman](#).

5.7.3.17 increaseFavour()

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

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

Implements [Bannerman](#).

5.7.3.18 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](#).

5.7.3.19 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](#).

5.7.3.20 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](#).

5.7.3.21 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](#).

5.7.3.22 setMaster()

```
void Commander::setMaster (
    MasterOfCoin * m ) [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 MasterOfCoin mediator.
----------	--

Implements [Bannerman](#).

5.7.3.23 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](#).

5.7.3.24 setRaven()

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

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

Parameters

<i>r</i>	- the Raven observer list to attach
----------	---

Implements [Bannerman](#).

5.7.3.25 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.
-----------------	---

5.7.3.26 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:

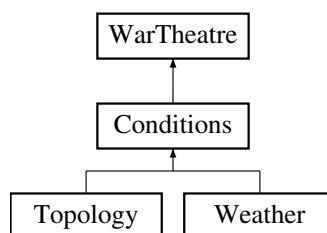
- Commander.h
- Commander.cpp

5.8 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](#) ()
Abstract. Decorates the location of the battle.
- [Conditions](#) ([WarTheatre](#) *myVenue)
constructor for the decorator class. merely used to assign variables and call the decorator function

Public Attributes

- [WarTheatre](#) * myVenue

5.8.1 Detailed Description

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

Author

Keabetswe Mothapo

Date

October 2022

5.8.2 Constructor & Destructor Documentation

5.8.2.1 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
----------------	--

5.8.3 Member Function Documentation

5.8.3.1 sendScout()

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

Abstract. Decorates the location of the battle.

Implements [WarTheatre](#).

Reimplemented in [Topology](#).

5.8.4 Member Data Documentation

5.8.4.1 myVenue

`WarTheatre* Conditions::myVenue`

Pointer to the decorated component

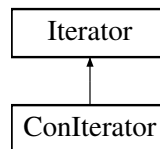
The documentation for this class was generated from the following files:

- Conditions.h
- Conditions.cpp

5.9 ConIterator Class Reference

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

Inheritance diagram for ConIterator:



Public Member Functions

- `ConIterator` (list< `Bannerman` * > X)
- `Bannerman` * `Current` ()
Abstract. 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` ()

Private Attributes

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

5.9.1 Detailed Description

@Brief 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

5.9.2 Constructor & Destructor Documentation

5.9.2.1 Conlterator()

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

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

Parameters

X	- the groundForces list which will be accessed sequentially.
---	--

5.9.3 Member Function Documentation

5.9.3.1 Current()

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

Implements [Iterator](#).

5.9.3.2 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](#).

5.9.3.3 isActive()

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

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

Returns

true
false

Test

Implements [Iterator](#).

5.9.3.4 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](#).

5.9.4 Member Data Documentation

5.9.4.1 armyList

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

The list of bannerman objects to traverse

5.9.4.2 it

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

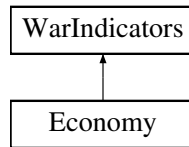
An iterator which is a pointer to the current bannerman object in [Commander](#)'s groundForces being accessed, primitive operations can be performed using this pointer.

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

- ConIterator.h
- ConIterator.cpp

5.10 Economy Class Reference

Inheritance diagram for Economy:



Public Member Functions

- **Economy** ([State](#) *state, int currency)
- void **SetState** ()
- void **decreaseCurrency** ()
- int **getCurrency** ()
- void **removeCurrency** (int i)

Private Attributes

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

Additional Inherited Members

5.10.1 Member Data Documentation

5.10.1.1 state

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

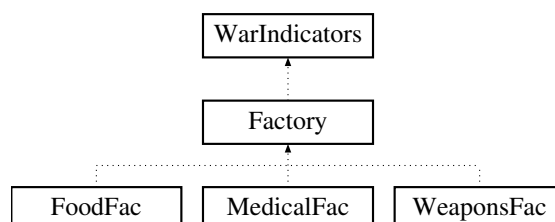
state of economy

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

- Economy.h
- Economy.cpp

5.11 Factory Class Reference

Inheritance diagram for Factory:



Public Member Functions

- void `operation` ()
Calls the method needed to manufacture the supplies.
- `ArmySupplies` * `getSupply` ()
Returns supplies.
- void `~Factory` ()
Deletes allocated memory to supply variable.

Protected Member Functions

- virtual `ArmySupplies` * `make` ()=0

Private Attributes

- `ArmySupplies` * `supply`

Additional Inherited Members

5.11.1 Constructor & Destructor Documentation

5.11.1.1 `~Factory()`

```
Factory::~~Factory ( )
```

Deletes allocated memory to supply variable.

Author

Ronin Brookes 19069686

5.11.2 Member Function Documentation

5.11.2.1 `getSupply()`

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

Returns supplies.

Returns

returns the supplies the `Factory` manufactured

Author

Ronin Brookes 19069686

5.11.2.2 make()

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

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

5.11.2.3 operation()

```
void Factory::operation ( )
```

Calls the method needed to manufacture the supplies.

Author

Ronin Brookes 19069686

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

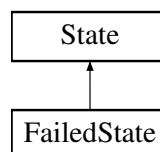
- [Factory.h](#)
- [Factory.cpp](#)

5.12 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](#) ()

Additional Inherited Members

5.12.1 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

5.12.2 Member Function Documentation

5.12.2.1 decreaseCurrency()

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

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

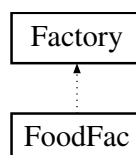
Implements [State](#).

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

- FailedState.h
- FailedState.cpp

5.13 FoodFac Class Reference

Inheritance diagram for FoodFac:



Public Member Functions

- [ArmySupplies](#) * [make](#) ()
Food [Factory](#) makes Food supplies.

Additional Inherited Members

5.13.1 Member Function Documentation

5.13.1.1 make()

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

Food [Factory](#) makes Food supplies.

Returns

returns the supplies the Food [Factory](#) just manufactured

Author

Ronin Brookes 19069686

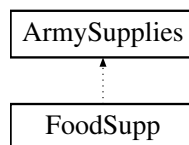
Implements [Factory](#).

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

- FoodFac.h
- FoodFac.cpp

5.14 FoodSupp Class Reference

Inheritance diagram for FoodSupp:



Public Member Functions

- int [getAmount](#) ()

Additional Inherited Members

5.14.1 Member Function Documentation

5.14.1.1 `getAmount()`

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

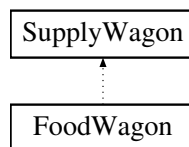
Implements [ArmySupplies](#).

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

- FoodSupp.h

5.15 FoodWagon Class Reference

Inheritance diagram for FoodWagon:



Additional Inherited Members

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

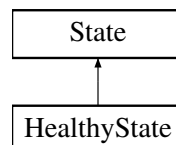
- FoodWagon.h

5.16 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](#) ()
- virtual [State](#) * [getDemotionState](#) ()

Additional Inherited Members

5.16.1 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

5.16.2 Member Function Documentation

5.16.2.1 decreaseCurrency()

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

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

Implements [State](#).

5.16.2.2 getDemotionState()

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

Todo gets lower level state of economy

Returns

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

Reimplemented from [State](#).

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

- HealthyState.h
- HealthyState.cpp

5.17 Historian Class Reference

Public Member Functions

- [Historian](#) ([Bannerman](#) *b)
constructor
- [History](#) * [setAlly](#) ()
set the memento object of the defecting bannerman
- [Bannerman](#) * [restoreAlly](#) ([History](#) *h)
return the bannerman that defected

Private Attributes

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

5.17.1 Constructor & Destructor Documentation

5.17.1.1 Historian()

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

constructor

Parameters

<i>b</i>	bannerman that is defecting
----------	-----------------------------

5.17.2 Member Function Documentation

5.17.2.1 restoreAlly()

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

return the bannerman that defected

Parameters

<i>h</i>	the memento object that needs to be restored
----------	--

Returns

the bannerman of that object

5.17.2.2 setAlly()

```
History * Historian::setAlly ( )
```

set the memento object of the defecting bannerman

Returns

the created memento object for that bannerman

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

- Historian.h
- Historian.cpp

5.18 History Class Reference

Public Member Functions

- [History](#) ([Bannerman](#) *b)
constructor
- [Bannerman](#) * [getBannerman](#) ()
get current saved bannerman

Private Attributes

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

5.18.1 Constructor & Destructor Documentation

5.18.1.1 History()

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

constructor

Parameters

<i>b</i>	initialisor for bannerman object whos state needs to be stored
----------	--

5.18.2 Member Function Documentation

5.18.2.1 getBannerman()

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

get current saved bannerman

Returns

the saved bannerman that is defecting

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

- History.h
- History.cpp

5.19 HistoryBook Class Reference

Public Member Functions

- void [add](#) (History *h)
add memento object to the list of defecetded allies
- History * [restoreAlly](#) (History *h)
restore the state of an ally returning

Private Attributes

- list< [History](#) * > **defectedAllies**

5.19.1 Member Function Documentation

5.19.1.1 add()

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

add memento object to the list of defecetded allies

Parameters

<i>h</i>	the memento object added
----------	--------------------------

5.19.1.2 restoreAlly()

```
History * HistoryBook::restoreAlly (
    History * h )
```

restore the state of an ally returning

Parameters

<i>h</i>	the ally that wants to return
----------	-------------------------------

Returns

the state of the ally to be returning

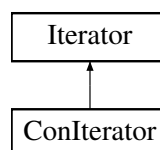
The documentation for this class was generated from the following files:

- HistoryBook.h
- HistoryBook.cpp

5.20 Iterator Class Reference

```
#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

5.20.1 Detailed Description

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

Author

Thapelo Thoka

Date

October 2022

5.20.2 Member Function Documentation

5.20.2.1 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 [ConIterator](#).

5.20.2.2 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 [ConIterator](#).

5.20.2.3 isActive()

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

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

Returns

true
false

Implemented in [ConIterator](#).

5.20.2.4 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](#)

5.21 Kingdom Class Reference

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

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

Public Member Functions

- [Kingdom](#) ([Economy](#) *economy, vector< [Bannerman](#) * > bannerman)
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](#) ()

Private Attributes

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

5.21.1 Detailed Description

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

Author

Morgan Bentley

Date

October 2022

5.21.2 Constructor & Destructor Documentation

5.21.2.1 Kingdom()

```
Kingdom::Kingdom (
    Economy * economy,
    vector< Bannerman * > bannerman )
```

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

Parameters

<i>economy</i>	- Economy pointer to player's Economy object.
<i>bannerman</i>	- vector list of Bannerman objects.

5.21.2.2 ~Kingdom()

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

[Todo](#) destructor. deallocates all pointers of this class

5.21.3 Member Function Documentation

5.21.3.1 add()

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

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

Parameters

<i>b</i>	- Bannerman that has defected from enemyKingdom
----------	---

5.21.3.2 remove()

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

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

Parameters

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

5.21.4 Member Data Documentation

5.21.4.1 bannerman

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

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

5.21.4.2 economy

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

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

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

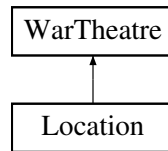
- Kingdom.h
- Kingdom.cpp

5.22 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
- void **createVenue ()**
creates default war theatre and calls the decorating function
- void **~Location ()**
free memory used in the pattern

5.22.1 Detailed Description

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

Author

Keabetswe Mothapo

Date

October 2022

5.22.2 Member Function Documentation

5.22.2.1 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:

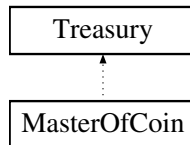
- Location.h
- Location.cpp

5.23 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 destructer 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.

Additional Inherited Members

5.23.1 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

5.23.2 Constructor & Destructor Documentation

5.23.2.1 MasterOfCoin()

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

Constructor to set the variables in this class.

Parameters

<i>economy</i>	holds the Economy 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

5.23.3 Member Function Documentation

5.23.3.1 notify()

```
void MasterOfCoin::notify (
    WarIndicators sender )
```

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

Parameters

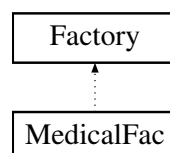
<i>sender</i>	is the variable that holds the object who sent the notify request
---------------	---

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

- MasterOfCoin.h
- MasterOfCoin.cpp

5.24 MedicalFac Class Reference

Inheritance diagram for MedicalFac:



Public Member Functions

- `ArmySupplies * make ()`
Medical [Factory](#) makes Medical supplies.

Additional Inherited Members

5.24.1 Member Function Documentation

5.24.1.1 `make()`

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

Medical [Factory](#) makes Medical supplies.

Returns

returns the supplies the Medical [Factory](#) just manufactured

Author

Ronin Brookes 19069686

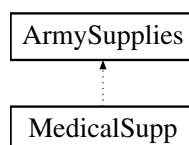
Implements [Factory](#).

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

- MedicalFac.h
- MedicalFac.cpp

5.25 MedicalSupp Class Reference

Inheritance diagram for MedicalSupp:



Public Member Functions

- `int getAmount ()`

Additional Inherited Members

5.25.1 Member Function Documentation

5.25.1.1 getAmount()

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

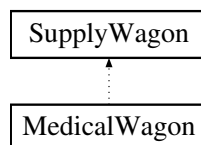
Implements [ArmySupplies](#).

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

- MedicalSupp.h

5.26 MedicalWagon Class Reference

Inheritance diagram for MedicalWagon:



Additional Inherited Members

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

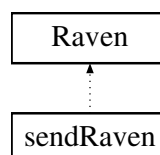
- MedicalWagon.h

5.27 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.

5.27.1 Detailed Description

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

Author

Sameet Keshav u21479373

Date

October 2022

5.27.2 Member Function Documentation

5.27.2.1 [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:

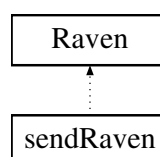
- Raven.h

5.28 [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.

Additional Inherited Members

5.28.1 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

5.28.2 Constructor & Destructor Documentation

5.28.2.1 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 SupplyWagon
<i>subject</i>	is a pointer for the Bannerman subject that this Observer will observe

5.28.3 Member Function Documentation

5.28.3.1 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:

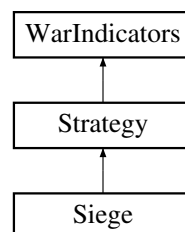
- [sendRaven.h](#)
- [sendRaven.cpp](#)

5.29 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)
- bool [attack](#) ([Bannerman](#) *myBannerman, [Bannerman](#) *enemyBannerman)
- [~Siege](#) ()

Private Attributes

- int **stealth**

Additional Inherited Members

5.29.1 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

5.29.2 Constructor & Destructor Documentation

5.29.2.1 Siege()

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

Todo 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 Bannerman .
<i>min</i>	- minimum supplies for food, weapons and medicine
<i>minFavour</i>	- minimum favour below which bannerman change allegiances

5.29.2.2 ~Siege()

```
Siege::~~Siege ( )
```

Todo destructor.

5.29.3 Member Function Documentation

5.29.3.1 attack()

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

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

Parameters

<i>myBannerman</i>	- attacking bannerman object.
<i>enemyBannerman</i>	- Bannerman 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:

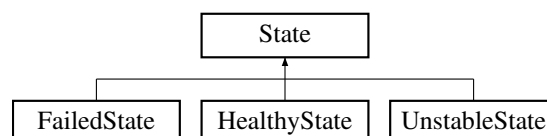
- [Siege.h](#)
- [Siege.cpp](#)

5.30 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)
- virtual void **decreaseCurrency** ()=0
- virtual [State](#) * **getDemotionState** ()
- virtual [~State](#) ()

Protected Attributes

- [Economy](#) * context

5.30.1 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

5.30.2 Constructor & Destructor Documentation

5.30.2.1 [~State](#)()

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

Todo destructor. deallocates context

5.30.3 Member Function Documentation

5.30.3.1 **decreaseCurrency**()

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

Todo Abstract method.

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

5.30.3.2 getDemotionState()

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

Todo 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](#).

5.30.3.3 setContext()

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

Todo sets context variable.

Parameters

<i>context</i>	- Economy pointer to player's Economy object.
----------------	---

5.30.4 Member Data Documentation

5.30.4.1 context

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

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

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

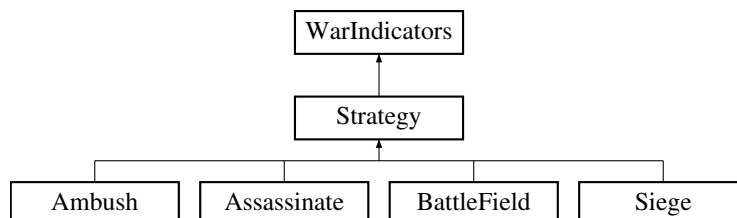
- State.h
- State.cpp

5.31 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)
- virtual bool [attack](#) ([Bannerman](#) *myBannerman, [Bannerman](#) *enemyBannerman)=0
- virtual string [getStrategyName](#) ()
- virtual [Bannerman](#) * [getMyBannerman](#) ()
- virtual [Bannerman](#) * [getEnemyBannerman](#) ()
- virtual [~Strategy](#) ()

Protected Attributes

- [Kingdom](#) * myKingdom
- [Kingdom](#) * enemyKingdom
- [Bannerman](#) * myBannerman
- [Bannerman](#) * enemyBannerman
- string [strategy](#)
- int [minFavour](#)
- int [minSupplies](#)

5.31.1 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

5.31.2 Constructor & Destructor Documentation

5.31.2.1 Strategy()

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

Todo constructor. initializes myKingdom, enemyKingdom, myBannerman, enemyBannerman, strategy, min↔ Supplies and minFavour.

Parameters

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

5.31.2.2 ~Strategy()

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

Todo destructor. deallocates all pointers of this class

5.31.3 Member Function Documentation

5.31.3.1 attack()

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

Todo Abstract method

Parameters

<i>myBannerman</i>	- attacking Bannerman object.
<i>enemyBannerman</i>	- Bannerman 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](#).

5.31.3.2 getEnemyBannerman()

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

Todo gets [Bannerman](#) being attacked

Returns

enemyBannerman pointer

5.31.3.3 getMyBannerman()

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

Todo gets attacking [Bannerman](#)

Returns

myBannerman pointer

5.31.3.4 getStrategyName()

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

Todo gets strategy variable

Returns

strategy variable

5.31.4 Member Data Documentation

5.31.4.1 enemyBannerman

`Bannerman* Strategy::enemyBannerman [protected]`

defending [Bannerman](#)

5.31.4.2 enemyKingdom

`Kingdom* Strategy::enemyKingdom [protected]`

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

5.31.4.3 minFavour

`int Strategy::minFavour [protected]`

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

5.31.4.4 minSupplies

`int Strategy::minSupplies [protected]`

minimum supplies for food,weapons and medicine

5.31.4.5 myBannerman

`Bannerman* Strategy::myBannerman [protected]`

attacking [Bannerman](#)

5.31.4.6 myKingdom

`Kingdom* Strategy::myKingdom [protected]`

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

5.31.4.7 strategy

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

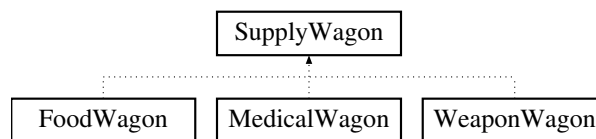
name of concrete strategy

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

- Strategy.h
- Strategy.cpp

5.32 SupplyWagon Class Reference

Inheritance diagram for SupplyWagon:



Public Member Functions

- 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`

5.32.1 Member Function Documentation

5.32.1.1 clone()

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

Clone the Supply Wagon object.

Returns

the new Supply Wagon clone.

Author

Ronin Brookes 19069686

5.32.1.2 getSupplies()

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

Return the supp variable.

Returns

returns the supp variable. Returns the necessary supplies.

Author

Ronin Brookes 19069686

5.32.1.3 setSup()

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

initialize the supp variable.

Parameters

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

Author

Ronin Brookes 19069686

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

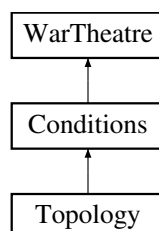
- SupplyWagon.h

5.33 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

- [Topology](#) * [getTopology](#) ()
getter for the [Topology](#) object
- void [setTopology](#) ()
setter for the topology of the war venue uses the inherited venue variable to adjust difficulty of the terrain
- [Topology](#) ()
default constructor that calls the decorator function
- void [sendScout](#) ()
implements the decorator function

Additional Inherited Members

5.33.1 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

5.33.2 Member Function Documentation

5.33.2.1 [getTopology\(\)](#)

```
Topology * Topology::getTopology ( )
```

getter for the [Topology](#) object

Returns

- this topology object

5.33.2.2 [sendScout\(\)](#)

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

implements the decorator function

Reimplemented from [Conditions](#).

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

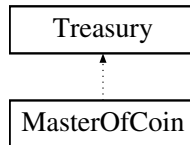
- [Topology.h](#)
- [Topology.cpp](#)

5.34 Treasury Class Reference

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

```
#include <Treasury.h>
```

Inheritance diagram for Treasury:



Public Member Functions

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

5.34.1 Detailed Description

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

Author

Sameet Keshav u21479373

Date

October 2022

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

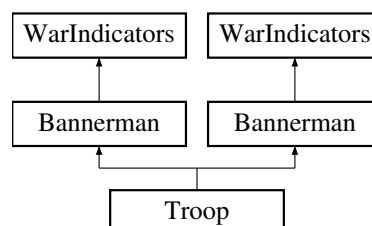
- Treasury.h

5.35 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](#) ()
Abstract. returns the favour of the bannerman.
- 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)
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)
- *Assigns a list of `Raven` Observers to the troop's `ravenList`.*
void `setMaster` (`MasterOfCoin` *m)
- *Assigns a `MasterofCoin` mediator to ensure that the troop has the supplies it needs.*
~`Troop` ()

Private Attributes

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

Additional Inherited Members

5.35.1 Detailed Description

A class that defines the primitive objects of the `Bannerman` composition.

Author

Thapelo Thoka

Date

October 2022

5.35.2 Constructor & Destructor Documentation

5.35.2.1 ~Troop()

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

Default destructor.

5.35.3 Member Function Documentation

5.35.3.1 attach()

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

Attaches a `Raven` observer to the troop.

Parameters

<i>o</i>	- the Raven observer object to attach
----------	---

Implements [Bannerman](#).

5.35.3.2 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](#).

5.35.3.3 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](#).

5.35.3.4 decreaseFavour()

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

Decreases the loyalty favour level of the troop by 1.

Implements [Bannerman](#).

5.35.3.5 decreaseFood()

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

decreases the numFood variable of the troop by 1.

Implements [Bannerman](#).

5.35.3.6 decreaseMedical()

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

decreases the numMedical variable of the troop by 1.

Implements [Bannerman](#).

5.35.3.7 decreasePower()

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

Decreases the damage capability of the troop.

Parameters

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

Implements [Bannerman](#).

5.35.3.8 decreaseWeapons()

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

decreases the numWeapons variable of the troop by 1.

Implements [Bannerman](#).

5.35.3.9 detach()

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

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

Parameters

<i>o</i>	- the Raven observer object to detach
----------	---

Implements [Bannerman](#).

5.35.3.10 getDamage()

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

Accessor. Returns the damage capacity of the troop.

Returns

damage.

Implements [Bannerman](#).

5.35.3.11 getFavour()

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

Abstract. returns the favour of the bannerman.

Implements [Bannerman](#).

5.35.3.12 getFood()

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

Accessor. Returns the amount of food the troop has.

Returns

numFood.

Implements [Bannerman](#).

5.35.3.13 getHP()

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

Returns the troop's HP.

Returns

HP.

Implements [Bannerman](#).

5.35.3.14 getMedical()

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

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

Returns

numMedical.

Implements [Bannerman](#).

5.35.3.15 getSize()

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

Returns the troop's size.

Returns

size.

5.35.3.16 getWeapons()

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

Accessor. Returns the number of weapons the troop has.

Returns

numWeapons.

Implements [Bannerman](#).

5.35.3.17 increaseFavour()

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

Increases the loyalty favour level of the troop by 1.

Implements [Bannerman](#).

5.35.3.18 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](#).

5.35.3.19 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](#).

5.35.3.20 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](#).

5.35.3.21 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](#).

5.35.3.22 setMaster()

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

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

Parameters

<i>m</i>	- the new MasterOfCoin mediator to assign to the troop.
----------	---

Implements [Bannerman](#).

5.35.3.23 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](#).

5.35.3.24 setRaven()

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

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

Parameters

<i>r</i>	- the list of Raven Observers to attach to the troop.
----------	---

Implements [Bannerman](#).

5.35.3.25 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](#).

5.35.4 Member Data Documentation

5.35.4.1 damage

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

The damage capability which the troop has

5.35.4.2 size

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

The number of soldiers in the troop

5.35.4.3 strategy

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

The attack strategy used by the troop in battle

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

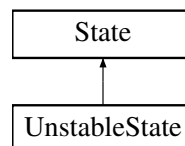
- Troop.h
- Troop.cpp

5.36 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](#) ()
- [State](#) * [getDemotionState](#) ()

Additional Inherited Members

5.36.1 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

5.36.2 Member Function Documentation

5.36.2.1 decreaseCurrency()

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

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

Implements [State](#).

5.36.2.2 getDemotionState()

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

Todo gets lower level state of economy

Returns

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

Reimplemented from [State](#).

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

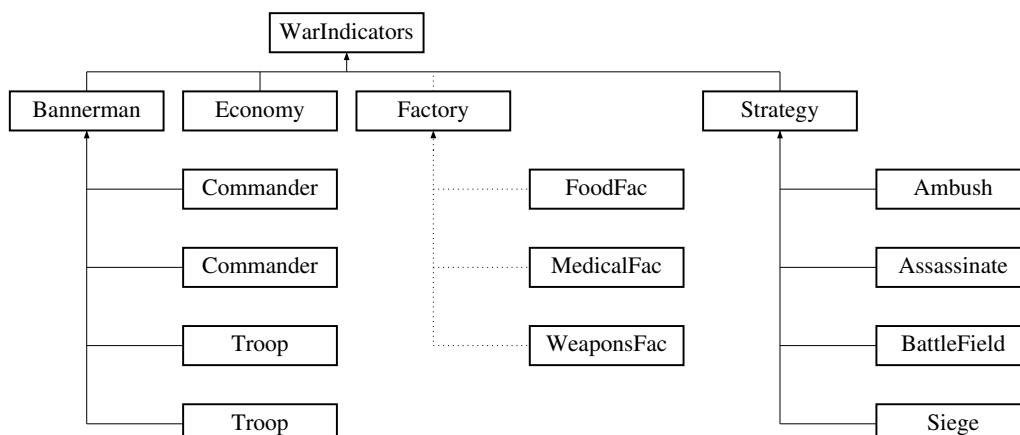
- UnstableState.h
- UnstableState.cpp

5.37 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.
- virtual **~WarIndicators** ()

Protected Attributes

- [Treasury](#) * m

5.37.1 Detailed Description

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

Author

Morgan Bentley

Date

October 2022

5.37.2 Constructor & Destructor Documentation

5.37.2.1 WarIndicators()

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

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

Parameters

<i>m</i>	- mediator which is of Treasury pointer type
----------	--

5.37.2.2 ~WarIndicators()

```
virtual WarIndicators::~~WarIndicators ( ) [inline], [virtual]
```

Todo destructor. deallocates [Treasury](#) pointer of this class

5.37.3 Member Data Documentation

5.37.3.1 m

`Treasury* WarIndicators::m [protected]`

`Treasury` mediator pointer

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

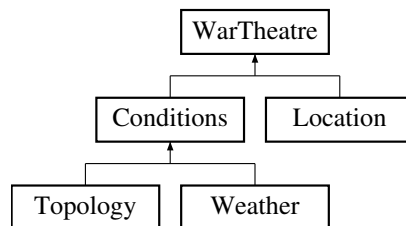
- WarIndicators.h

5.38 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 (Strategy *myStrat)`
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
- **`WarTheatre (Strategy myStrat)`**

Private Attributes

- `Strategy * strategy`
- `char venue`
- `string location`
- `int difficulty`

5.38.1 Detailed Description

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

Author

Keabetswe Mothapo

Date

October 2022

5.38.2 Constructor & Destructor Documentation

5.38.2.1 WarTheatre()

```
WarTheatre::WarTheatre (
    Strategy * myStrat )
```

Constructor.

Parameters

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

5.38.3 Member Function Documentation

5.38.3.1 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

5.38.3.2 `getStrategy()`

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

returns the strategy the location is based on

Returns

the strategy pointer

5.38.3.3 `sendScout()`

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

Abstract. Decorates the location of the battle.

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

5.38.4 Member Data Documentation

5.38.4.1 `difficulty`

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

Integer representing the difficulty of the battle at the decorated venue

5.38.4.2 `location`

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

The location of the venue as a string

5.38.4.3 `strategy`

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

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

5.38.4.4 venue

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

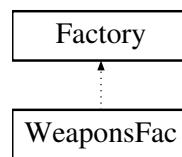
Char representing the venue. options a to d only

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

- WarTheatre.h
- WarTheatre.cpp

5.39 WeaponsFac Class Reference

Inheritance diagram for WeaponsFac:



Public Member Functions

- [ArmySupplies](#) * [make](#) ()
Weapon [Factory](#) makes Weapon supplies.

Additional Inherited Members

5.39.1 Member Function Documentation

5.39.1.1 make()

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

Weapon [Factory](#) makes Weapon supplies.

Returns

returns the supplies the Weapon [Factory](#) just manufactured

Author

Ronin Brookes 19069686

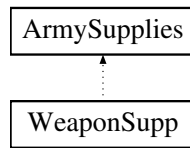
Implements [Factory](#).

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

- WeaponsFac.h
- WeaponsFac.cpp

5.40 WeaponSupp Class Reference

Inheritance diagram for WeaponSupp:



Public Member Functions

- int [getAmount](#) ()

Additional Inherited Members

5.40.1 Member Function Documentation

5.40.1.1 [getAmount\(\)](#)

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

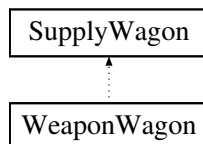
Implements [ArmySupplies](#).

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

- `WeaponSupp.h`

5.41 WeaponWagon Class Reference

Inheritance diagram for WeaponWagon:



Additional Inherited Members

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

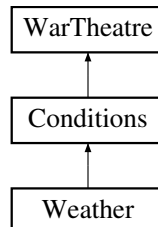
- `WeaponWagon.h`

5.42 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](#) (int val)
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

Private Attributes

- double [temp](#)
- bool [rain](#)
- int [windspeed](#)

Additional Inherited Members

5.42.1 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

5.42.2 Constructor & Destructor Documentation

5.42.2.1 Weather()

```
Weather::Weather (
    int val )
```

Costructor of the weather object.

Parameters

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

5.42.3 Member Function Documentation

5.42.3.1 calcEffect()

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

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

Returns

returns the adjusted difficulty value

5.42.3.2 setTemp()

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

setter for the temperature variable

Parameters

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

5.42.3.3 setWindSpeed()

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

setter for the windspeed variable

Parameters

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

5.42.4 Member Data Documentation

5.42.4.1 rain

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

Boolean to indicate whether it is raining or not

5.42.4.2 temp

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

Value of the temperature at the venue

5.42.4.3 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 6

File Documentation

6.1 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:
21     Ambush(int stealth, Kingdom* myKingdom, Kingdom* enemyKingdom, Bannerman* myBannerman, Bannerman*
        enemyBannerman, string name, int min, int minFavour);
27     bool attack(Bannerman* myBannerman, Bannerman* enemyBannerman);
29     ~Ambush();
30 };
31
32 #endif
```

6.2 ArmySupplies.h

```
1 #ifndef ARMYSUPPLIES_H
2 #define ARMYSUPPLIES_H
3
4 class ArmySupplies {
5
6
7 public:
8     virtual int getAmount() = 0;
9 };
10
11 #endif
```

6.3 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:
24     Assassinate(int stealth, bool alive, Kingdom* myKingdom, Kingdom* enemyKingdom, Bannerman* myBannerman,
        Bannerman* enemyBannerman, string name, int min, int minFavour);
30     bool attack(Bannerman* myBannerman, Bannerman* enemyBannerman);
32     ~Assassinate();
33 };
34
35 #endif
```

6.4 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
12
17 class Strategy;
18 class Bannerman: public WarIndicators {
19
20 protected:
21
22     string name;
23
24
25     int favour;
26
27
28     int numWeapons;
29
30
31     int damage;
32
33
34     int numFood;
35
36
37     int numMedical;
38
39
40     list<Raven*> ravenList;
41
42
43
44     bool assassin;
45
46
47     MasterOfCoin* m;
48
49
50     Strategy* strategy;
51
52
53     int HP;
54
55
56
57 public:
58     Bannerman();
59
60
61     virtual void increaseFavour() = 0;
62
63     virtual void decreaseFavour() = 0;
64
65     virtual void attach(Raven* o)=0;
66
67     virtual void detach(Raven* o)=0;
68
69     virtual void increaseHP(int boost) = 0; //implement
70
71     virtual void changeStrategy(Strategy* strategy) = 0;
72
73     virtual void attack(Bannerman* myBannerman, Bannerman* enemyBannerman) = 0;
74
75     virtual void increasePower(int boost) = 0;
76
77     string getName();
78
79     virtual int getHP() = 0;
80
81     virtual int getDamage() = 0;
82
83     virtual void receiveDamage(int boost) = 0;
84
85     virtual void decreasePower(int x) = 0;
86
87     virtual void decreaseWeapons() = 0;
88
89     virtual void decreaseFood() = 0;
90
91     virtual void decreaseMedical() = 0;
92
93     virtual int getWeapons() = 0;
94
95     virtual int getFood() = 0;
96
97     virtual int getMedical() = 0;
98
99     virtual void setWeapons(int numWeapons) = 0;
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 };
186
187 #endif

```

6.5 BattleField.h

```

1 #ifndef BATTLEFIELD_H
2 #define BATTLEFIELD_H
3 #include "Strategy.h"
4
10 class BattleField :public Strategy {
11
12 public:
18     BattleField(Kingdom* myKingdom, Kingdom* enemyKingdom, Bannerman* myBannerman, Bannerman*
        enemyBannerman, string name, int min, int minFavour);
24     bool attack(Bannerman* myBannerman, Bannerman* enemyBannerman);
26     ~BattleField();
27 };
28
29 #endif

```

6.6 Class.h

```

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

```

6.7 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 private:
19     list<Bannerman*> groundForces;
20     string name;
21 public:
22
26     Commander(string name);
27
30     Iterator* createIterator();
31
34     void removeBannerman(Bannerman* x);
35
40     void attack(Bannerman* myBannerman, Bannerman* enemyBannerman) override;
41
44     void addBannerman(Bannerman* b);
45
47     list<Bannerman*> getTroops();
48
54     int getHP() override;
55
61     int getDamage() override;
65     void attach(Raven* o) override;
66

```

```

70     void detach(Raven* o) override;
71
76     void receiveDamage(int x) override;
79     void decreaseWeapons() override;
82     void decreaseFood() override;
85     void decreaseMedical() override;
86
90     void increasePower(int boost) override;
91
97     int getWeapons() override;
98
104     int getFood() override;
105
111     int getMedical() override;
112
113
117     void increaseHP(int boost) override;
118
122     void changeStrategy(Strategy* strategy) override;
123
126     void increaseFavour() override;
127
130     void decreaseFavour() override;
131
132     int getFavour();
133
137     void setWeapons(int numWeapons) override;
138
142     void setFood(int numFood) override;
143
147     void setMedical(int numMedical) override;
148
152     void decreasePower(int x);
153
156     ~Commander() ;
157
158
159     //Julianna added:
160
164     void setRaven(list<Raven*> r);
165
170     void setMaster(MasterOfCoin* m);
171
175     void setStrategy(Strategy* s);
176 };
177
178 #endif

```

6.8 Conditions.h

```

1  #ifndef CONDITIONS_H
2  #define CONDITIONS_H
3  #include <cstdlib>
4  #include "WarTheatre.h"
5
12 class Conditions : public WarTheatre {
13
14 public:
16     WarTheatre* myVenue;
17
18     void sendScout();
19
24     Conditions(WarTheatre* myVenue);
25 };
26
27 #endif

```

6.9 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
15 class Conlterator : public Iterator {
16 private:
19     list<Bannerman*> armyList;

```



```

20
24     list<Bannerman*>::iterator it;
25 public:
30     ConIterator(list<Bannerman*> X);
31
37     Bannerman* Current();
38
44     Bannerman* next();
45
51     bool hasNext();
52
57     bool isActive();
58 };
59
60 #endif

```

6.10 Economy.h

```

1 #ifndef ECONOMY_H
2 #define ECONOMY_H
3 #include "WarIndicators.h"
4 using namespace std;
5 #include <iostream>
6 #include "State.h"
7 class State;
8 //circular dependency with state
9 class Economy :public WarIndicators {
10
11 private:
12     State* state;
13     int currency;
14
15 public:
16     Economy(State* state,int currency);
17
18     void SetState();
19
20     void decreaseCurrency();
21
22     int getCurrency();
23
24     void removeCurrency(int i);
25
26     virtual ~Economy();
27 };
28 #endif

```

6.11 Factory.h

```

1
2 #ifndef FACTORY_H
3 #define FACTORY_H
4 #include "WarIndicators.h"
5 #include "ArmySupplies.h"
6
7 class Factory : WarIndicators {
8
9 public:
10     void operation();
11     ArmySupplies* getSupply();
12     void ~Factory();
13
14 protected:
15     virtual ArmySupplies* make() = 0;
16 private:
17     ArmySupplies* supply;
18
19 };
20
21 #endif

```

6.12 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();
20     void decreaseCurrency();
21 };
22
23 #endif
```

6.13 FoodFac.h

```
1 #ifndef FOODFAC_H
2 #define FOODFAC_H
3 #include "Factory.h"
4 #include "ArmySupplies.h"
5 #include "FoodSupp.h"
6 class FoodFac : Factory {
7
8
9 public:
13     ArmySupplies* make();
14 };
15
16 #endif
```

6.14 FoodSupp.h

```
1 #ifndef FOODSUPP_H
2 #define FOODSUPP_H
3 #include "ArmySupplies.h"
4
5 class FoodSupp : ArmySupplies {
6
7
8 public:
9     int getAmount() {
10         return 50;
11     }
12 };
13
14 #endif
```

6.15 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
8 class FoodWagon : SupplyWagon {
9
10 public:
11     ~FoodWagon()
12 };
13
14 #endif
```

6.16 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();
25 };
26
27 #endif

```

6.17 Historian.h

```

1 #ifndef HISTORIAN_H
2 #define HISTORIAN_H
3 #include "Bannerman.h"
4 #include "History.h"
5 class Historian {
6
7 private:
8     Bannerman* bannerman;
9
10 public:
11     Historian(Bannerman* b);
12
13     History* setAlly();
14
15     Bannerman* restoreAlly(History* h);
16
17     ~Historian() {delete bannerman;};
18 };
19
20 #endif

```

6.18 History.h

```

1 #ifndef HISTORY_H
2 #define HISTORY_H
3 #include "Bannerman.h"
4 #include <iostream>
5 using namespace std;
6 class History {
7
8 private:
9     Bannerman* bannerman;
10
11 public:
12     History(Bannerman* b);
13
14     Bannerman* getBannerman();
15
16     ~History() {delete bannerman;};
17 };
18
19 #endif

```

6.19 HistoryBook.h

```

1 #ifndef HISTORYBOOK_H
2 #define HISTORYBOOK_H
3 #include "History.h"
4 #include <list>
5 class HistoryBook {
6
7 private:
8     list<History*> defectedAllies;
9
10 public:
11     void add(History* h);
12
13     History* restoreAlly(History* h);
14 };
15
16 #endif

```

6.20 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
```

6.21 Kingdom.h

```
1 #ifndef KINGDOM_H
2 #define KINGDOM_H
3 using namespace std;
4 #include <vector>
5 #include "Bannerman.h"
6 #include "Economy.h"
7
12 class Kingdom {
13
14 private:
16     vector<Bannerman*> bannerman;
18     Economy* economy;
19
20 public:
25     Kingdom(Economy* economy,vector<Bannerman*> bannerman);
29     void remove(Bannerman* b);
33     void add(Bannerman* b);
35     virtual ~Kingdom();
36 };
37
38 #endif
```

6.22 Location.h

```
1 #ifndef LOCATION_H
2 #define LOCATION_H
3 #include <iostream>
4 // #include "Conditions.h" causes circular dependency?
5
6 using namespace std;
7
14 class Location : public WarTheatre {
15
16
17 public:
18
20     Location();
21
23     void sendScout();
24
26     void createVenue();
27
29     void ~Location();
30 };
31
32 #endif
```

6.23 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
17 class MasterOfCoin : Treasury {
18
19 private:
20     Economy* economy;
21
22     Raven* observer;
23
24     Strategy* strategy;
25
26
27 public:
28     MasterOfCoin(Economy* economy, Raven* observer, Strategy strategy);
29
30     void notify(WarIndicators sender);
31
32     void decreaseCurrency();
33
34     void manufacture();
35
36     ~MasterOfCoin();
37 };
38
39 #endif

```

6.24 MedicalFac.h

```

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

```

6.25 MedicalSupp.h

```

1 #ifndef MEDICALSUPP_H
2 #define MEDICALSUPP_H
3 #include "ArmySupplies.h"
4
5 class MedicalSupp : ArmySupplies {
6
7
8 public:
9     int getAmount() {
10         return 50;
11     }
12 };
13
14 #endif

```

6.26 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 class MedicalWagon : SupplyWagon {
8
9 public:
10     ~MedicalWagon()
11 };
12
13 #endif

```

6.27 Raven.h

```
1 #ifndef RAVEN_H
2 #define RAVEN_H
3
4
5
6
7
8
9 class Raven {
10
11
12 public:
13     virtual void update() = 0;
14 };
15
16
17 #endif
```

6.28 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
10
11
12
13
14
15
16 class sendRaven : Raven {
17
18 private:
19
20
21     int numFood;
22
23
24     int numMedical;
25
26
27     int numWeapons;
28
29
30     SupplyWagon** supplies;
31
32
33     Bannerman* subject;
34
35 public:
36     sendRaven(SupplyWagon** supplies, Bannerman* subject);
37
38
39     void update();
40
41
42     void checkSupplies();
43
44
45     ~sendRaven();
46 };
47
48
49 #endif
```

6.29 Siege.h

```
1 #ifndef SIEGE_H
2 #define SIEGE_H
3 #include "Strategy.h"
4
5
6
7
8
9
10 class Siege :public Strategy {
11 private:
12     int stealth;
13 public:
14     Siege(int stealth, Kingdom* myKingdom, Kingdom* enemyKingdom, Bannerman* myBannerman, Bannerman*
15         enemyBannerman, string name, int min, int minFavour);
16     bool attack(Bannerman* myBannerman, Bannerman* enemyBannerman);
17     ~Siege();
18 };
19
20
21 #endif
```

6.30 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:
18     State();
19     virtual void setContext(Economy* context);
20     virtual void decreaseCurrency()=0;
21     virtual State* getDemotionState();
22     virtual ~State();
23 };
24 #endif

```

6.31 Strategy.h

```

1 #ifndef STRATEGY_H
2 #define STRATEGY_H
3 using namespace std;
4 #include <string>
5 #include "Bannerman.h"
6 #include "WarIndicators.h"
7 #include "MasterOfCoin.h"
8 #include "Kingdom.h"
9
16 class Strategy : public WarIndicators {
17
18 protected:
19     Kingdom* myKingdom;
20     Kingdom* enemyKingdom;
21     Bannerman* myBannerman;
22     Bannerman* enemyBannerman;
23     string strategy;
24     int minFavour;
25     int minSupplies;
26 public:
27     Strategy(Kingdom* myKingdom, Kingdom* enemyKingdom, Bannerman* myBannerman, Bannerman*
28     enemyBannerman, string name, int min, int minFavour);
29     virtual bool attack(Bannerman* myBannerman, Bannerman* enemyBannerman)=0;
30     virtual string getStrategyName();
31     virtual Bannerman* getMyBannerman();
32     virtual Bannerman* getEnemyBannerman();
33     virtual ~Strategy();
34 };
35 #endif

```

6.32 SupplyWagon.h

```

1 #ifndef SUPPLYWAGON_H
2 #define SUPPLYWAGON_H
3 #include "ArmySupplies.h"
4
5 class SupplyWagon {
6
7 public:
8     void setSup(ArmySupplies* sup)=0;
9     virtual SupplyWagon* clone() = 0;
10     virtual ArmySupplies* getSupplies()=0;
11     ArmySupplies* supp;
12 };
13 #endif

```

6.33 Topology.h

```

1 #ifndef TOPOLOGY_H
2 #define TOPOLOGY_H
3
9 class Topology : public Conditions {
10

```

```

11 public:
12
15     Topology* getTopology();
16
19     void setTopology();
20
22     Topology();
23
25     void sendScout();
26 };
27
28 #endif

```

6.34 Treasury.h

```

1 #ifndef TREASURY_H
2 #define TREASURY_H
3
10 class Treasury {
11
12
13 public:
15     virtual void notify(WarIndicators* sender) = 0;
16 };
17
18 #endif

```

6.35 Troop.h

```

1 #ifndef TROOP_H
2 #define TROOP_H
3 #include "Kingdom.h"
4 #include "Strategy.h"
5 #include "Bannerman.h"
6
12 class Troop : public Bannerman {
13
14 private:
15
17     int size;
18
20     Strategy* strategy;
21
23     int damage;
24
25 public:
30     Troop(string name, int favor, int numFood, int Medical, int
        HP, WarTheatre* warZone, Strategy* strategy, MasterOfCoin* m, bool assassin, int size);
31
37     int getHP() override;
38
44     int getSize();
45
49     void attach(Raven* o) override;
50
54     void detach(Raven* o) override;
55
59     void increasePower(int boost) override;
60
65     void attack(Bannerman* myBannerman, Bannerman* enemyBannerman) override;
66
69     void decreaseWeapons() override;
70
73     void decreaseFood() override;
74
77     void decreaseMedical() override;
78
82     void changeStrategy(Strategy* strategy) override;
83
86     void increaseFavour() override;
87
90     void decreaseFavour() override;
91
92     int getFavour();
93
97     void increaseHP(int boost) override;
103     int getDamage() override;
104
109     void receiveDamage(int X) override;

```



```

110
116     int getWeapons() override;
117
123     int getFood() override;
124
130     int getMedical() override;
131
135     void setWeapons(int numWeapons) override;
136
140     void setFood(int numFood) override;
141
142
143
147     void decreasePower(int x);
148
149
153     void setMedical(int numMedical) override;
154
158     void setRaven(list<Raven*> r);
159
163     void setMaster(MasterOfCoin* m);
164
167     ~Troop();
168 };
169
170 #endif

```

6.36 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();
25 };
26
27 #endif

```

6.37 WarIndicators.h

```

1 #ifndef WARINDICATORS_H
2 #define WARINDICATORS_H
3 #include "Treasury.h"
4
10 class WarIndicators {
11
12 protected:
14     Treasury* m;
15
16 public:
19     WarIndicators(){
20         m=NULLPTR;
21     }
22
26     WarIndicators(Treasury* m){
27         this->m=m;
28     }
29
31     virtual ~WarIndicators(){
32         delete m;
33     }
34 };
35
36 #endif

```

6.38 WarTheatre.h

```

1 #ifndef WARTHEATRE_H

```

```
2 #define WARTHEATRE_H
3
4 using namespace std;
5
12 class WarTheatre {
13
14 private:
15     Strategy* strategy;
16     char venue;
17
18     string location;
19
20     int difficulty;
21
22
23
24
25
26
27 public:
28     WarTheatre(Strategy* myStrat);
29
30     Strategy* getStrategy(); //may need to make sendScout the virtual one instead
31
32     virtual void sendScout() = 0;
33
34     char decideVenue(Strategy* strategy);
35
36     WarTheatre(Strategy myStrat);
37 };
38
39 #endif
```

6.39 WeaponsFac.h

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

6.40 WeaponSupp.h

```
1 #ifndef WEAPONSUPP_H
2 #define WEAPONSUPP_H
3 #include "ArmySupplies.h"
4
5 class WeaponSupp : ArmySupplies {
6
7 public:
8     int getAmount() {
9         return 50;
10     }
11 };
12
13 #endif
```

6.41 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
8 class WeaponWagon : SupplyWagon {
9
10 public:
11
12 ~WeaponWagon()
```

```
13
14 };
15
16 #endif
```

6.42 Weather.h

```
1 #ifndef WEATHER_H
2 #define WEATHER_H
3
10 class Weather : public Conditions {
11
12 private:
14     double temp;
15
17     bool rain;
18
20     int windspeed;
21
22 public:
25     Weather(int val);
26
29     int calcEffect();
30
33     void setTemp(double t);
34
37     void makeItRain();
38
41     void setWindSpeed(int SP);
42     //void weatherReport(); ask Jules
43 };
44
45 #endif
```


Index

- ~Ambush
 - Ambush, [10](#)
- ~Assassinate
 - Assassinate, [13](#)
- ~BattleField
 - BattleField, [26](#)
- ~Factory
 - Factory, [43](#)
- ~Kingdom
 - Kingdom, [55](#)
- ~Siege
 - Siege, [65](#)
- ~State
 - State, [67](#)
- ~Strategy
 - Strategy, [70](#)
- ~Troop
 - Troop, [78](#)
- ~WarIndicators
 - WarIndicators, [88](#)
- add
 - HistoryBook, [51](#)
 - Kingdom, [55](#)
- alive
 - Assassinate, [14](#)
- Ambush, [9](#)
 - ~Ambush, [10](#)
 - Ambush, [10](#)
 - attack, [11](#)
 - stealth, [11](#)
- armyList
 - Conlterator, [41](#)
- ArmySupplies, [11](#)
- assassin
 - Bannerman, [24](#)
- Assassinate, [12](#)
 - ~Assassinate, [13](#)
 - alive, [14](#)
 - Assassinate, [13](#)
 - attack, [13](#)
 - stealth, [14](#)
- attach
 - Bannerman, [16](#)
 - Commander, [30](#)
 - Troop, [78](#)
- attack
 - Ambush, [11](#)
 - Assassinate, [13](#)
 - Bannerman, [16](#)

- BattleField, [27](#)
- Commander, [30](#)
- Siege, [66](#)
- Strategy, [70](#)
- Troop, [79](#)
- Bannerman, [14](#)
 - assassin, [24](#)
 - attach, [16](#)
 - attack, [16](#)
 - changeStrategy, [17](#)
 - damage, [24](#)
 - decreaseFavour, [17](#)
 - decreaseFood, [17](#)
 - decreaseMedical, [17](#)
 - decreasePower, [17](#)
 - decreaseWeapons, [18](#)
 - detach, [18](#)
 - favour, [24](#)
 - getDamage, [18](#)
 - getFavour, [18](#)
 - getFood, [19](#)
 - getHP, [19](#)
 - getMedical, [19](#)
 - getName, [19](#)
 - getWeapons, [20](#)
 - HP, [24](#)
 - increaseFavour, [20](#)
 - increaseHP, [20](#)
 - increasePower, [21](#)
 - m, [24](#)
 - name, [24](#)
 - numFood, [24](#)
 - numMedical, [25](#)
 - numWeapons, [25](#)
 - ravenList, [25](#)
 - receiveDamage, [21](#)
 - setFood, [21](#)
 - setMaster, [21](#)
 - setMedical, [22](#)
 - setRaven, [22](#)
 - setWeapons, [22](#)
 - strategy, [25](#)
- bannerman
 - Kingdom, [56](#)
- BattleField, [25](#)
 - ~BattleField, [26](#)
 - attack, [27](#)
 - BattleField, [26](#)

- calcEffect
 - Weather, 95
- changeStrategy
 - Bannerman, 17
 - Commander, 31
 - Troop, 79
- Class, 27
- clone
 - SupplyWagon, 73
- Commander, 28
 - attach, 30
 - attack, 30
 - changeStrategy, 31
 - Commander, 30
 - decreaseFavour, 31
 - decreaseFood, 31
 - decreaseMedical, 31
 - decreasePower, 31
 - decreaseWeapons, 32
 - detach, 32
 - getDamage, 32
 - getFavour, 32
 - getFood, 33
 - getHP, 33
 - getMedical, 33
 - getTroops, 33
 - getWeapons, 34
 - increaseFavour, 34
 - increaseHP, 34
 - increasePower, 34
 - receiveDamage, 35
 - setFood, 35
 - setMaster, 35
 - setMedical, 36
 - setRaven, 36
 - setStrategy, 36
 - setWeapons, 37
- Conditions, 37
 - Conditions, 38
 - myVenue, 38
 - sendScout, 38
- ConIterator, 39
 - armyList, 41
 - ConIterator, 40
 - Current, 40
 - hasNext, 40
 - isActive, 40
 - it, 41
 - next, 41
- context
 - State, 68
- Current
 - ConIterator, 40
 - Iterator, 53
- damage
 - Bannerman, 24
 - Troop, 85
- decideVenue
 - WarTheatre, 90
- decreaseCurrency
 - FailedState, 45
 - HealthyState, 48
 - State, 67
 - UnstableState, 87
- decreaseFavour
 - Bannerman, 17
 - Commander, 31
 - Troop, 79
- decreaseFood
 - Bannerman, 17
 - Commander, 31
 - Troop, 79
- decreaseMedical
 - Bannerman, 17
 - Commander, 31
 - Troop, 80
- decreasePower
 - Bannerman, 17
 - Commander, 31
 - Troop, 80
- decreaseWeapons
 - Bannerman, 18
 - Commander, 32
 - Troop, 80
- detach
 - Bannerman, 18
 - Commander, 32
 - Troop, 80
- difficulty
 - WarTheatre, 91
- Economy, 42
 - state, 42
- economy
 - Kingdom, 56
- enemyBannerman
 - Strategy, 72
- enemyKingdom
 - Strategy, 72
- Factory, 42
 - ~Factory, 43
 - getSupply, 43
 - make, 43
 - operation, 44
- FailedState, 44
 - decreaseCurrency, 45
- favour
 - Bannerman, 24
- FoodFac, 45
 - make, 46
- FoodSupp, 46
 - getAmount, 46
- FoodWagon, 47
- getAmount
 - FoodSupp, 46

- MedicalSupp, 61
- WeaponSupp, 93
- getBannerman
 - History, 51
- getDamage
 - Bannerman, 18
 - Commander, 32
 - Troop, 81
- getDemotionState
 - HealthyState, 48
 - State, 67
 - UnstableState, 87
- getEnemyBannerman
 - Strategy, 71
- getFavour
 - Bannerman, 18
 - Commander, 32
 - Troop, 81
- getFood
 - Bannerman, 19
 - Commander, 33
 - Troop, 81
- getHP
 - Bannerman, 19
 - Commander, 33
 - Troop, 81
- getMedical
 - Bannerman, 19
 - Commander, 33
 - Troop, 82
- getMyBannerman
 - Strategy, 71
- getName
 - Bannerman, 19
- getSize
 - Troop, 82
- getStrategy
 - WarTheatre, 90
- getStrategyName
 - Strategy, 71
- getSupplies
 - SupplyWagon, 73
- getSupply
 - Factory, 43
- getTopology
 - Topology, 75
- getTroops
 - Commander, 33
- getWeapons
 - Bannerman, 20
 - Commander, 34
 - Troop, 82
- hasNext
 - Conlterator, 40
 - Iterator, 53
- HealthyState, 47
 - decreaseCurrency, 48
 - getDemotionState, 48
- Historian, 49
 - Historian, 49
 - restoreAlly, 49
 - setAlly, 50
- History, 50
 - getBannerman, 51
 - History, 50
- HistoryBook, 51
 - add, 51
 - restoreAlly, 52
- HP
 - Bannerman, 24
- increaseFavour
 - Bannerman, 20
 - Commander, 34
 - Troop, 82
- increaseHP
 - Bannerman, 20
 - Commander, 34
 - Troop, 83
- increasePower
 - Bannerman, 21
 - Commander, 34
 - Troop, 83
- isActive
 - Conlterator, 40
 - Iterator, 53
- it
 - Conlterator, 41
- Iterator, 52
 - Current, 53
 - hasNext, 53
 - isActive, 53
 - next, 54
- Kingdom, 54
 - ~Kingdom, 55
 - add, 55
 - bannerman, 56
 - economy, 56
 - Kingdom, 55
 - remove, 56
- Location, 57
 - sendScout, 57
- location
 - WarTheatre, 91
- m
 - Bannerman, 24
 - WarIndicators, 89
- make
 - Factory, 43
 - FoodFac, 46
 - MedicalFac, 60
 - WeaponsFac, 92
- MasterOfCoin, 58
 - MasterOfCoin, 59

- notify, [59](#)
- MedicalFac, [59](#)
 - make, [60](#)
- MedicalSupp, [60](#)
 - getAmount, [61](#)
- MedicalWagon, [61](#)
- minFavour
 - Strategy, [72](#)
- minSupplies
 - Strategy, [72](#)
- myBannerman
 - Strategy, [72](#)
- myKingdom
 - Strategy, [72](#)
- myVenue
 - Conditions, [38](#)
- name
 - Bannerman, [24](#)
- next
 - Conliterator, [41](#)
 - Iterator, [54](#)
- notify
 - MasterOfCoin, [59](#)
- numFood
 - Bannerman, [24](#)
- numMedical
 - Bannerman, [25](#)
- numWeapons
 - Bannerman, [25](#)
- operation
 - Factory, [44](#)
- rain
 - Weather, [96](#)
- Raven, [61](#)
 - update, [62](#)
- ravenList
 - Bannerman, [25](#)
- receiveDamage
 - Bannerman, [21](#)
 - Commander, [35](#)
 - Troop, [83](#)
- remove
 - Kingdom, [56](#)
- restoreAlly
 - Historian, [49](#)
 - HistoryBook, [52](#)
- sendRaven, [62](#)
 - sendRaven, [63](#)
 - update, [64](#)
- sendScout
 - Conditions, [38](#)
 - Location, [57](#)
 - Topology, [75](#)
 - WarTheatre, [91](#)
- setAlly
 - Historian, [50](#)
- setContext
 - State, [68](#)
- setFood
 - Bannerman, [21](#)
 - Commander, [35](#)
 - Troop, [84](#)
- setMaster
 - Bannerman, [21](#)
 - Commander, [35](#)
 - Troop, [84](#)
- setMedical
 - Bannerman, [22](#)
 - Commander, [36](#)
 - Troop, [84](#)
- setRaven
 - Bannerman, [22](#)
 - Commander, [36](#)
 - Troop, [85](#)
- setStrategy
 - Commander, [36](#)
- setSup
 - SupplyWagon, [74](#)
- setTemp
 - Weather, [95](#)
- setWeapons
 - Bannerman, [22](#)
 - Commander, [37](#)
 - Troop, [85](#)
- setWindSpeed
 - Weather, [95](#)
- Siege, [64](#)
 - ~Siege, [65](#)
 - attack, [66](#)
 - Siege, [65](#)
- size
 - Troop, [85](#)
- State, [66](#)
 - ~State, [67](#)
 - context, [68](#)
 - decreaseCurrency, [67](#)
 - getDemotionState, [67](#)
 - setContext, [68](#)
- state
 - Economy, [42](#)
- stealth
 - Ambush, [11](#)
 - Assassinate, [14](#)
- Strategy, [69](#)
 - ~Strategy, [70](#)
 - attack, [70](#)
 - enemyBannerman, [72](#)
 - enemyKingdom, [72](#)
 - getEnemyBannerman, [71](#)
 - getMyBannerman, [71](#)
 - getStrategyName, [71](#)
 - minFavour, [72](#)
 - minSupplies, [72](#)

- myBannerman, 72
- myKingdom, 72
- Strategy, 70
- strategy, 72
- strategy
 - Bannerman, 25
 - Strategy, 72
 - Troop, 85
 - WarTheatre, 91
- SupplyWagon, 73
 - clone, 73
 - getSupplies, 73
 - setSup, 74
- temp
 - Weather, 96
- Topology, 74
 - getTopology, 75
 - sendScout, 75
- Treasury, 76
- Troop, 76
 - ~Troop, 78
 - attach, 78
 - attack, 79
 - changeStrategy, 79
 - damage, 85
 - decreaseFavour, 79
 - decreaseFood, 79
 - decreaseMedical, 80
 - decreasePower, 80
 - decreaseWeapons, 80
 - detach, 80
 - getDamage, 81
 - getFavour, 81
 - getFood, 81
 - getHP, 81
 - getMedical, 82
 - getSize, 82
 - getWeapons, 82
 - increaseFavour, 82
 - increaseHP, 83
 - increasePower, 83
 - receiveDamage, 83
 - setFood, 84
 - setMaster, 84
 - setMedical, 84
 - setRaven, 85
 - setWeapons, 85
 - size, 85
 - strategy, 85
- UnstableState, 86
 - decreaseCurrency, 87
 - getDemotionState, 87
- update
 - Raven, 62
 - sendRaven, 64
- venue
 - WarTheatre, 91
- WarIndicators, 87
 - ~WarIndicators, 88
 - m, 89
 - WarIndicators, 88
- WarTheatre, 89
 - decideVenue, 90
 - difficulty, 91
 - getStrategy, 90
 - location, 91
 - sendScout, 91
 - strategy, 91
 - venue, 91
 - WarTheatre, 90
- WeaponsFac, 92
 - make, 92
- WeaponSupp, 93
 - getAmount, 93
- WeaponWagon, 93
- Weather, 94
 - calcEffect, 95
 - rain, 96
 - setTemp, 95
 - setWindSpeed, 95
 - temp, 96
 - Weather, 95
 - windspeed, 96
- windspeed
 - Weather, 96