

COS214 Spice Girls

0.1

Generated by Doxygen 1.9.6

1 Hierarchical Index	1
1.1 Class Hierarchy	1
2 Class Index	3
2.1 Class List	3
3 File Index	5
3.1 File List	5
4 Class Documentation	7
4.1 AddOn Class Reference	7
4.1.1 Detailed Description	8
4.1.2 Constructor & Destructor Documentation	8
4.1.2.1 AddOn()	8
4.1.3 Member Function Documentation	9
4.1.3.1 clone()	9
4.1.3.2 dealDamage()	9
4.1.3.3 getAlliance()	10
4.1.3.4 getDamage()	10
4.1.3.5 getEntity()	11
4.1.3.6 getHealth()	11
4.1.3.7 getType()	12
4.1.3.8 getValue()	12
4.1.3.9 setAlliance()	12
4.1.3.10 setDamage()	13
4.1.3.11 setEntity()	14
4.1.3.12 setHealth()	14
4.1.3.13 setType()	15
4.1.3.14 setValue()	15
4.1.3.15 takeDamage()	16
4.1.4 Member Data Documentation	17
4.1.4.1 entity	17
4.1.4.2 value	17
4.2 AerialType Class Reference	17
4.2.1 Detailed Description	18
4.2.2 Constructor & Destructor Documentation	18
4.2.2.1 AerialType()	18
4.2.3 Member Function Documentation	18
4.2.3.1 clone()	18
4.2.3.2 getTypeDesc()	19
4.3 Aggressive Class Reference	19
4.3.1 Detailed Description	19
4.3.2 Constructor & Destructor Documentation	20

4.3.2.1 Aggressive()	20
4.3.3 Member Function Documentation	20
4.3.3.1 clone()	20
4.3.3.2 performStrat()	20
4.4 Alliance Class Reference	21
4.4.1 Detailed Description	22
4.4.2 Constructor & Destructor Documentation	22
4.4.2.1 Alliance() [1/2]	22
4.4.2.2 Alliance() [2/2]	22
4.4.2.3 ~Alliance()	23
4.4.3 Member Function Documentation	23
4.4.3.1 addCountry()	23
4.4.3.2 addFactory()	24
4.4.3.3 addReserveEntity()	24
4.4.3.4 clone()	25
4.4.3.5 considerPeace()	25
4.4.3.6 getActive()	26
4.4.3.7 getID()	26
4.4.3.8 getReserveEntities()	26
4.4.3.9 numRemainingEntities()	27
4.4.3.10 offerPeace()	27
4.4.3.11 runFactories()	28
4.4.3.12 setActiveStatus()	28
4.4.3.13 setNegotiator()	28
4.4.3.14 surrender()	29
4.5 AquaticType Class Reference	30
4.5.1 Detailed Description	30
4.5.2 Constructor & Destructor Documentation	30
4.5.2.1 AquaticType()	30
4.5.3 Member Function Documentation	30
4.5.3.1 clone()	31
4.5.3.2 getTypeDesc()	31
4.6 Area Class Reference	32
4.6.1 Detailed Description	32
4.6.2 Constructor & Destructor Documentation	32
4.6.2.1 Area()	32
4.6.2.2 ~Area()	33
4.6.3 Member Function Documentation	33
4.6.3.1 addGeneral()	33
4.6.3.2 clone()	33
4.6.3.3 getAreaName()	34
4.6.3.4 isKeyPoint()	34

4.6.3.5 simulateBattle()	34
4.7 Armour Class Reference	34
4.7.1 Detailed Description	35
4.7.2 Constructor & Destructor Documentation	35
4.7.2.1 Armour()	35
4.7.3 Member Function Documentation	35
4.7.3.1 clone()	35
4.7.3.2 dealDamage()	36
4.7.3.3 takeDamage()	37
4.8 Cloudy Class Reference	37
4.8.1 Detailed Description	38
4.8.2 Constructor & Destructor Documentation	38
4.8.2.1 Cloudy()	38
4.8.3 Member Function Documentation	38
4.8.3.1 clone()	38
4.8.3.2 getWeather()	39
4.8.3.3 handleChange()	39
4.9 Country Class Reference	40
4.9.1 Detailed Description	40
4.9.2 Constructor & Destructor Documentation	40
4.9.2.1 Country()	40
4.9.3 Member Function Documentation	41
4.9.3.1 clone()	41
4.9.3.2 getID()	41
4.9.3.3 getName()	42
4.9.3.4 setID()	42
4.9.3.5 setName()	42
4.10 Defensive Class Reference	43
4.10.1 Detailed Description	43
4.10.2 Constructor & Destructor Documentation	43
4.10.2.1 Defensive()	44
4.10.3 Member Function Documentation	44
4.10.3.1 clone()	44
4.10.3.2 performStrat()	44
4.11 EasySetup Class Reference	45
4.11.1 Detailed Description	45
4.11.2 Constructor & Destructor Documentation	45
4.11.2.1 EasySetup()	45
4.11.3 Member Function Documentation	45
4.11.3.1 loadPrevSave()	46
4.11.3.2 loadSpecificSave()	46
4.11.3.3 runSimulation()	46

4.11.3.4 saveSimulationSetup()	46
4.11.3.5 setupSimulation()	47
4.12 Entity Class Reference	50
4.12.1 Detailed Description	50
4.12.2 Constructor & Destructor Documentation	51
4.12.2.1 Entity() [1/2]	51
4.12.2.2 Entity() [2/2]	51
4.12.3 Member Function Documentation	51
4.12.3.1 clone()	51
4.12.3.2 dealDamage()	52
4.12.3.3 getAlliance()	52
4.12.3.4 getDamage()	53
4.12.3.5 getHealth()	53
4.12.3.6 getType()	54
4.12.3.7 setAlliance()	54
4.12.3.8 setDamage()	55
4.12.3.9 setHealth()	55
4.12.3.10 setType()	56
4.12.3.11 takeDamage()	57
4.13 Factory Class Reference	58
4.13.1 Detailed Description	58
4.13.2 Constructor & Destructor Documentation	58
4.13.2.1 Factory()	58
4.13.2.2 ~Factory()	59
4.13.3 Member Function Documentation	59
4.13.3.1 clone()	59
4.13.3.2 createEntity()	60
4.13.3.3 getAddOn()	60
4.13.3.4 getType()	60
4.13.3.5 setAddOns()	60
4.13.3.6 setType()	61
4.14 General Class Reference	62
4.14.1 Detailed Description	62
4.14.2 Constructor & Destructor Documentation	62
4.14.2.1 General()	62
4.14.3 Member Function Documentation	62
4.14.3.1 getAlliance()	63
4.14.3.2 initiateStrategy()	63
4.14.3.3 setStrategy()	64
4.15 KeyPoint Class Reference	64
4.15.1 Detailed Description	65
4.15.2 Constructor & Destructor Documentation	65

4.15.2.1 KeyPoint() [1/2]	65
4.15.2.2 KeyPoint() [2/2]	66
4.15.2.3 ~KeyPoint()	66
4.15.3 Member Function Documentation	66
4.15.3.1 addEntity()	66
4.15.3.2 addGeneral()	67
4.15.3.3 changeWeather()	67
4.15.3.4 clearBattlefield()	68
4.15.3.5 clone()	69
4.15.3.6 getWeather()	69
4.15.3.7 isKeyPoint()	70
4.15.3.8 moveEntitiesInto()	70
4.15.3.9 moveEntitiesOutOf()	71
4.15.3.10 removeGeneral()	72
4.15.3.11 setWeather()	72
4.15.3.12 simulateBattle()	73
4.16 Negotiator Class Reference	74
4.16.1 Detailed Description	74
4.16.2 Constructor & Destructor Documentation	74
4.16.2.1 Negotiator()	74
4.16.2.2 ~Negotiator()	75
4.16.3 Member Function Documentation	75
4.16.3.1 addAlliance()	75
4.16.3.2 getNumAlliances()	75
4.16.3.3 removeAlliance()	76
4.16.3.4 sendPeace()	76
4.17 Passive Class Reference	77
4.17.1 Detailed Description	77
4.17.2 Constructor & Destructor Documentation	77
4.17.2.1 Passive()	78
4.17.3 Member Function Documentation	78
4.17.3.1 clone()	78
4.17.3.2 performStrat()	78
4.18 Personnel Class Reference	79
4.18.1 Detailed Description	79
4.18.2 Constructor & Destructor Documentation	79
4.18.2.1 Personnel()	79
4.18.3 Member Function Documentation	80
4.18.3.1 clone()	80
4.18.3.2 dealDamage()	80
4.18.3.3 takeDamage()	81
4.19 PersonnelFactory Class Reference	82

4.19.1 Detailed Description	82
4.19.2 Constructor & Destructor Documentation	82
4.19.2.1 PersonnelFactory()	82
4.19.3 Member Function Documentation	83
4.19.3.1 clone()	83
4.19.3.2 createEntity()	83
4.20 Piercing Class Reference	84
4.20.1 Detailed Description	85
4.20.2 Constructor & Destructor Documentation	85
4.20.2.1 Piercing()	85
4.20.3 Member Function Documentation	85
4.20.3.1 clone()	85
4.20.3.2 dealDamage()	86
4.20.3.3 takeDamage()	86
4.21 Rainy Class Reference	87
4.21.1 Detailed Description	87
4.21.2 Constructor & Destructor Documentation	88
4.21.2.1 Rainy()	88
4.21.3 Member Function Documentation	88
4.21.3.1 clone()	88
4.21.3.2 getWeather()	88
4.21.3.3 handleChange()	89
4.22 RoundStats Class Reference	89
4.22.1 Detailed Description	90
4.22.2 Member Function Documentation	90
4.22.2.1 clearStats()	90
4.22.2.2 toString()	90
4.22.3 Member Data Documentation	90
4.22.3.1 damageDone	90
4.22.3.2 entityMovementInformation	91
4.22.3.3 keyPointInformation	91
4.22.3.4 numContestedPoints	91
4.22.3.5 numEntitiesCreated	91
4.22.3.6 numEntitiesDestroyed	91
4.22.3.7 numLosingPoints	91
4.22.3.8 numWinningPoints	92
4.23 SaveArchive Class Reference	92
4.23.1 Detailed Description	92
4.23.2 Constructor & Destructor Documentation	92
4.23.2.1 SaveArchive()	93
4.23.3 Member Function Documentation	93
4.23.3.1 addNewSave()	93

4.23.3.2 clearSaveList()	93
4.23.3.3 deleteSave()	94
4.23.3.4 getLastSave()	95
4.23.3.5 getSave()	95
4.24 Strategy Class Reference	96
4.24.1 Detailed Description	97
4.24.2 Constructor & Destructor Documentation	97
4.24.2.1 Strategy()	97
4.24.2.2 ~Strategy()	97
4.24.3 Member Function Documentation	97
4.24.3.1 clone()	97
4.24.3.2 performStrat()	97
4.24.4 Member Data Documentation	98
4.24.4.1 strategy	98
4.25 Sunny Class Reference	98
4.25.1 Detailed Description	99
4.25.2 Constructor & Destructor Documentation	99
4.25.2.1 Sunny()	99
4.25.3 Member Function Documentation	99
4.25.3.1 clone()	99
4.25.3.2 getWeather()	100
4.25.3.3 handleChange()	100
4.26 Support Class Reference	101
4.26.1 Detailed Description	101
4.26.2 Constructor & Destructor Documentation	101
4.26.2.1 Support()	101
4.26.3 Member Function Documentation	102
4.26.3.1 clone()	102
4.26.3.2 dealDamage()	102
4.26.3.3 takeDamage()	103
4.27 SupportFactory Class Reference	104
4.27.1 Detailed Description	104
4.27.2 Constructor & Destructor Documentation	104
4.27.2.1 SupportFactory()	104
4.27.3 Member Function Documentation	105
4.27.3.1 clone()	105
4.27.3.2 createEntity()	105
4.28 TerrainType Class Reference	106
4.28.1 Detailed Description	107
4.28.2 Constructor & Destructor Documentation	107
4.28.2.1 TerrainType()	107
4.28.3 Member Function Documentation	107

4.28.3.1 clone()	107
4.28.3.2 getTypeDesc()	108
4.29 Type Class Reference	108
4.29.1 Detailed Description	109
4.29.2 Constructor & Destructor Documentation	109
4.29.2.1 Type()	109
4.29.3 Member Function Documentation	109
4.29.3.1 clone()	109
4.29.3.2 getTypeDesc()	110
4.30 Vehicle Class Reference	110
4.30.1 Detailed Description	111
4.30.2 Constructor & Destructor Documentation	111
4.30.2.1 Vehicle()	111
4.30.3 Member Function Documentation	111
4.30.3.1 clone()	111
4.30.3.2 dealDamage()	112
4.30.3.3 takeDamage()	113
4.31 VehicleFactory Class Reference	113
4.31.1 Detailed Description	114
4.31.2 Constructor & Destructor Documentation	114
4.31.2.1 VehicleFactory()	114
4.31.3 Member Function Documentation	114
4.31.3.1 clone()	114
4.31.3.2 createEntity()	115
4.32 WarEngine Class Reference	116
4.32.1 Detailed Description	116
4.32.2 Constructor & Destructor Documentation	116
4.32.2.1 WarEngine() [1/2]	117
4.32.2.2 WarEngine() [2/2]	117
4.32.2.3 ~WarEngine()	117
4.32.3 Member Function Documentation	117
4.32.3.1 addAlliance()	118
4.32.3.2 getInstance()	118
4.32.3.3 loadSave()	118
4.32.3.4 operator=()	119
4.32.3.5 saveState()	119
4.32.3.6 setWarTheatre()	119
4.32.3.7 simulate()	120
4.33 WarEngineMemento Class Reference	121
4.33.1 Detailed Description	121
4.33.2 Friends And Related Function Documentation	121
4.33.2.1 WarEngine	122

4.34 WarEngineState Class Reference	122
4.34.1 Detailed Description	122
4.34.2 Constructor & Destructor Documentation	123
4.34.2.1 WarEngineState()	123
4.34.2.2 ~WarEngineState()	123
4.34.3 Member Function Documentation	123
4.34.3.1 clone()	123
4.34.3.2 getAlliances()	124
4.34.3.3 getArea()	124
4.34.3.4 setAlliances()	124
4.34.3.5 setArea()	125
4.34.4 Friends And Related Function Documentation	126
4.34.4.1 WarEngine	126
4.35 WarTheatre Class Reference	126
4.35.1 Detailed Description	126
4.35.2 Constructor & Destructor Documentation	127
4.35.2.1 WarTheatre()	127
4.35.2.2 ~WarTheatre()	127
4.35.3 Member Function Documentation	127
4.35.3.1 addArea()	127
4.35.3.2 addGeneral()	128
4.35.3.3 clone()	128
4.35.3.4 isKeyPoint()	129
4.35.3.5 simulateBattle()	129
4.36 Weather Class Reference	130
4.36.1 Detailed Description	131
4.36.2 Constructor & Destructor Documentation	131
4.36.2.1 Weather()	131
4.36.2.2 ~Weather()	131
4.36.3 Member Function Documentation	131
4.36.3.1 clone()	131
4.36.3.2 getMultiplier()	132
4.36.3.3 getWeather()	132
4.36.3.4 handleChange()	132
4.36.4 Member Data Documentation	133
4.36.4.1 multiplier	133
5 File Documentation	135
5.1 AddOn.cpp	135
5.2 AddOn.h	136
5.3 AddOnTest.h	136
5.4 AerialType.cpp	139

5.5 AerialType.h	139
5.6 Aggressive.cpp	139
5.7 Aggressive.h	140
5.8 Alliance.cpp	140
5.9 Alliance.h	141
5.10 AquaticType.cpp	142
5.11 AquaticType.h	142
5.12 Area.cpp	142
5.13 Area.h	143
5.14 Armour.cpp	143
5.15 Armour.h	143
5.16 Cloudy.cpp	144
5.17 Cloudy.h	144
5.18 Country.cpp	144
5.19 Country.h	145
5.20 Defensive.cpp	145
5.21 Defensive.h	145
5.22 EasySetup.cpp	146
5.23 EasySetup.h	149
5.24 Entity.cpp	150
5.25 Entity.h	150
5.26 Factory.cpp	151
5.27 Factory.h	151
5.28 General.cpp	152
5.29 General.h	152
5.30 KeyPoint.cpp	153
5.31 KeyPoint.h	155
5.32 Negotiator.cpp	155
5.33 Negotiator.h	156
5.34 NegotiatorTest.h	156
5.35 Passive.cpp	157
5.36 Passive.h	157
5.37 Personnel.cpp	158
5.38 Personnel.h	158
5.39 PersonnelFactory.cpp	158
5.40 PersonnelFactory.h	159
5.41 Piercing.cpp	159
5.42 Piercing.h	159
5.43 Rainy.cpp	159
5.44 Rainy.h	160
5.45 RoundStats.cpp	160
5.46 RoundStats.h	161

5.47 SaveArchive.cpp	161
5.48 SaveArchive.h	162
5.49 Strategy.cpp	162
5.50 Strategy.h	162
5.51 Sunny.cpp	162
5.52 Sunny.h	163
5.53 Support.cpp	163
5.54 Support.h	163
5.55 SupportFactory.cpp	164
5.56 SupportFactory.h	164
5.57 TerrainType.cpp	164
5.58 TerrainType.h	165
5.59 testmain.cpp	165
5.60 Type.cpp	166
5.61 Type.h	166
5.62 Vehicle.cpp	166
5.63 Vehicle.h	167
5.64 VehicleFactory.cpp	167
5.65 VehicleFactory.h	167
5.66 WarEngine.cpp	168
5.67 WarEngine.h	169
5.68 WarEngineMemento.cpp	169
5.69 WarEngineMemento.h	170
5.70 WarEngineState.cpp	170
5.71 WarEngineState.h	171
5.72 WarTheatre.cpp	171
5.73 WarTheatre.h	172
5.74 Weather.cpp	172
5.75 Weather.h	172
Index	175

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Alliance	21
Area	32
KeyPoint	64
WarTheatre	126
Country	40
EasySetup	45
Entity	50
AddOn	7
Armour	34
Piercing	84
Personnel	79
Support	101
Vehicle	110
Factory	58
PersonnelFactory	82
SupportFactory	104
VehicleFactory	113
General	62
Negotiator	74
RoundStats	89
SaveArchive	92
Strategy	96
Aggressive	19
Defensive	43
Passive	77
Type	108
AerialType	17
AquaticType	30
TerrainType	106
WarEngine	116
WarEngineMemento	121
WarEngineState	122
Weather	130
Cloudy	37
Rainy	87
Sunny	98

Chapter 2

Class Index

2.1 Class List

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

AddOn		
AddOn	class	7
AerialType		
AerialType	class	17
Aggressive	19
Alliance	21
AquaticType		
AquaticType	class	30
Area	32
Armour		
Armour	class	34
Cloudy	37
Country	40
Defensive	43
EasySetup	45
Entity		
Entity	class	50
Factory		
Factory	class	58
General	62
KeyPoint		
Keypoint	class	64
Negotiator	74
Passive	77
Personnel		
Personnel	class	79
PersonnelFactory		
PersonnelFactory	class	82
Piercing		
Piercing	class	84
Rainy	87
RoundStats	89
SaveArchive		
Stores a list of mementos containing simulation state	92
Strategy	96

Sunny	98
Support	
Support class	101
SupportFactory	
SupportFactory class	104
TerrainType	
TerrainType class	106
Type	
Type class	108
Vehicle	
Vehicle class	110
VehicleFactory	
VehicleFactory class	113
WarEngine	116
WarEngineMemento	121
WarEngineState	
Class for storing current state of entire simulation	122
WarTheatre	126
Weather	130

Chapter 3

File Index

3.1 File List

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

AddOn.cpp	??
AddOn.h	??
AddOnTest.h	??
AerialType.cpp	??
AerialType.h	??
Aggressive.cpp	??
Aggressive.h	??
Alliance.cpp	??
Alliance.h	??
AquaticType.cpp	??
AquaticType.h	??
Area.cpp	??
Area.h	??
Armour.cpp	??
Armour.h	??
Cloudy.cpp	??
Cloudy.h	??
Country.cpp	??
Country.h	??
Defensive.cpp	??
Defensive.h	??
EasySetup.cpp	??
EasySetup.h	??
Entity.cpp	??
Entity.h	??
Factory.cpp	??
Factory.h	??
General.cpp	??
General.h	??
KeyPoint.cpp	??
KeyPoint.h	??
Negotiator.cpp	??
Negotiator.h	??
NegotiatorTest.h	??
Passive.cpp	??

Passive.h	??
Personnel.cpp	??
Personnel.h	??
PersonnelFactory.cpp	??
PersonnelFactory.h	??
Piercing.cpp	??
Piercing.h	??
Rainy.cpp	??
Rainy.h	??
RoundStats.cpp	??
RoundStats.h	??
SaveArchive.cpp	??
SaveArchive.h	??
Strategy.cpp	??
Strategy.h	??
Sunny.cpp	??
Sunny.h	??
Support.cpp	??
Support.h	??
SupportFactory.cpp	??
SupportFactory.h	??
TerrainType.cpp	??
TerrainType.h	??
testmain.cpp	??
Type.cpp	??
Type.h	??
Vehicle.cpp	??
Vehicle.h	??
VehicleFactory.cpp	??
VehicleFactory.h	??
WarEngine.cpp	??
WarEngine.h	??
WarEngineMemento.cpp	??
WarEngineMemento.h	??
WarEngineState.cpp	??
WarEngineState.h	??
WarTheatre.cpp	??
WarTheatre.h	??
Weather.cpp	??
Weather.h	??

Chapter 4

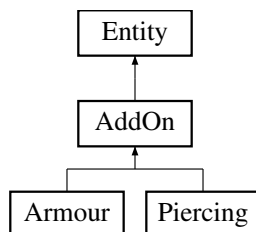
Class Documentation

4.1 AddOn Class Reference

[AddOn](#) class.

```
#include <AddOn.h>
```

Inheritance diagram for AddOn:



Public Member Functions

- [AddOn](#) (int value)
Instantiates an [AddOn](#).
- void [setValue](#) (int value)
Sets the [AddOn](#)'s value attribute.
- int [getValue](#) ()
Returns the [AddOn](#)'s value attribute.
- void [setEntity](#) ([Entity](#) *entity)
Sets the [AddOn](#)'s entity attribute.
- [Entity](#) * [getEntity](#) ()
Returns the [AddOn](#)'s entity attribute.
- [Type](#) * [getType](#) ()
Returns entities type state.
- void [setType](#) ([Type](#) *type)
Sets the entities type state.
- [Alliance](#) * [getAlliance](#) ()
Returns entities alliance.

- void [setAlliance](#) ([Alliance](#) *alliance)
Sets the entities alliance.
- int [getHealth](#) ()
Returns entities health.
- void [setHealth](#) (int health)
Sets the entities health.
- int [getDamage](#) ()
Returns entities damage.
- void [setDamage](#) (int damage)
Sets the entities damage.
- virtual void [takeDamage](#) (int damage)=0
Reduces health from the [Entity](#) object.
- virtual void [dealDamage](#) ([Entity](#) *entity)=0
Inflicts damage onto another entity.
- virtual [AddOn](#) * [clone](#) ()=0
Clones the [AddOn](#)'s object and returns the the cloned object.

Protected Attributes

- int [value](#)
- [Entity](#) * [entity](#)

4.1.1 Detailed Description

[AddOn](#) class.

Used to add additional functionality to [Entity](#) objects.

Definition at line 10 of file [AddOn.h](#).

4.1.2 Constructor & Destructor Documentation

4.1.2.1 AddOn()

```
AddOn::AddOn (
    int value )
```

Instantiates an [AddOn](#).

Parameters

value	must be an int
-----------------------	----------------

Definition at line 4 of file [AddOn.cpp](#).

```
00004      : Entity() {
```

```
00005      this->value = value;
00006      entity = NULL;
00007 }
```

4.1.3 Member Function Documentation

4.1.3.1 clone()

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

Clones the [AddOn](#)'s object and returns the the cloned object.

PostConditions:

- The returns the cloned object of [AddOn](#) object

Returns

[AddOn](#)* the return object

Implements [Entity](#).

Implemented in [Armour](#), and [Piercing](#).

4.1.3.2 dealDamage()

```
virtual void AddOn::dealDamage (
    Entity * entity ) [pure virtual]
```

Inflicts damage onto another entity.

Preconditions:

- entity must be an [Entity](#)*

Postconditions:

- Reduces the health of the entity

Parameters

<i>entity</i>	must be an Entity *
---------------	-------------------------------------

Returns

void

Implements [Entity](#).

Implemented in [Armour](#), and [Piercing](#).

4.1.3.3 getAlliance()

```
Alliance * AddOn::getAlliance ( ) [virtual]
```

Returns entities alliance.

Postconditions:

- Returns the alliance

Returns

Type* The alliance of the entity object

Reimplemented from [Entity](#).

Definition at line 37 of file [AddOn.cpp](#).

```
00037     {  
00038         return entity->getAlliance();  
00039     }
```

4.1.3.4 getDamage()

```
int AddOn::getDamage ( ) [virtual]
```

Returns entities damage.

Postconditions:

- Returns the damage

Returns

int The damage of the entity object

Reimplemented from [Entity](#).

Definition at line 53 of file [AddOn.cpp](#).

```
00053     {  
00054         return entity->getDamage();  
00055     }
```


4.1.3.5 getEntity()

```
Entity * AddOn::getEntity ( )
```

Returns the [AddOn](#)'s entity attribute.

Postconditions:

- Returns the entity attribute of the [AddOn](#) object

Returns

Entity* The entity of the [AddOn](#)

Definition at line 25 of file [AddOn.cpp](#).

```
00025 {  
00026     return this->entity;  
00027 }
```

4.1.3.6 getHealth()

```
int AddOn::getHealth ( ) [virtual]
```

Returns entities health.

Postconditions:

- Returns the health

Returns

int The health of the entity object

Reimplemented from [Entity](#).

Definition at line 45 of file [AddOn.cpp](#).

```
00045 {  
00046     return entity->getHealth();  
00047 }
```

4.1.3.7 getType()

```
Type * AddOn::getType ( ) [virtual]
```

Returns entities type state.

Postconditions:

- Returns the type

Returns

Type* The type state of the entity object

Reimplemented from [Entity](#).

Definition at line 29 of file [AddOn.cpp](#).

```
00029     {  
00030         return entity->getType();  
00031     }
```

4.1.3.8 getValue()

```
int AddOn::getValue ( )
```

Returns the [AddOn](#)'s value attribute.

Postconditions:

- Returns the value attribute of the [AddOn](#) object

Returns

int The values of the [AddOn](#)

Definition at line 17 of file [AddOn.cpp](#).

```
00017     {  
00018         return value;  
00019     }
```

4.1.3.9 setAlliance()

```
void AddOn::setAlliance (  
    Alliance * alliance ) [virtual]
```

Sets the entities alliance.

Preconditions:

- alliance must be an Alliance*

Postconditions:

- Sets the alliance of the entity object

Parameters

<i>alliance</i>	must be a Alliance*
-----------------	---------------------

Returns

void

Reimplemented from [Entity](#).

Definition at line 41 of file [AddOn.cpp](#).

```
00041                                     {  
00042     entity->setAlliance(alliance);  
00043 }
```

4.1.3.10 setDamage()

```
void AddOn::setDamage (  
    int damage ) [virtual]
```

Sets the entities damage.

Preconditions:

- damage must be an int

Postconditions:

- Sets the damage of the entity object

Parameters

<i>damage</i>	must be an int
---------------	----------------

Returns

void

Reimplemented from [Entity](#).

Definition at line 57 of file [AddOn.cpp](#).

```
00057                                     {  
00058     entity->setDamage(damage);  
00059 }
```

4.1.3.11 setEntity()

```
void AddOn::setEntity (
    Entity * entity )
```

Sets the [AddOn](#)'s entity attribute.

Preconditions:

- entity must be an Entity*

Postconditions:

- Sets the entity attribute of the [AddOn](#) object to the passed in entity

Parameters

<i>entity</i>	must be an Entity*
---------------	--------------------

Returns

void

Definition at line 21 of file [AddOn.cpp](#).

```
00021 {
00022     this->entity = entity;
00023 }
```

4.1.3.12 setHealth()

```
void AddOn::setHealth (
    int health ) [virtual]
```

Sets the entities health.

Preconditions:

- health must be an int

Postconditions:

- Sets the health of the entity object

Parameters

<i>health</i>	must be an int
---------------	----------------

Returns

void

Reimplemented from [Entity](#).

Definition at line 49 of file [AddOn.cpp](#).

```
00049 {  
00050     entity->setHealth(health);  
00051 }
```

4.1.3.13 setType()

```
void AddOn::setType (  
    Type * type ) [virtual]
```

Sets the entities type state.

Preconditions:

- type must be an Type*

Postconditions:

- Sets the type state of the entity object

Parameters

<i>type</i>	must be a Type*
-------------	-----------------

Returns

void

Reimplemented from [Entity](#).

Definition at line 33 of file [AddOn.cpp](#).

```
00033 {  
00034     entity->setType(type);  
00035 }
```

4.1.3.14 setValue()

```
void AddOn::setValue (  
    int value )
```

Sets the [AddOn](#)'s value attribute.

Preconditions:

- value must be an int

Postconditions:

- Sets the value attribute of the [AddOn](#) object to the passed in value

Parameters

<i>value</i>	must be an int
--------------	----------------

Returns

void

Definition at line 9 of file [AddOn.cpp](#).

```
00009      {  
00010  
00011          if (value <= 0)  
00012              throw std::invalid_argument("value must be greater than zero");  
00013  
00014          this->value = value;  
00015      }
```

4.1.3.15 takeDamage()

```
virtual void AddOn::takeDamage (  
    int damage ) [pure virtual]
```

Reduces health from the [Entity](#) object.

Preconditions:

- damage must be an int

Postconditions:

- Reduces the health of the [Entity](#) object

Parameters

<i>damage</i>	must be an int
---------------	----------------

Returns

void

Implements [Entity](#).

Implemented in [Armour](#), and [Piercing](#).

4.1.4 Member Data Documentation

4.1.4.1 entity

```
Entity* AddOn::entity [protected]
```

Definition at line 14 of file [AddOn.h](#).

4.1.4.2 value

```
int AddOn::value [protected]
```

Definition at line 13 of file [AddOn.h](#).

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

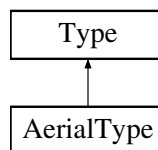
- [AddOn.h](#)
- [AddOn.cpp](#)

4.2 AerialType Class Reference

[AerialType](#) class.

```
#include <AerialType.h>
```

Inheritance diagram for [AerialType](#):



Public Member Functions

- [AerialType](#) ()
Instantiates the ariel type.
- string [getTypeDesc](#) ()
Returns ariel type description.
- [Type](#) * [clone](#) ()
returns the the cloned object of [Type](#)

4.2.1 Detailed Description

[AerialType](#) class.

Used to define [Entity](#) objects as ariel type.

Definition at line 11 of file [AerialType.h](#).

4.2.2 Constructor & Destructor Documentation

4.2.2.1 AerialType()

```
AerialType::AerialType ( )
```

Instantiates the ariel type.

Definition at line 3 of file [AerialType.cpp](#).

```
00003 {}
```

4.2.3 Member Function Documentation

4.2.3.1 clone()

```
Type * AerialType::clone ( ) [virtual]
```

returns the the cloned object of [Type](#)

PostConditions:

- returns Type* type

Returns

Type* The cloned [Type](#) object

Implements [Type](#).

Definition at line 9 of file [AerialType.cpp](#).

```
00009      {  
00010      return new AerialType();  
00011  }
```


4.2.3.2 getTypeDesc()

```
string AerialType::getTypeDesc ( ) [virtual]
```

Returns ariel type description.

Postconditions:

- Returns the ariel type

Returns

string The ariel type string

Implements [Type](#).

Definition at line 5 of file [AerialType.cpp](#).

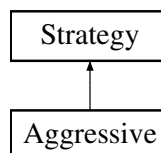
```
00005 {
00006     return "Aerial";
00007 }
```

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

- AerialType.h
- AerialType.cpp

4.3 Aggressive Class Reference

Inheritance diagram for Aggressive:



Public Member Functions

- void [performStrat](#) ([KeyPoint](#) *keyPoint, [Alliance](#) *alliance)
This function will perform an [Aggressive](#) strategy.
- [Strategy](#) * [clone](#) ()
Returns the clone of the Deffensive [Strategy](#) object.

Additional Inherited Members

4.3.1 Detailed Description

Definition at line 5 of file [Aggressive.h](#).

4.3.2 Constructor & Destructor Documentation

4.3.2.1 Aggressive()

Aggressive::Aggressive ()

Definition at line 4 of file [Aggressive.cpp](#).

```
00004 {}
```

4.3.3 Member Function Documentation

4.3.3.1 clone()

[Strategy](#) * Aggressive::clone () [virtual]

Returns the clone of the Defensive [Strategy](#) object.

Returns

Strategy* The clone of the [Defensive Strategy](#) object

Implements [Strategy](#).

Definition at line 11 of file [Aggressive.cpp](#).

```
00011 {  
00012     return new Aggressive();  
00013 }
```

4.3.3.2 performStrat()

```
void Aggressive::performStrat (  
    KeyPoint * keyPoint,  
    Alliance * alliance ) [virtual]
```

This function will perform an [Aggressive](#) strategy.

Author

Antwi-Antwi

Preconditions:

- Takes in object of type [KeyPoint](#) as parameter

Postconditions:

- Returns the [Strategy](#) type

Parameters

<i>keyPoint</i>	an Aggressive strategy will then be performed at this specific keypoint
-----------------	-----------------------------------------------------------------------------------------

Returns

void The function will return a void

Implements [Strategy](#).

Definition at line 6 of file [Aggressive.cpp](#).

```
00006                                     {
00007     int randomNumber = (rand() % 10) + 5;
00008     keyPoint->moveEntitiesInto(alliance, randomNumber);
00009 }
```

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

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

4.4 Alliance Class Reference

Public Member Functions

- [Alliance](#) ()
Instantiates the [Alliance](#).
- [Alliance](#) ([Alliance](#) &alliance)
Instantiates the [Alliance](#).
- [~Alliance](#) ()
Destructor for the [Alliance](#) object.
- void [setNegotiator](#) ([Negotiator](#) *newNegotiator)
Sets the entity negotiator.
- void [addCountry](#) ([Country](#) *nation)
Adds a country into the members vector which holds countries.
- vector< [Entity](#) * > [getReserveEntities](#) (int number)
Return a given number of reserve entites vector.
- void [addReserveEntity](#) ([Entity](#) *entity)
Adds a entity to the reserve entities.
- bool [considerPeace](#) ()
Considers to stop war with the allaince passed into the function header.
- void [addFactory](#) ([Factory](#) *factory)
Adds a factory into the production vector which holds factories.
- void [surrender](#) ()
Makes the current alliance give up of the war by surrendering.
- int [getID](#) ()
Returns [Alliance](#)'s aID.
- bool [offerPeace](#) ()
Offers peace to stop war with the alliance fighting against using sendPeace.
- [Alliance](#) * [clone](#) ()

- Instantiates and returns a clone of the current [Alliance](#).*
- void [setActiveStatus](#) (int active)
Sets variable active to the passed in parameter.
- int [getActive](#) ()
Get the active state of the [Alliance](#).
- int [numRemainingEntities](#) ()
Gets the number of the remaining number of entities.
- void [runFactories](#) ()
Will create reserve Entities.

4.4.1 Detailed Description

Definition at line 13 of file [Alliance.h](#).

4.4.2 Constructor & Destructor Documentation

4.4.2.1 Alliance() [1/2]

```
Alliance::Alliance ( )
```

Instantiates the [Alliance](#).

Definition at line 12 of file [Alliance.cpp](#).

```
00012     {
00013     this->active = 1;
00014     this->aID = totalNum++;
00015     this->negotiator = NULL;
00016     srand(time(0));
00017 }
```

4.4.2.2 Alliance() [2/2]

```
Alliance::Alliance (
    Alliance & alliance )
```

Instantiates the [Alliance](#).

Definition at line 19 of file [Alliance.cpp](#).

```
00019     {
00020     this->active = alliance.active;
00021     this->aID = alliance.aID;
00022
00023     for (int i = 0; i < alliance.members.size(); i++)
00024         this->addCountry(alliance.members[i]->clone());
00025
00026     for (int i = 0; i < alliance.production.size(); i++)
00027         this->addFactory(alliance.production[i]->clone());
00028
00029     for (int i = 0; i < alliance.reserveEntities.size(); i++)
00030         this->addReserveEntity(alliance.reserveEntities[i]->clone());
00031
00032     this->negotiator = NULL;
00033 }
```

4.4.2.3 ~Alliance()

```
Alliance::~~Alliance ( )
```

Destructor for the [Alliance](#) object.

Definition at line 35 of file [Alliance.cpp](#).

```
00035     {
00036
00037     for (int i = 0; i < members.size(); i++)
00038         //delete members[i];
00039
00040     if (this->negotiator != NULL) {
00041         this->negotiator->removeAlliance(this);
00042
00043         if (this->negotiator->getNumAlliances() == 1)
00044             delete this->negotiator;
00045     }
00046 }
```

4.4.3 Member Function Documentation

4.4.3.1 addCountry()

```
void Alliance::addCountry (
    Country * nation )
```

Adds a country into the members vector which holds countries.

Preconditions:

- nation must be an Country*

Postconditions:

- [Country](#) is added to the members vector

Parameters

<i>nation</i>	must be an Country*
---------------	---------------------

Returns

void

Definition at line 52 of file [Alliance.cpp](#).

```
00052     {
00053         members.push_back(nation);
00054 }
```

4.4.3.2 addFactory()

```
void Alliance::addFactory (
    Factory * factory )
```

Adds a factory into the production vector which holds factories.

Preconditions:

- f must be an Factory*

Postconditions:

- Factory is added to the production vector

Parameters

<i>factory</i>	must be a Factory*
----------------	--------------------

Returns

void

Definition at line 78 of file Alliance.cpp.

```
00078 {
00079     production.push_back(factory);
00080 }
```

4.4.3.3 addReserveEntity()

```
void Alliance::addReserveEntity (
    Entity * entity )
```

Adds a entity to the reserve entities.

Preconditions:

- nation must be an Entity*

Postconditions:

- Entity is added to the reserveEntities vector

Parameters

<i>entity</i>	must be an Entity*
---------------	--------------------

Returns

void

Definition at line 66 of file [Alliance.cpp](#).

```
00066 {  
00067     reserveEntities.push_back(entity);  
00068 }
```

4.4.3.4 clone()

```
Alliance * Alliance::clone ( )
```

Instantiates and returns a clone of the current [Alliance](#).

Postconditions:

- Returns the clone of the current [Alliance](#)

Returns

Alliance* The alliance clone

Definition at line 114 of file [Alliance.cpp](#).

```
00114 {  
00115     return new Alliance(*this);  
00116 }
```

4.4.3.5 considerPeace()

```
bool Alliance::considerPeace ( )
```

Considers to stop war with the alliance passed into the function header.

Preconditions:

- id must be an integer

Postconditions:

- Result of consideration returned in the form of a bool

Returns

bool

Definition at line 74 of file [Alliance.cpp](#).

```
00074 {  
00075     return (rand() % 2 == 0);  
00076 }
```

4.4.3.6 getActive()

```
int Alliance::getActive ( )
```

Get the active state of the [Alliance](#).

PostConditions:

- returns an active variable

Returns

int the active variable

Definition at line 110 of file [Alliance.cpp](#).

```
00110 {  
00111     return active;  
00112 }
```

4.4.3.7 getID()

```
int Alliance::getID ( )
```

Returns [Alliance](#)'s aID.

Postconditions:

- Returns the aID

Returns

int The ID of the [Alliance](#) object

Definition at line 95 of file [Alliance.cpp](#).

```
00095 {  
00096     return this->aID;  
00097 }
```

4.4.3.8 getReserveEntities()

```
vector< Entity * > Alliance::getReserveEntities (   
    int number )
```

Return a given number of reserve entites vector.

Precondition:

- number must be an int

Postconditions:

- Return a given number of reserve entities
- If not enough reseerves return amount available

Parameters

<i>number</i>	must be an int
---------------	----------------

Returns

vector<Entity*>*

Definition at line 56 of file [Alliance.cpp](#).

```
00056                                     {
00057     vector<Entity*> out;
00058     for (int i = 0; i < number && i < reserveEntities.size(); i++) {
00059         out.push_back(reserveEntities[i]);
00060         reserveEntities.erase(reserveEntities.begin() + i);
00061     }
00062
00063     return out;
00064 }
```

4.4.3.9 numRemainingEntities()

```
int Alliance::numRemainingEntities ( )
```

Gets the number of the remaining number of entities.

PostConditions:

- Returns an int

Returns

int The number of entities remaining

Definition at line 70 of file [Alliance.cpp](#).

```
00070                                     {
00071     return reserveEntities.size();
00072 }
```

4.4.3.10 offerPeace()

```
bool Alliance::offerPeace ( )
```

Offers peace to stop war with the alliance fighting against using sendPeace.

Postconditions:

- Result of consideration returned from the enemy alliance which considered peace

Returns

bool

Definition at line 99 of file [Alliance.cpp](#).

```
00099                                     {
00100
00101     if (this->negotiator->sendPeace(this)) //Send the peace deal to all the alliances fighting against
00102     {
00103         this->active = 3; //Number 3 means that Alliance chose to peacefully pull out of war
00104         return true;
00105     }
00106
00107     return false;
00108 }
```

4.4.3.11 runFactories()

```
void Alliance::runFactories ( )
```

Will create reserve Entities.

PostConditions

- will create reserve entities for later use

Returns

void

Definition at line 82 of file [Alliance.cpp](#).

```
00082         {
00083     for (int i = 0; i < production.size(); i++) {
00084         RoundStats::numEntitiesCreated++;
00085         reserveEntities.push_back(production[i]->createEntity(this));
00086     }
00087 }
```

4.4.3.12 setActiveStatus()

```
void Alliance::setActiveStatus (
    int active )
```

Sets variable active to the passed in parameter.

PreConditions:

- active must be an a bool

PostConditions:

- The variable active is set to the passed in the parameter

Parameters

<i>ID</i>	a bool parameter
-----------	------------------

4.4.3.13 setNegotiator()

```
void Alliance::setNegotiator (
    Negotiator * newNegotiator )
```

Sets the entity negotiator.

Preconditions:

- n must be an `Negotiator*`

Postconditions:

- Sets the negotiator of the [Alliance](#) object

Parameters

n	must be a <code>Negotiator*</code>
-----	------------------------------------

Returns

void

Definition at line 48 of file [Alliance.cpp](#).

```
00048                                     {
00049     this->negotiator = negotiator;
00050 }
```

4.4.3.14 `surrender()`

```
void Alliance::surrender ( )
```

Makes the current alliance give up of the war by surrendering.

Postconditions:

- Sets the active variable to false
- Removes this alliance from the [Negotiator](#) vector

Returns

void

Definition at line 89 of file [Alliance.cpp](#).

```
00089     {
00090     this->active = 2; //Number 2 means that Alliance has surrendered
00091
00092     this->negotiator->removeAlliance(this);
00093 }
```

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

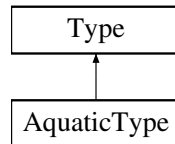
- [Alliance.h](#)
- [Alliance.cpp](#)

4.5 AquaticType Class Reference

[AquaticType](#) class.

```
#include <AquaticType.h>
```

Inheritance diagram for [AquaticType](#):



Public Member Functions

- [AquaticType](#) ()
Instantiates the aquatic type.
- string [getTypeDesc](#) ()
Returns aquatic type description.
- [Type](#) * [clone](#) ()
returns the the cloned object of [Type](#)

4.5.1 Detailed Description

[AquaticType](#) class.

Used to define [Entity](#) objects as aquatic type.

Definition at line 11 of file [AquaticType.h](#).

4.5.2 Constructor & Destructor Documentation

4.5.2.1 [AquaticType](#)()

```
AquaticType::AquaticType ( )
```

Instantiates the aquatic type.

Definition at line 5 of file [AquaticType.cpp](#).

```
00005 {}
```

4.5.3 Member Function Documentation

4.5.3.1 clone()

```
Type * AquaticType::clone ( ) [virtual]
```

returns the the cloned object of [Type](#)

PostConditions:

- returns Type* type

Returns

Type* The cloned [Type](#) object

Implements [Type](#).

Definition at line 11 of file [AquaticType.cpp](#).

```
00011         {  
00012     return new AquaticType();  
00013 }
```

4.5.3.2 getTypeDesc()

```
string AquaticType::getTypeDesc ( ) [virtual]
```

Returns aquatic type description.

Postconditions:

- Returns the aquatic type

Returns

string The aquatic type string

Implements [Type](#).

Definition at line 7 of file [AquaticType.cpp](#).

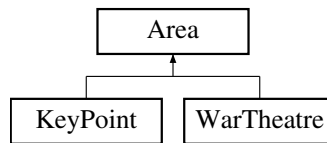
```
00007         {  
00008     return "Aquatic";  
00009 }
```

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

- AquaticType.h
- AquaticType.cpp

4.6 Area Class Reference

Inheritance diagram for Area:



Public Member Functions

- [Area](#) (std::string areaName)
Instantiates the area.
- virtual [~Area](#) ()
Destroys the area object.
- virtual bool [isKeyPoint](#) ()=0
- virtual void [simulateBattle](#) ([Alliance](#) *alliance)=0
- std::string [getAreaName](#) () const
Get the [Area Type](#) object.
- virtual [Area](#) * [clone](#) ()=0
Instantiates and returns a clone of the current war theatre.
- virtual void [addGeneral](#) ([General](#) *general)=0
Adds a general to all the points held by the [WarTheatre](#).

4.6.1 Detailed Description

Definition at line 8 of file [Area.h](#).

4.6.2 Constructor & Destructor Documentation

4.6.2.1 Area()

```
Area::Area (
    std::string areaName )
```

Instantiates the area.

Definition at line 5 of file [Area.cpp](#).

```
00005 {
00006     this->areaName = areaName;
00007 }
```

4.6.2.2 ~Area()

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

Destroys the area object.

Definition at line 9 of file [Area.cpp](#).

```
00009 {}
```

4.6.3 Member Function Documentation

4.6.3.1 addGeneral()

```
virtual void Area::addGeneral (
    General * general ) [pure virtual]
```

Adds a general to all the points held by the [WarTheatre](#).

Preconditions:

- general must be a General*

Postconditions:

- Add general to all points

Parameters

<i>general</i>	must be a General*
----------------	--------------------

Implemented in [KeyPoint](#), and [WarTheatre](#).

4.6.3.2 clone()

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

Instantiates and returns a clone of the current war theatre.

Postconditions:

- Returns the clone of the current war theatre

Returns

WarTheatre* The war theatre clone

Implemented in [KeyPoint](#), and [WarTheatre](#).

4.6.3.3 getAreaName()

```
std::string Area::getAreaName ( ) const
```

Get the [Area Type](#) object.

Returns

std::string reaturns the type

Definition at line 11 of file [Area.cpp](#).

```
00011 {  
00012     return areaName;  
00013 }
```

4.6.3.4 isKeyPoint()

```
virtual bool Area::isKeyPoint ( ) [pure virtual]
```

Implemented in [KeyPoint](#), and [WarTheatre](#).

4.6.3.5 simulateBattle()

```
virtual void Area::simulateBattle (  
    Alliance * alliance ) [pure virtual]
```

Implemented in [KeyPoint](#), and [WarTheatre](#).

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

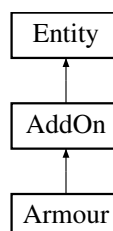
- [Area.h](#)
- [Area.cpp](#)

4.7 Armour Class Reference

[Armour](#) class.

```
#include <Armour.h>
```

Inheritance diagram for [Armour](#):



Public Member Functions

- [Armour](#) (int value)
Instantiates an [Armour](#).
- void [takeDamage](#) (int damage)
Decreases the entities' armour value (or health when their armour has depleted)
- void [dealDamage](#) ([Entity](#) *entity)
Adds to the damage [Entity](#) objects inflict.
- [AddOn](#) * [clone](#) ()
Instantiates and returns a clone of the current [Armour](#).

Additional Inherited Members

4.7.1 Detailed Description

[Armour](#) class.

Used to add protective armour to [Entity](#) objects.

Definition at line 11 of file [Armour.h](#).

4.7.2 Constructor & Destructor Documentation

4.7.2.1 Armour()

```
Armour::Armour (
    int value )
```

Instantiates an [Armour](#).

Parameters

<i>value</i>	must be an int
--------------	----------------

Definition at line 4 of file [Armour.cpp](#).

```
00004 : AddOn(value) {}
```

4.7.3 Member Function Documentation

4.7.3.1 clone()

```
AddOn * Armour::clone ( ) [virtual]
```

Instantiates and returns a clone of the current [Armour](#).

Postconditions:

- Returns the clone of the current [Armour](#)

Returns

Armour* The [Armour](#) clone

Implements [AddOn](#).

Definition at line 22 of file [Armour.cpp](#).

```
00022     {
00023         Armour* armour = new Armour(value);
00024         if (getEntity() != NULL)
00025             armour->setEntity(entity->clone());
00026         return armour;
00027     }
```

4.7.3.2 dealDamage()

```
void Armour::dealDamage (
    Entity * entity ) [virtual]
```

Adds to the damage [Entity](#) objects inflict.

Preconditions:

- entity must be an Entity*

Postconditions:

- Does nothing

Parameters

<i>entity</i>	must be an Entity*
---------------	--------------------

Returns

void

Implements [AddOn](#).

Definition at line 18 of file [Armour.cpp](#).

```
00018     {
00019         this->entity->dealDamage(entity);
00020     }
```

4.7.3.3 takeDamage()

```
void Armour::takeDamage (
    int damage ) [virtual]
```

Decreases the entities' armour value (or health when their armour has depleted)

Preconditions:

- damage must be an int

Postconditions:

- Decreases the entities' armour value (or health when their armour has diminished) by the passed in value

Exceptions:

- damage less 0

Parameters

<i>damage</i>	must be an int and is greater than 0
---------------	--------------------------------------

Returns

void

Implements [AddOn](#).

Definition at line 6 of file [Armour.cpp](#).

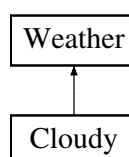
```
00006 {
00007
00008     if (damage <= 0)
00009         throw std::invalid_argument("damage must be greater than zero");
00010
00011     if (value > 0) {
00012         value -= damage;
00013     } else {
00014         entity->takeDamage(damage);
00015     }
00016 }
```

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

- Armour.h
- Armour.cpp

4.8 Cloudy Class Reference

Inheritance diagram for Cloudy:



Public Member Functions

- [Cloudy](#) ()
Instantiates the [Cloudy](#) object of the state pattern.
- `std::string` [getWeather](#) ()
Returns string which tells us the weather.
- `void` [handleChange](#) ([KeyPoint](#) *keypoint)
Will change the current state of the weather inside the specific keypoint.
- [Weather](#) * [clone](#) ()
Returns a clone of the [Cloudy](#) object.

Additional Inherited Members

4.8.1 Detailed Description

Definition at line 6 of file [Cloudy.h](#).

4.8.2 Constructor & Destructor Documentation

4.8.2.1 Cloudy()

```
Cloudy::Cloudy ( )
```

Instantiates the [Cloudy](#) object of the state pattern.

Definition at line 4 of file [Cloudy.cpp](#).

```
00004         : Weather() {  
00005     this->multiplier = 0.75;  
00006 }
```

4.8.3 Member Function Documentation

4.8.3.1 clone()

```
Weather * Cloudy::clone ( ) [virtual]
```

Returns a clone of the [Cloudy](#) object.

Returns

`Weather*` Clone of cloudy object

Implements [Weather](#).

Definition at line 17 of file [Cloudy.cpp](#).

```
00017     {  
00018     return new Cloudy();  
00019 }
```

4.8.3.2 getWeather()

```
std::string Cloudy::getWeather ( ) [virtual]
```

Returns string which tells us the weather.

Postconditions:

- Returns the wether of ths current state

Returns

std::string which is the current state

Implements [Weather](#).

Definition at line 8 of file [Cloudy.cpp](#).

```
00008 {  
00009     return "Cloudy";  
00010 }
```

4.8.3.3 handleChange()

```
void Cloudy::handleChange (   
    KeyPoint * keypoint ) [virtual]
```

Will change the current state of the weather inside the specific keypoint.

Preconditions:

- k must be a KeyPoint*

Postconditions:

- Changes the current weather to the next one in the state pattern ([Rainy](#))

Parameters

<i>k</i>	must be a KeyPoint*
----------	---------------------

Returns

void

Implements [Weather](#).

Definition at line 12 of file [Cloudy.cpp](#).

```
00012                                     {
00013     Rainy* newWeather = new Rainy();
00014     k->setWeather(newWeather);
00015 }
```

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

- Cloudy.h
- Cloudy.cpp

4.9 Country Class Reference

Public Member Functions

- [Country](#) (std::string name)
Instantiates the [Country](#).
- [Country](#) * [clone](#) ()
Instantiates and returns a clone of the current [Country](#).
- void [setName](#) (std::string name)
Set the name of the country.
- void [setID](#) (int id)
Set the id of the country.
- std::string [getName](#) () const
Get the name of the country.
- int [getID](#) () const
Get the id of the country.

4.9.1 Detailed Description

Definition at line 5 of file [Country.h](#).

4.9.2 Constructor & Destructor Documentation

4.9.2.1 Country()

```
Country::Country (
    std::string name )
```

Instantiates the [Country](#).

Definition at line 5 of file [Country.cpp](#).

```
00005     {
00006         this->name = name;
00007         this->id = rand() % 1000;
00008     }
```

4.9.3 Member Function Documentation

4.9.3.1 clone()

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

Instantiates and returns a clone of the current [Country](#).

Postconditions:

- Returns the clone of the current [Country](#)

Returns

Country* The country clone

Definition at line 11 of file [Country.cpp](#).

```
00011         {  
00012     return new Country(this->name);  
00013 }
```

4.9.3.2 getID()

```
int Country::getID ( ) const
```

Get the id of the country.

PostConditions:

- return the id the id of the country

Returns

int

Definition at line 19 of file [Country.cpp](#).

```
00019     {  
00020     return this->id;  
00021 }
```

4.9.3.3 getName()

```
string Country::getName ( ) const
```

Get the name of the country.

PostConditions:

- Return the name of the country

Returns

string

Definition at line 15 of file [Country.cpp](#).

```
00015         {  
00016     return this->name;  
00017 }
```

4.9.3.4 setID()

```
void Country::setID (  
    int id )
```

Set the if of the country.

Precondition:

- The variale if is type of int

Preconditions:

- The variable id is set the the passed in parameter

Parameters

<i>id</i>	
-----------	--

4.9.3.5 setName()

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

Set the name of the country.

Precondition:

- The variable name is type of string

Preconditions:

- The variable name is set the the passed in parameter

Parameters

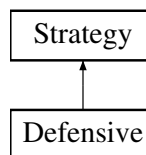
<i>name</i>	
-------------	--

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

- Country.h
- Country.cpp

4.10 Defensive Class Reference

Inheritance diagram for Defensive:



Public Member Functions

- void [performStrat](#) ([KeyPoint](#) *keyPoint, [Alliance](#) *alliance)
This function will perform an [Defensive](#) strategy.
- [Strategy](#) * [clone](#) ()
Returns the clone of the Defensive [Strategy](#) object.

Additional Inherited Members

4.10.1 Detailed Description

Definition at line 7 of file [Defensive.h](#).

4.10.2 Constructor & Destructor Documentation

4.10.2.1 Defensive()

```
Defensive::Defensive ( )
```

Definition at line 3 of file [Defensive.cpp](#).

```
00003         {
00004
00005 }
```

4.10.3 Member Function Documentation

4.10.3.1 clone()

```
Strategy * Defensive::clone ( ) [virtual]
```

Returns the clone of the Deffensive [Strategy](#) object.

Returns

[Strategy](#)* The clone of the [Defensive Strategy](#) object

Implements [Strategy](#).

Definition at line 13 of file [Defensive.cpp](#).

```
00013         {
00014             return new Defensive();
00015 }
```

4.10.3.2 performStrat()

```
void Defensive::performStrat (
    KeyPoint * keyPoint,
    Alliance * alliance ) [virtual]
```

This function will perform an [Defensive](#) strategy.

Author

Antwi-Antwi

Parameters

<i>keyPoint</i>	an Defensive strategy will then be performed at this specific keypoint
-----------------	----------------------------------------------------------------------------------------

Returns

void The function will return a void

Implements [Strategy](#).

Definition at line 7 of file [Defensive.cpp](#).

```
00007                                     {
00008
00009     int randomNumber = (rand() % 5) + 1;
00010     keyPoint->moveEntitiesInto(alliance, randomNumber);
00011 }
```

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

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

4.11 EasySetup Class Reference

Public Member Functions

- void [setupSimulation](#) ()
- void [runSimulation](#) ()
- void [loadPrevSave](#) ()
- void [loadSpecificSave](#) (std::string name)
- void [saveSimulationSetup](#) ()

4.11.1 Detailed Description

Definition at line 11 of file [EasySetup.h](#).

4.11.2 Constructor & Destructor Documentation

4.11.2.1 EasySetup()

```
EasySetup::EasySetup ( )
```

Definition at line 21 of file [EasySetup.cpp](#).

```
00021     {
00022     saveArchive = new SaveArchive();
00023 }
```

4.11.3 Member Function Documentation

4.11.3.1 loadPrevSave()

void EasySetup::loadPrevSave ()

Definition at line 284 of file [EasySetup.cpp](#).

```
00284         {
00285
00286     try{
00287         WarEngineMemento* saveFile = saveArchive->getLastSave();
00288
00289         WarEngine::getInstance().loadSave(saveFile);
00290     }
00291     catch(const std::exception& error){
00292
00293         std::cerr << error.what() << "\n";
00294
00295     }
00296 }
```

4.11.3.2 loadSpecificSave()

void EasySetup::loadSpecificSave (
std::string name)

Definition at line 298 of file [EasySetup.cpp](#).

```
00298         {
00299
00300     try{
00301
00302         WarEngineMemento* saveFile = saveArchive->getSave(name);
00303
00304         WarEngine::getInstance().loadSave(saveFile);
00305     }
00306     catch(const std::out_of_range& range_error){
00307
00308         std::cerr << range_error.what() << "\n";
00309
00310     }
00311 }
```

4.11.3.3 runSimulation()

void EasySetup::runSimulation ()

Definition at line 267 of file [EasySetup.cpp](#).

```
00267         {
00268
00269         WarEngine::getInstance().simulate();
00270 }
```

4.11.3.4 saveSimulationSetup()

void EasySetup::saveSimulationSetup ()

Definition at line 272 of file [EasySetup.cpp](#).

```
00272         {
00273
00274         // Getting the name of the save
00275         cout << "Please enter name of save: ";
00276         string saveName;
00277         getline(cin, saveName);
00278
00279         // saving the current state of the simulation
00280         saveArchive->addNewSave(saveName, WarEngine::getInstance().saveState());
00281
00282 }
```

4.11.3.5 setupSimulation()

void EasySetup::setupSimulation ()

Definition at line 25 of file [EasySetup.cpp](#).

```

00025     {
00026         while (true)
00027         {
00028             cout << "Load simulation (L) or New Simulation (N): ";
00029             string selectedOption;
00030             cin >> selectedOption;
00031             cin.ignore();
00032
00033             if(toupper(selectedOption[0]) == 'L')
00034             {
00035                 string saveName;
00036                 cout << "Please enter the name of the save to be re-simulated" << endl;
00037                 getline(cin, saveName); // getting the name of the save-archive
00038                 try {
00039                     this->loadSpecificSave(saveName); // loading the save-archive
00040                     return; // will return if the above the function does not throw an exception
00041                 } catch(const std::exception& exception) {
00042                     cout << "Error: " << exception.what() << endl;
00043
00044                     if (strcmp(exception.what(), "Save archive is empty") == 0) {
00045                         cout << "Please create new simulation" << endl;
00046                         goto setup;
00047                     } else if (strcmp(exception.what(), "No save with given name exists") == 0) {
00048
00049                         cout << "Please enter the correct name of save-archive and try again or create new
00050 simulation" << endl;
00051                     }
00052                 }
00053             } else if(toupper(selectedOption[0]) == 'N') {
00054                 // setting up a new simulation
00055                 goto setup;
00056             } else {
00057                 cout << "Incorrect input: Please enter (L) or (N)" << endl;
00058             }
00059         }
00060     }
00061
00062     setup:
00063     // Creating alliances and generals
00064     int numAlliesAndGenerals;
00065     cout << "Enter number of alliances: ";
00066     cin >> numAlliesAndGenerals;
00067
00068     Alliance** alliances = new Alliance*[numAlliesAndGenerals];
00069     General** generals = new General*[numAlliesAndGenerals];
00070
00071     int numCountries,
00072         numFactories;
00073     string countryName,
00074         factoryType,
00075         selectedFactory,
00076         selectedAddOn;
00077     Country* country;
00078     Type* type;
00079     AddOn* addOn;
00080     Factory* factory;
00081
00082     Negotiator* negotiator = new Negotiator();
00083
00084     for (int i = 0; i < numAlliesAndGenerals; i++) {
00085         alliances[i] = new Alliance();
00086         negotiator->addAlliance(alliances[i]);
00087         alliances[i]->setNegotiator(negotiator);
00088         WarEngine::getInstance().addAlliance(alliances[i]);
00089
00090         cout << "Enter number of countries for Alliance " << alliances[i]->getID() << ": ";
00091         cin >> numCountries;
00092         cin.ignore();
00093
00094         for (int k = 0; k < numCountries; k++) {
00095             cout << "Enter name of county " << k+1 << ": ";
00096             getline(cin, countryName);
00097             country = new Country(countryName);
00098             alliances[i]->addCountry(country);
00099         }
00100
00101         cout << "Enter number of factories for Alliance " << alliances[i]->getID() << ": ";
00102         cin >> numFactories;
00103     }

```

```

00104         for (int k = 0; k < numFactories; k++) {
00105             retryType:
00106                 cout << "Factory " << k+1 << " is of type Aquatic(Q), Aerial(E), or Terrain(T) : ";
00107                 cin >> factoryType;
00108                 cin.ignore();
00109
00110                 if (toupper(factoryType[0]) == 'Q') {
00111                     type = new AerialType;
00112                 } else if (toupper(factoryType[0]) == 'E') {
00113                     type = new AerialType;
00114                 } else if (toupper(factoryType[0]) == 'T') {
00115                     type = new TerrainType;
00116                 } else {
00117                     cout << "Invalid type input! Try again" << endl;
00118                     goto retryType;
00119                 }
00120
00121             retryAddOn:
00122                 cout << "Select AddOn for factory " << k+1 << " Armour(A), Piercing(P) or None(N) : ";
00123                 getline(cin, selectedAddOn);
00124                 if (toupper(selectedAddOn[0]) == 'A') {
00125                     int value;
00126                     cout << "Enter armour value: ";
00127                     cin >> value;
00128                     cin.ignore();
00129                     addOn = new Armour(value);
00130                 } else if (toupper(selectedAddOn[0]) == 'P') {
00131                     int value;
00132                     cout << "Enter piercing value: ";
00133                     cin >> value;
00134                     cin.ignore();
00135                     addOn = new Piercing(value);
00136                 } else if (toupper(selectedAddOn[0]) == 'N') {
00137                     addOn = NULL;
00138                 } else {
00139                     cout << "Invalid AddOn input! Try again" << endl;
00140                     goto retryAddOn;
00141                 }
00142
00143             retryFactory:
00144                 cout << "Which factory is factory " << k+1 << " Vehicle(V), Personnel(P), or Support (S) : ";
00145
00146                 getline(cin, selectedFactory);
00147                 if (toupper(selectedFactory[0]) == 'V') {
00148                     factory = new VehicleFactory(type, addOn);
00149                 } else if (toupper(selectedFactory[0]) == 'P') {
00150                     factory = new PersonnelFactory(type, addOn);
00151                 } else if (toupper(selectedFactory[0]) == 'S') {
00152                     factory = new SupportFactory(type, addOn);
00153                 } else {
00154                     cout << "Invalid factory input! Try again" << endl;
00155                     goto retryFactory;
00156                 }
00157                 alliances[i]->addFactory(factory);
00158             }
00159
00160             string selectedStrat;
00161             Strategy* strat;
00162
00163             retryStrat:
00164             Aggressive(A) : ";
00165                 cout << "What is this Alliances generals strategy Passive(P), Defensive(D), or
00166                 Aggressive(A) : ";
00167                 getline(cin, selectedStrat);
00168                 if (toupper(selectedStrat[0]) == 'P') {
00169                     strat = new Passive();
00170                 } else if (toupper(selectedStrat[0]) == 'D') {
00171                     strat = new Defensive();
00172                 } else if (toupper(selectedStrat[0]) == 'A') {
00173                     strat = new Aggressive();
00174                 } else {
00175                     cout << "Invalid strategy input! Try again" << endl;
00176                     goto retryStrat;
00177                 }
00178                 generals[i] = new General(alliances[i], strat);
00179             }
00180
00181             int factoryRun;
00182             cout << "How many production runs do you wish to perform: ";
00183             cin >> factoryRun;
00184             cin.ignore();
00185             for (int i = 0; i < numAlliesAndGenerals; i++) {
00186                 for (int j = 0; j < factoryRun; j++) {
00187                     alliances[i]->runFactories();
00188                 }
00189             }

```

```

00189
00190 // Creating main WarTheatre
00191 WarTheatre* mainBattleGround;
00192 cout << "Creating the main battle ground" << endl;
00193 string battleGroundName;
00194 cout << "Set main battle ground's name: ";
00195 getline(cin, battleGroundName);
00196 mainBattleGround = new WarTheatre(battleGroundName);
00197
00198 int sizeOfGrounds;
00199 cout << "Enter number of battle grounds in " << battleGroundName << " battle ground: ";
00200 cin >> sizeOfGrounds;
00201 cin.ignore();
00202 WarTheatre** battleGrounds = new WarTheatre*[sizeOfGrounds];
00203
00204 // Creating sub WarTheatres
00205 for (int i = 0; i < sizeOfGrounds; i++) {
00206     battleGroundName.clear();
00207     cout << "Set battle ground " << i+1 << "'s name: ";
00208     getline(cin, battleGroundName);
00209     battleGrounds[i] = new WarTheatre(battleGroundName);
00210 }
00211
00212 vector<int> numKeyPoints;
00213 int numKeyPoint = 0;
00214
00215 for (int i = 0; i < sizeOfGrounds; i++) {
00216     cout << "Enter number of key points in " << battleGrounds[i]->getAreaName() << " battle
ground: ";
00217     cin >> numKeyPoint;
00218     cin.ignore();
00219     numKeyPoints.push_back(numKeyPoint);
00220     numKeyPoint = 0;
00221 }
00222
00223 KeyPoint* keyPoint;
00224 string keyPointName;
00225 int numEntitiesInKeyPt;
00226
00227 // Creating KeyPoints for the sub WarTheatres
00228 for (int i = 0; i < sizeOfGrounds; i++) {
00229     numKeyPoint = numKeyPoints[i];
00230     cout << "For " << battleGrounds[i]->getAreaName() << "'s key points" << endl;
00231
00232     for (int k = 0; k < numKeyPoint; k++) {
00233         cout << "Set key point " << i+1 << "'s name: ";
00234         getline(cin, keyPointName);
00235         keyPoint = new KeyPoint(keyPointName);
00236
00237         for (int a = 0; a < numAlliesAndGenerals; a++) {
00238             tryAgain:
00239             cout << "There are " << alliances[a]->numRemainingEntities() << " for Alliance " <<
a+1 << endl;
00240             cout << "How many would you like to place in " << keyPointName << " keypoint? ";
00241             cin >> numEntitiesInKeyPt;
00242             cin.ignore();
00243
00244             if (alliances[a]->numRemainingEntities() > 0 &&
alliances[a]->numRemainingEntities() < numEntitiesInKeyPt) {
00245                 cout << "You selected more than the available amount. Try again " << endl;
00246                 goto tryAgain;
00247             } else if (alliances[a]->numRemainingEntities() <= 0) {
00248                 continue;
00249             } else {
00250                 keyPoint->moveEntitiesInto(alliances[a], numEntitiesInKeyPt);
00251             }
00252         }
00253
00254         battleGrounds[i]->addArea(keyPoint);
00255     }
00256
00257     mainBattleGround->addArea(battleGrounds[i]);
00258 }
00259
00260 for (int i = 0; i < numAlliesAndGenerals; i++) {
00261     mainBattleGround->addGeneral(generals[i]);
00262 }
00263
00264 WarEngine::getInstance().setWarTheatre(mainBattleGround);
00265 }

```

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

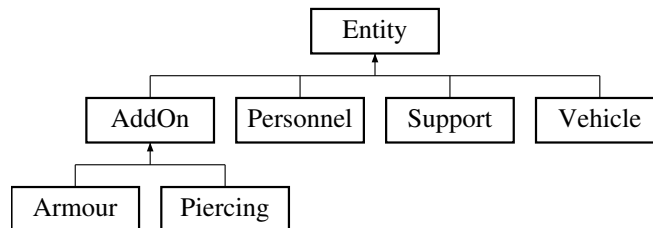
- EasySetup.h
- EasySetup.cpp

4.12 Entity Class Reference

[Entity](#) class.

```
#include <Entity.h>
```

Inheritance diagram for Entity:



Public Member Functions

- [Entity](#) ([Type](#) *type, int health, int damage)
Instantiates the entity.
- virtual [Type](#) * [getType](#) ()
Returns entities type state.
- virtual void [setType](#) ([Type](#) *type)
Sets the entities type state.
- virtual [Alliance](#) * [getAlliance](#) ()
Returns entities alliance.
- virtual void [setAlliance](#) ([Alliance](#) *alliance)
Sets the entities alliance.
- virtual int [getHealth](#) ()
Returns entities health.
- virtual void [setHealth](#) (int health)
Sets the entities health.
- virtual int [getDamage](#) ()
Returns entities damage.
- virtual void [setDamage](#) (int damage)
Sets the entities damage.
- virtual void [takeDamage](#) (int damage)=0
Reduces health from the [Personnel](#) object.
- virtual void [dealDamage](#) ([Entity](#) *entity)=0
Inflicts damage onto another entity.
- virtual [Entity](#) * [clone](#) ()=0
Clones the current [Entity](#) object and returns the cloned object.

4.12.1 Detailed Description

[Entity](#) class.

Used to simulate war entity objects.

Definition at line 13 of file [Entity.h](#).

4.12.2 Constructor & Destructor Documentation

4.12.2.1 Entity() [1/2]

```
Entity::Entity ( )
```

Definition at line 5 of file [Entity.cpp](#).

```
00005      {
00006      health = 0;
00007      damage = 0;
00008      type = NULL;
00009  }
```

4.12.2.2 Entity() [2/2]

```
Entity::Entity (
    Type * type,
    int health,
    int damage )
```

Instantiates the entity.

Parameters

<i>type</i>	must be a Type*
-------------	-----------------

Definition at line 11 of file [Entity.cpp](#).

```
00011      {
00012      this->health = health;
00013      this->damage = damage;
00014      this->type = type;
00015  }
```

4.12.3 Member Function Documentation

4.12.3.1 clone()

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

Clones the current [Entity](#) object and returns the cloned object.

PostConditions:

- Returns the cloned object of [Entity](#)

Returns

Entity* The cloned object

Implemented in [Armour](#), [Personnel](#), [Piercing](#), [Support](#), [Vehicle](#), and [AddOn](#).

4.12.3.2 dealDamage()

```
virtual void Entity::dealDamage (
    Entity * entity ) [pure virtual]
```

Inflicts damage onto another entity.

Preconditions:

- entity must be an Entity*

Postconditions:

- Reduces the health of the entity

Parameters

<i>entity</i>	must be an Entity*
---------------	--------------------

Returns

void

Implemented in [Armour](#), [Personnel](#), [Piercing](#), [Support](#), [Vehicle](#), and [AddOn](#).

4.12.3.3 getAlliance()

```
Alliance * Entity::getAlliance ( ) [virtual]
```

Returns entities alliance.

Postconditions:

- Returns the alliance

Returns

Type* The alliance of the entity object

Reimplemented in [AddOn](#).

Definition at line 25 of file [Entity.cpp](#).

```
00025 {
00026     return this->alliance;
00027 }
```

4.12.3.4 getDamage()

```
int Entity::getDamage ( ) [virtual]
```

Returns entities damage.

Postconditions:

- Returns the damage

Returns

int The damage of the entity object

Reimplemented in [AddOn](#).

Definition at line 41 of file [Entity.cpp](#).

```
00041     {  
00042         return this->damage;  
00043     }
```

4.12.3.5 getHealth()

```
int Entity::getHealth ( ) [virtual]
```

Returns entities health.

Postconditions:

- Returns the health

Returns

int The health of the entity object

Reimplemented in [AddOn](#).

Definition at line 33 of file [Entity.cpp](#).

```
00033     {  
00034         return this->health;  
00035     }
```

4.12.3.6 getType()

```
Type * Entity::getType ( ) [virtual]
```

Returns entities type state.

Postconditions:

- Returns the type

Returns

Type* The type state of the entity object

Reimplemented in [AddOn](#).

Definition at line 17 of file [Entity.cpp](#).

```
00017 {  
00018     return this->type;  
00019 }
```

4.12.3.7 setAlliance()

```
void Entity::setAlliance (  
    Alliance * alliance ) [virtual]
```

Sets the entities alliance.

Preconditions:

- alliance must be an Alliance*

Postconditions:

- Sets the alliance of the entity object

Parameters

<i>alliance</i>	must be a Alliance*
-----------------	---------------------

Returns

void

Reimplemented in [AddOn](#).

Definition at line 29 of file [Entity.cpp](#).

```
00029                                     {
00030     this->alliance = alliance;
00031 }
```

4.12.3.8 setDamage()

```
void Entity::setDamage (
    int damage ) [virtual]
```

Sets the entities damage.

Preconditions:

- damage must be an int

Postconditions:

- Sets the damage of the entity object

Parameters

<i>damage</i>	must be an int
---------------	----------------

Returns

void

Reimplemented in [AddOn](#).

Definition at line 45 of file [Entity.cpp](#).

```
00045                                     {
00046     this->damage = damage;
00047 }
```

4.12.3.9 setHealth()

```
void Entity::setHealth (
    int health ) [virtual]
```

Sets the entities health.

Preconditions:

- health must be an int

Postconditions:

- Sets the health of the entity object

Parameters

<i>health</i>	must be an int
---------------	----------------

Returns

void

Reimplemented in [AddOn](#).Definition at line 37 of file [Entity.cpp](#).

```
00037 {  
00038     this->health = health;  
00039 }
```

4.12.3.10 setType()

```
void Entity::setType (  
    Type * type ) [virtual]
```

Sets the entities type state.

Preconditions:

- type must be an Type*

Postconditions:

- Sets the type state of the entity object

Parameters

<i>type</i>	must be a Type*
-------------	-----------------

Returns

void

Reimplemented in [AddOn](#).Definition at line 21 of file [Entity.cpp](#).

```
00021 {  
00022     this->type = type;  
00023 }
```

4.12.3.11 takeDamage()

```
virtual void Entity::takeDamage (  
    int damage ) [pure virtual]
```

Reduces health from the [Personnel](#) object.

Preconditions:

- damage must be an int

Postconditions:

- Reduces the health of the [Entity](#) object

Parameters

<i>damage</i>	must be an int
---------------	----------------

Returns

void

Reduces health from the [Personnel](#) object

Preconditions:

- damage must be an int

Postconditions:

- Reduces the health of the [Entity](#) object

Parameters

<i>damage</i>	must be an int
---------------	----------------

Returns

void

Implemented in [Armour](#), [Personnel](#), [Piercing](#), [Support](#), [Vehicle](#), and [AddOn](#).

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

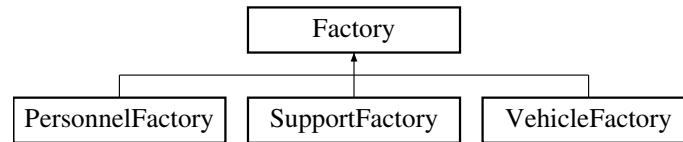
- Entity.h
- Entity.cpp

4.13 Factory Class Reference

[Factory](#) class.

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

Inheritance diagram for [Factory](#):



Public Member Functions

- [Factory](#) ([Type](#) *type, [AddOn](#) *addOn)
Instantiates the factory.
- [~Factory](#) ()
Destroys the factory object.
- virtual [Entity](#) * [createEntity](#) ([Alliance](#) *alliance)=0
- [Type](#) * [getType](#) ()
Returns factories type state.
- void [setType](#) ([Type](#) *type)
Sets the factories type state.
- [AddOn](#) * [getAddOn](#) ()
Returns factories add ons.
- void [setAddOns](#) ([AddOn](#) *addOn)
Sets the factories add ons.
- virtual [Factory](#) * [clone](#) ()=0
the factoru object will be cloned and returned

4.13.1 Detailed Description

[Factory](#) class.

Used to instantiate [Entity](#) objects.

Definition at line 12 of file [Factory.h](#).

4.13.2 Constructor & Destructor Documentation

4.13.2.1 [Factory](#)()

```

Factory::Factory (
    Type * type,
    AddOn * addOn )
  
```

Instantiates the factory.

Parameters

<i>type</i>	must be a Type*
<i>addOn</i>	must be a AddOn*

Definition at line 3 of file [Factory.cpp](#).

```
00003                                     {
00004     this->type = type;
00005     this->addOn = addOn;
00006 }
```

4.13.2.2 ~Factory()

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

Destroys the factory object.

Postconditions:

- All dynamic memory should be deallocated from the factory object

Definition at line 8 of file [Factory.cpp](#).

```
00008     {
00009         delete type;
00010         delete addOn;
00011 }
```

4.13.3 Member Function Documentation**4.13.3.1 clone()**

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

the factoru object will be cloned and returned

PostConditions:

- returns the cloned object of type Factory*

Returns

Factory* the cloned object

Implemented in [PersonnelFactory](#), [SupportFactory](#), and [VehicleFactory](#).

4.13.3.2 createEntity()

```
virtual Entity * Factory::createEntity (
    Alliance * alliance ) [pure virtual]
```

Implemented in [PersonnelFactory](#), [SupportFactory](#), and [VehicleFactory](#).

4.13.3.3 getAddOn()

```
AddOn * Factory::getAddOn ( )
```

Returns factories add ons.

Postconditions:

- Returns the add ons of the factory

Returns

AddOn* The decorators for the factory object

Definition at line 21 of file [Factory.cpp](#).

```
00021     {
00022     return this->addOn;
00023 }
```

4.13.3.4 getType()

```
Type * Factory::getType ( )
```

Returns factories type state.

Postconditions:

- Returns the type

Returns

Type* The type state of the factory object

Definition at line 13 of file [Factory.cpp](#).

```
00013     {
00014     return this->type;
00015 }
```

4.13.3.5 setAddOns()

```
void Factory::setAddOns (
    AddOn * addOn )
```

Sets the factories add ons.

Preconditions:

- addOns must be an AddOn*

Postconditions:

- Sets the add ons of the factory object

Parameters

<i>addOn</i>	must be a AddOn*
--------------	------------------

Returns

void

Definition at line 25 of file [Factory.cpp](#).

```
00025 {  
00026     this->addOn = addOn;  
00027 }
```

4.13.3.6 setType()

```
void Factory::setType (  
    Type * type )
```

Sets the factories type state.

Preconditions:

- type must be an Type*

Postconditions:

- Sets the type state of the factory object

Parameters

<i>type</i>	must be a Type*
-------------	-----------------

Returns

void

Definition at line 17 of file [Factory.cpp](#).

```
00017 {  
00018     this->type = type;  
00019 }
```

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

- Factory.h
- Factory.cpp

4.14 General Class Reference

Public Member Functions

- [General](#) ([Alliance](#) *alliance, [Strategy](#) *strategy)
Construct a new [General](#) object.
- void [initiateStrategy](#) ([KeyPoint](#) *keyPoint)
The function initiates the strategy.
- bool [setStrategy](#) ([Strategy](#) *strategy)
Set the [Strategy](#) object.
- [Alliance](#) * [getAlliance](#) ()
Returns the [Alliance](#) object.

4.14.1 Detailed Description

Definition at line 8 of file [General.h](#).

4.14.2 Constructor & Destructor Documentation

4.14.2.1 General()

```
General::General (
    Alliance * alliance,
    Strategy * strategy )
```

Construct a new [General](#) object.

Parameters

<i>alliance</i>	must be an Alliance*
<i>strategy</i>	must be a Strategy*

Definition at line 3 of file [General.cpp](#).

```
00003                                     {
00004     this->alliance = alliance;
00005     this->strategy = strategy;
00006     numDeaths = 0;
00007 }
```

4.14.3 Member Function Documentation

4.14.3.1 getAlliance()

```
Alliance * General::getAlliance ( )
```

Returns the [Alliance](#) object.

PostConditions:

- Returns the alliance of the general

Returns

Alliance* The alliance that the general is associated

Definition at line 22 of file [General.cpp](#).

```
00022     {
00023         return this->alliance;
00024     }
```

4.14.3.2 initiateStrategy()

```
void General::initiateStrategy (
    KeyPoint * keyPoint )
```

The function initiates the strategy.

Precondition:

- keyPoint must be a KeyPoint*

Parameters

<i>keyPoint</i>	must be a KeyPoint*
-----------------	---------------------

Returns

void

Definition at line 9 of file [General.cpp](#).

```
00009     {
00010         numDeaths++;
00011         if (numDeaths >= 5) {
00012             strategy->performStrat(keyPoint, this->alliance);
00013             numDeaths = 0;
00014         }
00015     }
```

4.14.3.3 setStrategy()

```
bool General::setStrategy (
    Strategy * strategy )
```

Set the [Strategy](#) object.

PreConditons:

- strategy must be of type [Strategy](#)*

PostConditions:

- true is returned if setting the strategy was successful
- false is returned if setting the strategy was unsuccessful

Parameters

<i>strategy</i>	
-----------------	--

Returns

true if the setting the [Strategy](#) object was successful
false if the setting the [Strategy](#) object was unsuccessful

Definition at line 17 of file [General.cpp](#).

```
00017                                     {
00018     this->strategy = strategy;
00019     return true;
00020 }
```

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

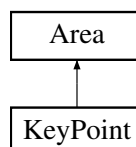
- General.h
- General.cpp

4.15 KeyPoint Class Reference

Keypoint class.

```
#include <KeyPoint.h>
```

Inheritance diagram for KeyPoint:



Public Member Functions

- [KeyPoint](#) (std::string areaName)
Instantiates the key point.
- [KeyPoint](#) ([KeyPoint](#) &keyPoint)
Instantiates a copy of a [KeyPoint](#).
- bool [isKeyPoint](#) ()
Returns area type.
- void [simulateBattle](#) ([Alliance](#) *alliance)
Simulate Battle with troops from the alliance passed in.
- void [clearBattlefield](#) ([Alliance](#) *alliance)
Clears the battlefield of all deceased troops.
- void [moveEntitiesInto](#) ([Alliance](#) *alliance, int numTroops)
Moves a specific alliances troops into this keypoint.
- void [moveEntitiesOutOf](#) ([Alliance](#) *alliance, int numTroops)
Moves a specific alliances troops out of the keypoint.
- void [addEntity](#) ([Entity](#) *entity)
Adds an entity to the key point object.
- void [addGeneral](#) ([General](#) *general)
Adds a general to all the points held by the [WarTheatre](#).
- void [removeGeneral](#) ([General](#) *general)
removes a general to all the points held by the [WarTheatre](#)
- [Area](#) * [clone](#) ()
Instantiates and returns a clone of the current Keypoint.
- void [changeWeather](#) ()
Switches the [Weather](#) object to the next state.
- void [setWeather](#) ([Weather](#) *weather)
Set the [Weather](#) object.
- std::string [getWeather](#) () const
The weather at the current state is returned.

4.15.1 Detailed Description

Keypoint class.

Used to emulate strategic positions.

Definition at line 17 of file [KeyPoint.h](#).

4.15.2 Constructor & Destructor Documentation

4.15.2.1 KeyPoint() [1/2]

```
KeyPoint::KeyPoint (
    std::string areaName )
```

Instantiates the key point.

Parameters

<i>areaName</i>	must be an string
-----------------	-------------------

4.15.2.2 KeyPoint() [2/2]

```
KeyPoint::KeyPoint (
    KeyPoint & keyPoint )
```

Instantiates a copy of a [KeyPoint](#).

Parameters

<i>keyPoint</i>	must be an KeyPoint instance
-----------------	----------------------------------------------

Definition at line 15 of file [KeyPoint.cpp](#).

```
00015                                     : Area(keyPoint.getAreaName()) {
00016     for (int i = 0; i < keyPoint.entities.size(); i++)
00017         this->addEntity(keyPoint.entities[i]->clone());
00018
00019     weather = keyPoint.weather->clone();
00020 }
```

4.15.2.3 ~KeyPoint()

```
KeyPoint::~~KeyPoint ( )
```

Definition at line 22 of file [KeyPoint.cpp](#).

```
00022     {
00023     for (int i = 0; i < entities.size(); i++)
00024         delete entities[i];
00025
00026     for (int i = 0; i < generals.size(); i++)
00027         delete generals[i];
00028
00029     delete weather;
00030 }
```

4.15.3 Member Function Documentation**4.15.3.1 addEntity()**

```
void KeyPoint::addEntity (
    Entity * entity )
```

Adds an entity to the key point object.

Preconditions:

- entity must be an Entity*

Postconditions:

- Add entity to key point

Parameters

<i>entity</i>	must be an Entity*
---------------	--------------------

Returns

void

Definition at line 123 of file [KeyPoint.cpp](#).

```
00123                                     {  
00124     entities.push_back(entity);  
00125 }
```

4.15.3.2 addGeneral()

```
void KeyPoint::addGeneral (  
    General * general ) [virtual]
```

Adds a general to all the points held by the [WarTheatre](#).

Precoditions:

- general must be a General*

Postconditions:

- Add general to all points

Parameters

<i>general</i>	must be a General*
----------------	--------------------

Implements [Area](#).Definition at line 127 of file [KeyPoint.cpp](#).

```
00127                                     {  
00128     generals.push_back(general);  
00129 }
```

4.15.3.3 changeWeather()

```
void KeyPoint::changeWeather ( )
```

Switches the [Weather](#) object to the next state.Definition at line 150 of file [KeyPoint.cpp](#).

```

00150                                     {
00151
00152     srand(time(0));
00153
00154     int randomNum = 1 + (rand() % 10);
00155     std::string currWeather = this->weather->getWeather();
00156
00157     if (currWeather == "Sunny" && randomNum > 6) // 60% chance of not changing weather from Sunny and
    staying
00158         this->weather->handleChange(this);
00159     else if (currWeather == "Cloudy" && randomNum > 3) // 30% chance of not changing weather from
    Cloudy and staying
00160         this->weather->handleChange(this);
00161     else if (currWeather == "Rainy" && randomNum > 1) // 10% chance of not changing weather from Rainy
    and staying
00162         this->weather->handleChange(this);
00163
00164
00165 }

```

4.15.3.4 clearBattlefield()

```

void KeyPoint::clearBattlefield (
    Alliance * alliance )

```

Clears the battlefield of all deceased troops.

Postconditions:

- Notify command centers of each troop who is killed

Parameters

<i>alliance</i>	must be an Alliance*
-----------------	----------------------

Returns

void

Definition at line 61 of file [KeyPoint.cpp](#).

```

00061                                     {
00062     int destroyed = 0;
00063     double numUnits = 0;
00064     for (vector<Entity*>::iterator it = entities.begin(); it != entities.end(); ++it) {
00065         if ((*it)->getHealth() <= 0) {
00066             destroyed++;
00067             for (int i = 0; i < generals.size(); i++) {
00068                 if (generals[i]->getAlliance() == (*it)->getAlliance()) {
00069                     generals[i]->initiateStrategy(this);
00070                     delete *it;
00071                     entities.erase(it);
00072                 }
00073             }
00074             } else if ((*it)->getAlliance() == alliance) {
00075                 numUnits++;
00076             }
00077         }
00078
00079     // saving stats
00080     string stats = getAreaName() + ":\n";
00081     stats += "Key Point Satus: ";
00082     if (numUnits / entities.size() >= 0.6) {
00083         stats += "Winning\n";
00084         RoundStats::numWinningPoints++;
00085     } else if (numUnits / entities.size() >= 0.35) {

```

```

00086         stats += "Contested\n";
00087         RoundStats::numContestedPoints++;
00088     } else {
00089         stats += "Losing\n";
00090         RoundStats::numLosingPoints++;
00091     }
00092
00093     stats += "Number of Entities Destroyed by Alliance: " + to_string(destroyed) + "\n";
00094     stats += "Number of Entities/Total Amount of Entities: " + to_string((int)numUnits) + "/" +
to_string(entities.size());
00095
00096     RoundStats::keyPointInformation.push_back(stats);
00097     RoundStats::numEntitiesDestroyed += destroyed;
00098 }

```

4.15.3.5 clone()

`Area * KeyPoint::clone () [virtual]`

Instantiates and returns a clone of the current Keypoint.

Postconditions:

- Returns the clone of the current Keypoint

Returns

Area* The Keypoint clone

Implements [Area](#).

Definition at line 141 of file [KeyPoint.cpp](#).

```

00141     {
00142         return new KeyPoint (*this);
00143     }

```

4.15.3.6 getWeather()

`std::string KeyPoint::getWeather () const`

The weather at the current state is returned.

Returns

string The weather state

Definition at line 167 of file [KeyPoint.cpp](#).

```

00167     {
00168         return this->weather->getWeather();
00169     }

```

4.15.3.7 isKeyPoint()

```
bool KeyPoint::isKeyPoint ( ) [virtual]
```

Returns area type.

Postconditions:

- Returns true

Returns

bool The area type

Implements [Area](#).

Definition at line 32 of file [KeyPoint.cpp](#).

```
00032     {  
00033         return true;  
00034     }
```

4.15.3.8 moveEntitiesInto()

```
void KeyPoint::moveEntitiesInto (  
    Alliance * alliance,  
    int numTroops )
```

Moves a specific alliances troops into this keypoint.

Preconditions:

- alliance must be an Alliance*
- numTroops must be an int

Postconditions:

- Move troops to into this keypoint

Parameters

<i>alliance</i>	must be an Alliance*
<i>numTroops</i>	must be an int

Returns

void

Definition at line 100 of file [KeyPoint.cpp](#).

```
00100                                     {
00101     vector<Entity*> troops = alliance->getReserveEntities(numTroops);
00102     for (int i = 0; i < troops.size(); i++)
00103         entities.push_back(troops[i]);
00104
00105     string stats = "Alliance " + to_string(alliance->getID()) + " moved " + to_string(troops.size()) +
00106     " entities into " + getAreaName();
00107     RoundStats::entityMovementInformation.push_back(stats);
00107 }
```

4.15.3.9 moveEntitiesOutOf()

```
void KeyPoint::moveEntitiesOutOf (
    Alliance * alliance,
    int numTroops )
```

Moves a specific alliances troops out of the keypoint.

Preconditions:

- alliance must be an Alliance*
- numTroops must be an int

Postconditions:

- Move troops to reserve

Parameters

<i>alliance</i>	must be an Alliance*
<i>numTroops</i>	must be an int

Returns

void

Definition at line 109 of file [KeyPoint.cpp](#).

```
00109                                     {
00110     int numMoved = 0;
00111     for (vector<Entity*>::iterator it = entities.begin(); it != entities.end() && numMoved !=
00112     numTroops; ++it) {
00113         if ((*it)->getAlliance() == alliance) {
00114             numMoved++;
00115             alliance->addReserveEntity(*it);
00116             entities.erase(it);
00117         }
00118     }
00119     string stats = "Alliance " + to_string(alliance->getID()) + " moved " + to_string(numMoved) +
00120     " entities out of " + getAreaName();
00121     RoundStats::entityMovementInformation.push_back(stats);
00121 }
```

4.15.3.10 removeGeneral()

```
void KeyPoint::removeGeneral (
    General * general )
```

removes a general to all the points held by the [WarTheatre](#)

Preconditions:

- general must be a General*

Postconditions:

- Add general to all points

Parameters

<i>general</i>	must be a General*
----------------	--------------------

Definition at line 131 of file [KeyPoint.cpp](#).

```
00131 {
00132     for (vector<General*>::iterator it = generals.begin(); it != generals.end(); ++it) {
00133         if (*it == general) {
00134             delete *it;
00135             generals.erase(it);
00136             return;
00137         }
00138     }
00139 }
```

4.15.3.11 setWeather()

```
void KeyPoint::setWeather (
    Weather * weather )
```

Set the [Weather](#) object.

Preconditions:

- weather must be a Weather*

Postconditions:

- must set the keyPoints weather state

Parameters

<i>weather</i>	must be a Weather*
----------------	--------------------

Returns

void

Definition at line 145 of file [KeyPoint.cpp](#).

```

00145                                     {
00146     delete this->weather;
00147     this->weather = weather;
00148 }
```

4.15.3.12 simulateBattle()

```

void KeyPoint::simulateBattle (
    Alliance * alliance ) [virtual]
```

Simulate Battle with troops from the alliance passed in.

Preconditions:

- alliance must be an Alliance*

Postconditions:

- Perform attacks on other alliance troops

Parameters

<i>alliance</i>	must be an Alliance*
-----------------	----------------------

Returns

void

Implements [Area](#).

Definition at line 36 of file [KeyPoint.cpp](#).

```

00036                                     {
00037     int numUnits = 0;
00038     for (int i = 0; i < entities.size(); i++) {
00039         if (entities[i]->getAlliance() == alliance) {
00040             numUnits++;
00041         }
00042     }
00043
00044     if (numUnits != entities.size()) {
00045         for (int i = 0; i < entities.size(); i++) {
00046             if (entities[i]->getAlliance() == alliance) {
00047                 int random;
00048                 do {
00049                     random = rand() % entities.size();
00050                 } while (entities[random]->getAlliance() == alliance);
00051
00052                 if (rand() % (int)(weather->getMultiplier() * 100) <= (int)(weather->getMultiplier() *
00053 100))
00054                     entities[i]->dealDamage(entities[random]);
00055             }
00056         }
00057     }
```

```
00057
00058     clearBattlefield(alliance);
00059 }
```

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

- KeyPoint.h
- KeyPoint.cpp

4.16 Negotiator Class Reference

Public Member Functions

- [Negotiator](#) ()
Instantiates the [Negotiator](#).
- [~Negotiator](#) ()
Destructor for the [Negotiator](#) object.
- bool [sendPeace](#) ([Alliance](#) *offerAlliance)
Tries to offer peace to all the alliances in vector.
- void [removeAlliance](#) ([Alliance](#) *oldAlliance)
Removes an alliance from the alliance vector.
- void [addAlliance](#) ([Alliance](#) *newAlliance)
Adds an alliance to the alliance vector.
- int [getNumAlliances](#) ()
gets the number of Alliances in the negotiator

4.16.1 Detailed Description

Definition at line 6 of file [Negotiator.h](#).

4.16.2 Constructor & Destructor Documentation

4.16.2.1 Negotiator()

```
Negotiator::Negotiator ( )
```

Instantiates the [Negotiator](#).

Definition at line 4 of file [Negotiator.cpp](#).

```
00004 {}
```


4.16.2.2 ~Negotiator()

Negotiator::~~Negotiator ()

Destructor for the [Negotiator](#) object.

Definition at line 6 of file [Negotiator.cpp](#).

```
00006         {
00007     alliances.clear();
00008 }
```

4.16.3 Member Function Documentation

4.16.3.1 addAlliance()

```
void Negotiator::addAlliance (
    Alliance * newAlliance )
```

Adds an alliance to the alliance vector.

Preconditions:

- newAlliance must be an [Alliance](#) pointer

Postconditions:

- [Alliance](#) is added to the vector

Returns

void

Definition at line 34 of file [Negotiator.cpp](#).

```
00034                                     {
00035
00036     if (std::find(alliances.begin(), alliances.end(), newAlliance) != alliances.end())
00037         alliances.push_back(newAlliance);
00038
00039 }
```

4.16.3.2 getNumAlliances()

```
int Negotiator::getNumAlliances ( )
```

gets the number of Alliances in the negotiator

Postconditions:

- Returns the number of alliances

Returns

int Number of alliances

Definition at line 41 of file [Negotiator.cpp](#).

```
00041     {
00042     return this->alliances.size();
00043 }
```

4.16.3.3 removeAlliance()

```
void Negotiator::removeAlliance (
    Alliance * oldAlliance )
```

Removes an alliance from the alliance vector.

Preconditions:

- oldAlliance must be an Alliance pointer

Postconditions:

- Alliance is removed from vector

Returns

void

Definition at line 24 of file Negotiator.cpp.

```
00024                                     {
00025
00026     for (int xx = 0; xx < alliances.size(); xx++)
00027     {
00028         if (alliances[xx]->getID() == oldAlliance->getID())
00029             alliances.erase( alliances.begin() + xx ); // Removes the specific alliances from this
negotiator
00030     }
00031
00032 }
```

4.16.3.4 sendPeace()

```
bool Negotiator::sendPeace (
    Alliance * offerAlliance )
```

Tries to offer peace to all the alliances in vector.

Preconditions:

- offerAlliance must be an Alliance pointer

Postconditions:

- Iterates through alliance vector and calls considerPeace for the enemies

Parameters

<i>id</i>	must be an int
-----------	----------------

Returns

bool

Definition at line 10 of file [Negotiator.cpp](#).

```

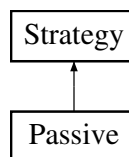
00010                                     {
00011
00012     for (int yy = 0; yy < alliances.size(); yy++)
00013     {
00014         if (alliances[yy] != offerAlliance) {
00015             if (alliances[yy]->considerPeace() == false)
00016                 return false; // There is at least one enemy alliances that does not want the peace
00017         deal
00018     }
00019 }
00020
00021 return true; // All the alliances being fought against agreed to the peace deal
00022 }
```

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

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

4.17 Passive Class Reference

Inheritance diagram for Passive:



Public Member Functions

- void [performStrat](#) ([KeyPoint](#) *keyPoint, [Alliance](#) *alliance)
This function will perform a [Passive](#) strategy.
- [Strategy](#) * [clone](#) ()
Returns the clone of the [Passive Strategy](#) object.

Additional Inherited Members

4.17.1 Detailed Description

Definition at line 6 of file [Passive.h](#).

4.17.2 Constructor & Destructor Documentation

4.17.2.1 Passive()

```
Passive::Passive ( )
```

Definition at line 5 of file [Passive.cpp](#).

```
00005 {}
```

4.17.3 Member Function Documentation

4.17.3.1 clone()

```
Strategy * Passive::clone ( ) [virtual]
```

Returns the clone of the [Passive Strategy](#) object.

Returns

Strategy* The clone of the [Passive Strategy](#) object

Implements [Strategy](#).

Definition at line 13 of file [Passive.cpp](#).

```
00013 {
00014     return new Passive();
00015 }
```

4.17.3.2 performStrat()

```
void Passive::performStrat (
    KeyPoint * keyPoint,
    Alliance * alliance ) [virtual]
```

This function will perform a [Passive](#) strategy.

Parameters

<i>keyPoint</i>	a Passive strategy will then be performed at this specific keypoint
-----------------	-------------------------------------------------------------------------------------

Returns

void The function will return void

Implements [Strategy](#).

Definition at line 7 of file [Passive.cpp](#).

```
00007
```

```
{
```

```

00008
00009     int randomNumber = (rand() % 10) + 5;
00010     keyPoint->moveEntitiesOutOf(alliance, randomNumber);
00011 }

```

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

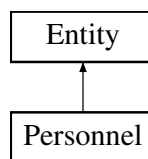
- Passive.h
- Passive.cpp

4.18 Personnel Class Reference

[Personnel](#) class.

```
#include <Personnel.h>
```

Inheritance diagram for Personnel:



Public Member Functions

- [Personnel](#) ([Type](#) *type, int health=100, int damage=10)
Instantiates the [Personnel](#).
- void [takeDamage](#) (int damage)
Removes health from the [Personnel](#) object.
- void [dealDamage](#) ([Entity](#) *entity)
Inflicts damage onto another entity.
- [Entity](#) * [clone](#) ()
Clones the current [Entity](#) object and returns the cloned object.

4.18.1 Detailed Description

[Personnel](#) class.

Used to add additional functionality to [Entity](#) objects.

Definition at line 11 of file [Personnel.h](#).

4.18.2 Constructor & Destructor Documentation

4.18.2.1 Personnel()

```

Personnel::Personnel (
    Type * type,
    int health = 100,
    int damage = 10 )

```

Instantiates the [Personnel](#).

Parameters

<i>health</i>	must be an int
<i>damage</i>	must be an int
<i>type</i>	must be a Type*

Definition at line 6 of file [Personnel.cpp](#).

```
00006 : Entity(type, health, damage) {}
```

4.18.3 Member Function Documentation

4.18.3.1 clone()

```
Entity * Personnel::clone ( ) [virtual]
```

Clones the current [Entity](#) object and returns the cloned object.

PostConditions:

- Returns the cloned object of [Entity](#)

Returns

Entity* The cloned object

Implements [Entity](#).

Definition at line 20 of file [Personnel.cpp](#).

```
00020 {
00021     Personnel* p;
00022     if (this->getType() == NULL) {
00023         p = new Personnel(NULL, this->getHealth(), this->getDamage());
00024     } else {
00025         p = new Personnel(this->getType()->clone(), this->getHealth(), this->getDamage());
00026     }
00027
00028     p->setAlliance(this->getAlliance());
00029
00030     return p;
00031 }
```

4.18.3.2 dealDamage()

```
void Personnel::dealDamage (
    Entity * entity ) [virtual]
```

Inflicts damage onto another entity.

Preconditions:

- entity must be an Entity*

Postconditions:

- Reduces the health of the entity

Parameters

<i>entity</i>	must be an Entity*
---------------	--------------------

Returns

void

Implements [Entity](#).Definition at line 15 of file [Personnel.cpp](#).

```
00015 {
00016     RoundStats::damageDone += getDamage\(\);
00017     entity->takeDamage\(getDamage\(\)\);
00018 }
```

4.18.3.3 takeDamage()

```
void Personnel::takeDamage (
    int damage ) [virtual]
```

Removes health from the [Personnel](#) object.

Preconditions:

- damage must be an int

Postconditions:

- Reduces the health of the [Personnel](#) object

Exception:

- damage less than 0

Parameters

<i>damage</i>	must be an int and greater than 0
---------------	-----------------------------------

Returns

void

Implements [Entity](#).Definition at line 8 of file [Personnel.cpp](#).

```
00008 {
```

```

00009     if (damage <= 0)
00010         throw std::invalid_argument("damage must be greater than zero");
00011
00012     setHealth(getHealth() - damage);
00013 }

```

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

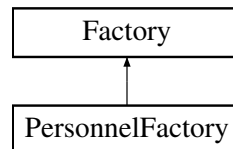
- Personnel.h
- Personnel.cpp

4.19 PersonnelFactory Class Reference

[PersonnelFactory](#) class.

```
#include <PersonnelFactory.h>
```

Inheritance diagram for PersonnelFactory:



Public Member Functions

- [PersonnelFactory](#) ([Type](#) *type, [AddOn](#) *addOn)
Instantiates the [Personnel](#) factory.
- [Entity](#) * [createEntity](#) ([Alliance](#) *alliance)
Instantiates and returns a [Personnel](#) for the given alliance.
- [Factory](#) * [clone](#) ()
Instantiates and returns a clone of the current [Personnel](#) factory.

4.19.1 Detailed Description

[PersonnelFactory](#) class.

Used to instantiate [Personnel](#) objects.

Definition at line 11 of file [PersonnelFactory.h](#).

4.19.2 Constructor & Destructor Documentation

4.19.2.1 PersonnelFactory()

```

PersonnelFactory::PersonnelFactory (
    Type * type,
    AddOn * addOn )

```

Instantiates the [Personnel](#) factory.

Parameters

<i>type</i>	must be a Type*
<i>addOn</i>	must be a AddOn*

Definition at line 5 of file [PersonnelFactory.cpp](#).

```
00005 : Factory(type, addOn) {}
```

4.19.3 Member Function Documentation

4.19.3.1 clone()

```
Factory * PersonnelFactory::clone ( ) [virtual]
```

Instantiates and returns a clone of the current [Personnel](#) factory.

Postconditions:

- Returns the clone of the current [Personnel](#) factory

Returns

[Factory](#)* The [Personnel](#) factory clone

Implements [Factory](#).

Definition at line 19 of file [PersonnelFactory.cpp](#).

```
00019 {  
00020     return new PersonnelFactory(getType(), getAddOn());  
00021 }
```

4.19.3.2 createEntity()

```
Entity * PersonnelFactory::createEntity (  
    Alliance * alliance ) [virtual]
```

Instantiates and returns a [Personnel](#) for the given alliance.

Preconditions:

- alliance must be an [Alliance](#)*

Postconditions:

- Returns the instantiated [Personnel](#) object with specific state

Parameters

<i>alliance</i>	must be a Alliance*
-----------------	---------------------

Returns

Entity* The instantiated personnel

Implements [Factory](#).

Definition at line 7 of file [PersonnelFactory.cpp](#).

```

00007
00008     Personnel* p = new Personnel(getType()->clone());
00009     p->setAlliance(alliance);
00010     if (getAddOn() != NULL) {
00011         AddOn* personnelAddOn = getAddOn()->clone();
00012         personnelAddOn->setEntity(p);
00013         return personnelAddOn;
00014     } else {
00015         return p;
00016     }
00017 }
```

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

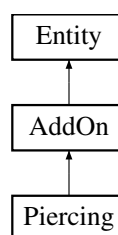
- PersonnelFactory.h
- PersonnelFactory.cpp

4.20 Piercing Class Reference

[Piercing](#) class.

```
#include <Piercing.h>
```

Inheritance diagram for Piercing:



Public Member Functions

- [Piercing](#) (int value)
Instantiates an [Piercing](#).
- void [takeDamage](#) (int damage)
Decreases the entities' armour value (or health when their armour has depleted)
- void [dealDamage](#) (Entity *entity)
Adds to the damage [Entity](#) objects inflict.
- [AddOn](#) * [clone](#) ()
Instantiates and returns a clone of the current [Piercing](#).

Additional Inherited Members

4.20.1 Detailed Description

[Piercing](#) class.

Used to add to the damage [Entity](#) objects inflict.

Definition at line 11 of file [Piercing.h](#).

4.20.2 Constructor & Destructor Documentation

4.20.2.1 Piercing()

```
Piercing::Piercing (
    int value )
```

Instantiates an [Piercing](#).

Parameters

<i>value</i>	must be an int
--------------	----------------

Definition at line 5 of file [Piercing.cpp](#).

```
00005 : AddOn(value) {}
```

4.20.3 Member Function Documentation

4.20.3.1 clone()

```
AddOn * Piercing::clone ( ) [virtual]
```

Instantiates and returns a clone of the current [Piercing](#).

Postconditions:

- Returns the clone of the current [Piercing](#)

Returns

[Piercing](#)* The [Piercing](#) clone

Implements [AddOn](#).

Definition at line 20 of file [Piercing.cpp](#).

```
00020 {
00021     Piercing* piercing = new Piercing(value);
00022     if (getEntity() != NULL)
00023         piercing->setEntity(entity->clone());
00024     return piercing;
00025 }
```

4.20.3.2 dealDamage()

```
void Piercing::dealDamage (
    Entity * entity ) [virtual]
```

Adds to the damage [Entity](#) objects inflict.

Preconditions:

- entity must be an Entity*

Postconditions:

- Inflicts damage to passed in [Entity](#) objects using the sum of it's value and the entity onto which it has been added's value

Parameters

<i>entity</i>	must be an Entity*
---------------	--------------------

Returns

void

Implements [AddOn](#).

Definition at line 14 of file [Piercing.cpp](#).

```
00014 {
00015     int sumValue = this->entity->getDamage() + value;
00016     entity->takeDamage(sumValue);
00017     RoundStats::damageDone += sumValue;
00018 }
```

4.20.3.3 takeDamage()

```
void Piercing::takeDamage (
    int damage ) [virtual]
```

Decreases the entities' armour value (or health when their armour has depleted)

Preconditions:

- damage must be an int

Postconditions:

- Does nothing

Exceptions:

- damage less than 0

Parameters

<code>damage</code>	must be an int and greater than 0
---------------------	-----------------------------------

Returns

void

Implements [AddOn](#).Definition at line 7 of file [Piercing.cpp](#).

```

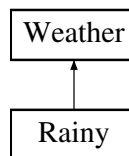
00007                                     {
00008     if (damage <= 0)
00009         throw std::invalid_argument("damage must be greater than zero");
00010
00011     entity->takeDamage(damage);
00012 }
```

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

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

4.21 Rainy Class Reference

Inheritance diagram for Rainy:



Public Member Functions

- [Rainy](#) ()
Instantiates the Runny object of the state pattern.
- `std::string` [getWeather](#) ()
Returns string which tels us the weather.
- `void` [handleChange](#) ([KeyPoint](#) *keypoint)
Will change the current state of the weather inside the specific keypoint.
- [Weather](#) * [clone](#) ()
Returns a clone of the [Rainy](#) object.

Additional Inherited Members

4.21.1 Detailed Description

Definition at line 6 of file [Rainy.h](#).

4.21.2 Constructor & Destructor Documentation

4.21.2.1 Rainy()

```
Rainy::Rainy ( )
```

Instantiates the Runny object of the state pattern.

Definition at line 4 of file [Rainy.cpp](#).

```
00004      : Weather() {  
00005      this->multiplier = 0.5;  
00006 }
```

4.21.3 Member Function Documentation

4.21.3.1 clone()

```
Weather * Rainy::clone ( ) [virtual]
```

Returns a clone of the [Rainy](#) object.

Returns

Weather* Clone of [Rainy](#) object

Implements [Weather](#).

Definition at line 17 of file [Rainy.cpp](#).

```
00017 {  
00018     return new Rainy();  
00019 }
```

4.21.3.2 getWeather()

```
std::string Rainy::getWeather ( ) [virtual]
```

Returns string which tells us the weather.

Postconditions:

- Returns the wether of ths current state

Returns

std::string which is the current state

Implements [Weather](#).

Definition at line 8 of file [Rainy.cpp](#).

```
00008 {  
00009     return "Rainy";  
00010 }
```

4.21.3.3 handleChange()

```
void Rainy::handleChange (
    KeyPoint * keypoint ) [virtual]
```

Will change the current state of the weather inside the specific keypoint.

Preconditions:

- keypoint must be a [KeyPoint](#)*

Postconditions:

- Changes the current weather to the next one in the state pattern ([Sunny](#))

Parameters

<i>keypoint</i>	must be a KeyPoint *
-----------------	--------------------------------------

Returns

void

Implements [Weather](#).

Definition at line 12 of file [Rainy.cpp](#).

```
00012 {
00013     Sunny* newWeather = new Sunny();
00014     keypoint->setWeather(newWeather);
00015 }
```

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

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

4.22 RoundStats Class Reference

Static Public Member Functions

- static void [clearStats](#) ()
- static string [toString](#) ()

Static Public Attributes

- static int [numEntitiesCreated](#) = 0
- static int [numEntitiesDestroyed](#) = 0
- static int [damageDone](#) = 0
- static int [numLosingPoints](#) = 0
- static int [numContestedPoints](#) = 0
- static int [numWinningPoints](#) = 0
- static vector< string > [keyPointInformation](#)
- static vector< string > [entityMovementInformation](#)

4.22.1 Detailed Description

Definition at line 9 of file [RoundStats.h](#).

4.22.2 Member Function Documentation

4.22.2.1 clearStats()

```
void RoundStats::clearStats ( ) [static]
```

Definition at line 12 of file [RoundStats.cpp](#).

```
00012     {
00013         numEntitiesCreated = 0;
00014         numEntitiesDestroyed = 0;
00015         damageDone = 0;
00016         numLosingPoints = 0;
00017         numContestedPoints = 0;
00018         numWinningPoints = 0;
00019         keyPointInformation.clear();
00020         entityMovementInformation.clear();
00021     }
```

4.22.2.2 toString()

```
string RoundStats::toString ( ) [static]
```

Definition at line 23 of file [RoundStats.cpp](#).

```
00023     {
00024         string out = "Number of Key Points Winning/Contested/Losing: " + to_string(numWinningPoints) +
"/" + to_string(numContestedPoints) + "/" + to_string(numLosingPoints) + "\n";
00025         out += "Number of Entities Created: " + to_string(numEntitiesCreated) + "\n";
00026         out += "Number of Entities Destroyed by Alliance: " + to_string(numEntitiesDestroyed) + "\n";
00027         out += "Damage Given by Alliance: " + to_string(damageDone) + "\n";
00028
00029         out += "\nKey Point Round Information:\n";
00030         for (int i = 0; i < keyPointInformation.size(); i++)
00031             out += keyPointInformation[i] + "\n";
00032
00033         out += "\nMovement Round Information:\n";
00034         for (int i = 0; i < entityMovementInformation.size(); i++)
00035             out += entityMovementInformation[i] + "\n";
00036
00037         return out;
00038     }
```

4.22.3 Member Data Documentation

4.22.3.1 damageDone

```
int RoundStats::damageDone = 0 [static]
```

Definition at line 13 of file [RoundStats.h](#).

4.22.3.2 entityMovementInformation

```
vector< string > RoundStats::entityMovementInformation [static]
```

Definition at line 18 of file [RoundStats.h](#).

4.22.3.3 keyPointInformation

```
vector< string > RoundStats::keyPointInformation [static]
```

Definition at line 17 of file [RoundStats.h](#).

4.22.3.4 numContestedPoints

```
int RoundStats::numContestedPoints = 0 [static]
```

Definition at line 15 of file [RoundStats.h](#).

4.22.3.5 numEntitiesCreated

```
int RoundStats::numEntitiesCreated = 0 [static]
```

Definition at line 11 of file [RoundStats.h](#).

4.22.3.6 numEntitiesDestroyed

```
int RoundStats::numEntitiesDestroyed = 0 [static]
```

Definition at line 12 of file [RoundStats.h](#).

4.22.3.7 numLosingPoints

```
int RoundStats::numLosingPoints = 0 [static]
```

Definition at line 14 of file [RoundStats.h](#).

4.22.3.8 numWinningPoints

```
int RoundStats::numWinningPoints = 0 [static]
```

Definition at line 16 of file [RoundStats.h](#).

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

- RoundStats.h
- RoundStats.cpp

4.23 SaveArchive Class Reference

Stores a list of mementos containing simulation state.

```
#include <SaveArchive.h>
```

Public Member Functions

- [SaveArchive](#) ()
Instantiates the [SaveArchive](#) class.
- void [addNewSave](#) (std::string newSaveName, [WarEngineMemento](#) *newSave)
Adds a new save to the list of stored mementos.
- [WarEngineMemento](#) * [getLastSave](#) ()
Returns the last saved memento.
- [WarEngineMemento](#) * [getSave](#) (std::string name)
Returns the last saved memento. Preconditions:
- void [clearSaveList](#) ()
Erases all saved mementos from the list of saves. Postconditions:
- void [deleteSave](#) (std::string name)
Deletes a memento with the matching given name from the list of saved mementos. Preconditions:

4.23.1 Detailed Description

Stores a list of mementos containing simulation state.

Definition at line 11 of file [SaveArchive.h](#).

4.23.2 Constructor & Destructor Documentation

4.23.2.1 SaveArchive()

```
SaveArchive::SaveArchive ( )
```

Instantiates the [SaveArchive](#) class.

Definition at line 3 of file [SaveArchive.cpp](#).

```
00003 {}
```

4.23.3 Member Function Documentation

4.23.3.1 addNewSave()

```
void SaveArchive::addNewSave (
    std::string newSaveName,
    WarEngineMemento * newSave )
```

Adds a new save to the list of stored mementos.

Preconditions:

- newSave must be a WarEngineMemento*
- newSaveName must be a string

Postconditions:

- Adds a new memento to list of saves

Parameters

<i>newSave</i>	must be a WarEngineMemento*
<i>newSaveName</i>	must be a string

Returns

void

Definition at line 5 of file [SaveArchive.cpp](#).

```
00005
00006     saveList.insert({newSaveName, newSave});
00007 }
```

4.23.3.2 clearSaveList()

```
void SaveArchive::clearSaveList ( )
```

Erases all saved mementos from the list of saves. Postconditions:

- Clears all elements in the saveList vector

Returns

void

Definition at line 35 of file [SaveArchive.cpp](#).

```
00035     {
00036         saveList.clear();
00037     }
```

4.23.3.3 deleteSave()

```
void SaveArchive::deleteSave (
    std::string name )
```

Deletes a memento with the matching given name from the list of saved mementos. Preconditions:

- name must be a string in date/time format

Postconditions:

- Removes the element in the saveList vector with a name matching that of the parameter

Parameters

<i>name</i>	a string
-------------	----------

Returns

void

Exceptions

<i>std::out_of_range</i>	save archive is empty
--------------------------	-----------------------

Definition at line 39 of file [SaveArchive.cpp](#).

```
00039     {
00040         if(saveList.size() == 0){
00041             std::__throw_out_of_range("Save archive is empty");
00042         }
00043
00044         auto iter = saveList.find(name) ;
00045
00046         if(iter == saveList.end())
00047             return;
00048
00049         saveList.erase( iter );
00050     }
```

4.23.3.4 getLastSave()

```
WarEngineMemento * SaveArchive::getLastSave ( )
```

Returns the last saved memento.

Postconditions:

- Returns the last element in the saveList vector

Returns

WarEngineMemento*

Exceptions

<i>std::out_of_range</i>	save archive is empty
<i>std::invalid_argument</i>	memento with given name is not found in memento list.

Definition at line 9 of file [SaveArchive.cpp](#).

```
00009                                     {
00010
00011     if(saveList.size() == 0){
00012         throw "Save archive is empty.";
00013     }
00014
00015     WarEngineMemento* lastSave = saveList.begin()->second;
00016
00017     saveList.erase( saveList.begin() );
00018
00019     return lastSave;
00020 }
```

4.23.3.5 getSave()

```
WarEngineMemento * SaveArchive::getSave (
    std::string name )
```

Returns the last saved memento. Preconditions:

- name must be a string

Postconditions:

- Returns the last element in the saveList vector

Parameters

<i>name</i>	a string
-------------	----------

Returns

WarEngineMemento*

Exceptions

<code>std::out_of_range</code>	save archive is empty
--------------------------------	-----------------------

Definition at line 22 of file [SaveArchive.cpp](#).

```

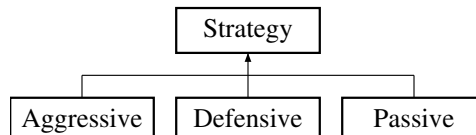
00022                                     {
00023     if(saveList.size() == 0){
00024         std::__throw_out_of_range("Save archive is empty");
00025     }
00026
00027     auto iter = saveList.find(name);
00028
00029     if(iter == saveList.end())
00030         std::__throw_invalid_argument("No save with given name exists");
00031
00032     return iter->second;
00033 }
```

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

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

4.24 Strategy Class Reference

Inheritance diagram for Strategy:

**Public Member Functions**

- [Strategy](#) ()
Construct a new [Strategy](#) object.
- [~Strategy](#) ()
Destroy the [Strategy](#) object.
- virtual void [performStrat](#) ([KeyPoint](#) *keyPoint, [Alliance](#) *alliance)=0
This function will perform a strategy.
- virtual [Strategy](#) * [clone](#) ()=0
Returns the cloned [Strategy](#) object.

Protected Attributes

- std::string [strategy](#)

4.24.1 Detailed Description

Definition at line 10 of file [Strategy.h](#).

4.24.2 Constructor & Destructor Documentation

4.24.2.1 Strategy()

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

Construct a new [Strategy](#) object.

Definition at line 7 of file [Strategy.cpp](#).

```
00007 {}
```

4.24.2.2 ~Strategy()

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

Destroy the [Strategy](#) object.

Definition at line 9 of file [Strategy.cpp](#).

```
00009 {}
```

4.24.3 Member Function Documentation

4.24.3.1 clone()

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

Returns the cloned [Strategy](#) object.

PostConditions:

- Returns the clone of the current [Strategy](#)

Returns

[Strategy](#)* The cloned object

Implemented in [Aggressive](#), [Defensive](#), and [Passive](#).

4.24.3.2 performStrat()

```
virtual void Strategy::performStrat (
    KeyPoint * keyPoint,
    Alliance * alliance ) [pure virtual]
```

This function will perform a strategy.

Parameters

<i>keyPoint</i>	a strategy will then be performed at this specific keypoint
-----------------	-------------------------------------------------------------

Returns

void The function will return void

Implemented in [Aggressive](#), [Defensive](#), and [Passive](#).

4.24.4 Member Data Documentation

4.24.4.1 strategy

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

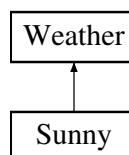
Definition at line 13 of file [Strategy.h](#).

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

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

4.25 Sunny Class Reference

Inheritance diagram for Sunny:



Public Member Functions

- [Sunny](#) ()
Instantiates the [Sunny](#) object of the state pattern.
- virtual std::string [getWeather](#) ()
Returns string which tells us the weather.
- virtual void [handleChange](#) ([KeyPoint](#) *keypoint)
Will change the current state of the weather inside the specific keypoint.
- [Weather](#) * [clone](#) ()
Returns the cloned object of [Sunny](#) PostConditions:

Additional Inherited Members

4.25.1 Detailed Description

Definition at line 8 of file [Sunny.h](#).

4.25.2 Constructor & Destructor Documentation

4.25.2.1 Sunny()

```
Sunny::Sunny ( )
```

Instantiates the [Sunny](#) object of the state pattern.

Definition at line 4 of file [Sunny.cpp](#).

```
00004         {  
00005     this->multiplier = 1.0;  
00006 }
```

4.25.3 Member Function Documentation

4.25.3.1 clone()

```
Weather * Sunny::clone ( ) [virtual]
```

Returns the cloned object of [Sunny](#) PostConditions:

- Returns cloned object of [Sunny](#)

Returns

[Weather](#)* The cloned object

Implements [Weather](#).

Definition at line 17 of file [Sunny.cpp](#).

```
00017         {  
00018     return new Sunny();  
00019 }
```

4.25.3.2 `getWeather()`

```
std::string Sunny::getWeather ( ) [virtual]
```

Returns string which tells us the weather.

Postconditions:

- Returns the wether of ths current state

Returns

std::string which is the current state

Implements [Weather](#).

Definition at line 8 of file [Sunny.cpp](#).

```
00008 {  
00009     return "Sunny";  
00010 }
```

4.25.3.3 `handleChange()`

```
void Sunny::handleChange (  
    KeyPoint * keypoint ) [virtual]
```

Will change the current state of the weather inside the specific keypoint.

Preconditions:

- k must be a [KeyPoint*](#)

Postconditions:

- Changes the current weather to the next one in the state pattern ([Cloudy](#))

Parameters

<i>keypoint</i>	must be a KeyPoint*
-----------------	-------------------------------------

Returns

void

Implements [Weather](#).

Definition at line 12 of file [Sunny.cpp](#).

```

00012     {
00013         Cloudy* newWeather = new Cloudy();
00014         k->setWeather(newWeather);
00015     }

```

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

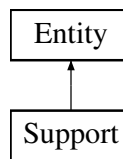
- Sunny.h
- Sunny.cpp

4.26 Support Class Reference

[Support](#) class.

```
#include <Support.h>
```

Inheritance diagram for Support:



Public Member Functions

- [Support](#) ([Type](#) *type, int health=1000, int damage=30)
Instantiates the support.
- void [takeDamage](#) (int damage)
Removes health from the support object.
- void [dealDamage](#) ([Entity](#) *entity)
Inflicts damage onto another entity.
- [Entity](#) * [clone](#) ()
Returns the clone of the [Support](#) object.

4.26.1 Detailed Description

[Support](#) class.

Used to add additional functionality to [Entity](#) objects.

Definition at line 11 of file [Support.h](#).

4.26.2 Constructor & Destructor Documentation

4.26.2.1 Support()

```

Support::Support (
    Type * type,
    int health = 1000,
    int damage = 30 )

```

Instantiates the support.

Parameters

<i>health</i>	must be an int
<i>damage</i>	must be an int
<i>type</i>	must be a Type*

Definition at line 5 of file [Support.cpp](#).

```
00005 : Entity(type, health, damage) {}
```

4.26.3 Member Function Documentation

4.26.3.1 clone()

```
Entity * Support::clone ( ) [virtual]
```

Returns the clone of the [Support](#) object.

Returns

Entity* The clone of the support object

Implements [Entity](#).

Definition at line 19 of file [Support.cpp](#).

```
00019 {
00020     Support* s;
00021     if (this->getType() == NULL) {
00022         s = new Support(NULL, this->getHealth(), this->getDamage());
00023     } else {
00024         s = new Support(this->getType()->clone(), this->getHealth(), this->getDamage());
00025     }
00026
00027     s->setAlliance(this->getAlliance());
00028
00029     return s;
00030 }
```

4.26.3.2 dealDamage()

```
void Support::dealDamage (
    Entity * entity ) [virtual]
```

Inflicts damage onto another entity.

Preconditions:

- entity must be an Entity*

Postconditions:

- Reduces the health of the entity

Parameters

<i>entity</i>	must be an Entity*
---------------	--------------------

Returns

void

Implements [Entity](#).Definition at line 7 of file [Support.cpp](#).

```
00007 {  
00008     RoundStats::damageDone += getDamage\(\);  
00009     entity->takeDamage(getDamage\(\));  
00010 }
```

4.26.3.3 takeDamage()

```
void Support::takeDamage (  
    int damage ) [virtual]
```

Removes health from the support object.

Preconditions:

- damage must be an int

Postconditions:

- Reduces the health of the support object

Exceptions:

- damage less than 0

Parameters

<i>damage</i>	must be an int and greater than 0
---------------	-----------------------------------

Returns

void

Implements [Entity](#).Definition at line 12 of file [Support.cpp](#).

```
00012 {
```

```

00013     if (damage <= 0)
00014         throw std::invalid_argument("damage must be greater than zero");
00015
00016     this->setHealth(this->getHealth() - damage);
00017 }

```

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

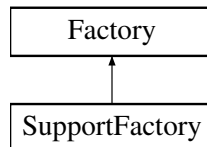
- Support.h
- Support.cpp

4.27 SupportFactory Class Reference

[SupportFactory](#) class.

```
#include <SupportFactory.h>
```

Inheritance diagram for SupportFactory:



Public Member Functions

- [SupportFactory](#) ([Type](#) *type, [AddOn](#) *addOn)
Instantiates the support factory.
- [Entity](#) * [createEntity](#) ([Alliance](#) *alliance)
Instantiates and returns a support for the given alliance.
- [Factory](#) * [clone](#) ()
Instantiates and returns a clone of the current support factory.

4.27.1 Detailed Description

[SupportFactory](#) class.

Used to instantiate [Support](#) objects.

Definition at line 11 of file [SupportFactory.h](#).

4.27.2 Constructor & Destructor Documentation

4.27.2.1 SupportFactory()

```

SupportFactory::SupportFactory (
    Type * type,
    AddOn * addOn )

```

Instantiates the support factory.

Parameters

<i>type</i>	must be a Type*
<i>addOn</i>	must be a AddOn*

Definition at line 4 of file [SupportFactory.cpp](#).

```
00004 : Factory(type, addOn) {}
```

4.27.3 Member Function Documentation

4.27.3.1 clone()

```
Factory * SupportFactory::clone ( ) [virtual]
```

Instantiates and returns a clone of the current support factory.

Postconditions:

- Returns the clone of the current support factory

Returns

Factory* The support factory clone

Implements [Factory](#).

Definition at line 18 of file [SupportFactory.cpp](#).

```
00018 {  
00019     return new SupportFactory(getType()->clone(), getAddOn()->clone());  
00020 }
```

4.27.3.2 createEntity()

```
Entity * SupportFactory::createEntity (  
    Alliance * alliance ) [virtual]
```

Instantiates and returns a support for the given alliance.

Preconditions:

- alliance must be an Alliance*

Postconditions:

- Returns the instantiated support object with specific state

Parameters

<i>alliance</i>	must be a Alliance*
-----------------	---------------------

Returns

Entity* The instatiated support

Implements [Factory](#).

Definition at line 6 of file [SupportFactory.cpp](#).

```

00006                                     {
00007     Support* s = new Support (getType()->clone());
00008     s->setAlliance(alliance);
00009     if (getAddOn() != NULL) {
00010         AddOn* personnelAddOn = getAddOn()->clone();
00011         personnelAddOn->setEntity(s);
00012         return personnelAddOn;
00013     } else {
00014         return s;
00015     }
00016 }
```

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

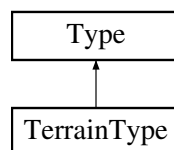
- SupportFactory.h
- SupportFactory.cpp

4.28 TerrainType Class Reference

[TerrainType](#) class.

```
#include <TerrainType.h>
```

Inheritance diagram for TerrainType:

**Public Member Functions**

- [TerrainType](#) ()
Instantiates the terrain type.
- string [getTypeDesc](#) ()
Returns terrain type description.
- [Type](#) * [clone](#) ()
returns the the cloned object of [Type](#)

4.28.1 Detailed Description

[TerrainType](#) class.

Used to define [Entity](#) objects as terrain type.

Definition at line 11 of file [TerrainType.h](#).

4.28.2 Constructor & Destructor Documentation

4.28.2.1 TerrainType()

```
TerrainType::TerrainType ( )
```

Instantiates the terrain type.

Definition at line 3 of file [TerrainType.cpp](#).

```
00003 {}
```

4.28.3 Member Function Documentation

4.28.3.1 clone()

```
Type * TerrainType::clone ( ) [virtual]
```

returns the the cloned object of [Type](#)

PostConditions:

- returns [Type](#)* type

Returns

[Type](#)* The cloned [Type](#) object

Implements [Type](#).

Definition at line 9 of file [TerrainType.cpp](#).

```
00009 {  
00010     return new TerrainType();  
00011 }
```

4.28.3.2 getTypeDesc()

```
string TerrainType::getTypeDesc ( ) [virtual]
```

Returns terrain type description.

Postconditions:

- Returns the terrain type

Returns

string The terrain type string

Implements [Type](#).

Definition at line 5 of file [TerrainType.cpp](#).

```
00005 {
00006     return "Terrain";
00007 }
```

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

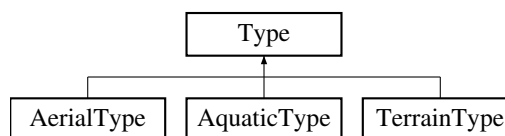
- TerrainType.h
- TerrainType.cpp

4.29 Type Class Reference

[Type](#) class.

```
#include <Type.h>
```

Inheritance diagram for Type:



Public Member Functions

- [Type](#) ()
Instantiates the type.
- virtual string [getTypeDesc](#) ()=0
Returns terrain type description.
- virtual [Type](#) * [clone](#) ()=0
returns the the cloned object of [Type](#)

4.29.1 Detailed Description

[Type](#) class.

Used to define [Entity](#) objects type.

Definition at line 13 of file [Type.h](#).

4.29.2 Constructor & Destructor Documentation

4.29.2.1 Type()

```
Type::Type ( )
```

Instantiates the type.

Definition at line 3 of file [Type.cpp](#).

```
00003 {}
```

4.29.3 Member Function Documentation

4.29.3.1 clone()

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

returns the the cloned object of [Type](#)

PostConditions:

- returns `Type* type`

Returns

`Type*` The cloned [Type](#) object

Implemented in [AerialType](#), [AquaticType](#), and [TerrainType](#).

4.29.3.2 getTypeDesc()

```
virtual string Type::getTypeDesc ( ) [pure virtual]
```

Returns terrain type description.

Postconditions:

- Returns the terrain type

Returns

string The terrain type string

Implemented in [AerialType](#), [AquaticType](#), and [TerrainType](#).

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

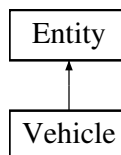
- Type.h
- Type.cpp

4.30 Vehicle Class Reference

[Vehicle](#) class.

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

Inheritance diagram for Vehicle:



Public Member Functions

- [Vehicle](#) ([Type](#) *type, int health=500, int damage=10)
Instantiates the vehicle.
- void [takeDamage](#) (int damage)
Removes health from the vehicle object.
- void [dealDamage](#) ([Entity](#) *entity)
Inflicts damage onto another entity.
- [Entity](#) * [clone](#) ()
Returns the clone of the [Vehicle](#) object.

4.30.1 Detailed Description

[Vehicle](#) class.

Used to add additional functionality to [Entity](#) objects.

Definition at line 11 of file [Vehicle.h](#).

4.30.2 Constructor & Destructor Documentation

4.30.2.1 Vehicle()

```
Vehicle::Vehicle (
    Type * type,
    int health = 500,
    int damage = 10 )
```

Instantiates the vehicle.

Parameters

<i>health</i>	must be an int
<i>damage</i>	must be an int
<i>type</i>	must be a Type*

Definition at line 5 of file [Vehicle.cpp](#).

```
00005 : Entity(type, health, damage) {}
```

4.30.3 Member Function Documentation

4.30.3.1 clone()

```
Entity * Vehicle::clone ( ) [virtual]
```

Returns the clone of the [Vehicle](#) object.

Returns

Entity* The clone of the vehicle object

Implements [Entity](#).

Definition at line 19 of file [Vehicle.cpp](#).

```
00019         {
00020     Vehicle* v;
00021     if (this->getType() == NULL) {
00022         v = new Vehicle(NULL, this->getHealth(), this->getDamage());
00023     } else {
00024         v = new Vehicle(this->getType()->clone(), this->getHealth(), this->getDamage());
00025     }
00026     v->setAlliance(this->getAlliance());
00027     return v;
00028 }
00029
00030 }
```

4.30.3.2 dealDamage()

```
void Vehicle::dealDamage (
    Entity * entity ) [virtual]
```

Inflicts damage onto another entity.

Preconditions:

- entity must be an Entity*

Postconditions:

- Reduces the health of the entity

Parameters

<i>entity</i>	must be an Entity*
---------------	--------------------

Returns

void

Implements [Entity](#).

Definition at line 14 of file [Vehicle.cpp](#).

```
00014     {
00015     RoundStats::damageDone += getDamage();
00016     entity->takeDamage(getDamage());
00017 }
```

4.30.3.3 takeDamage()

```
void Vehicle::takeDamage (
    int damage ) [virtual]
```

Removes health from the vehicle object.

Preconditions:

- damage must be an int

Postconditions:

- does nothing

Exceptions:

- damage less than 0

Parameters

<i>damage</i>	must be an int and greater than 0
---------------	-----------------------------------

Returns

void

Implements [Entity](#).

Definition at line 7 of file [Vehicle.cpp](#).

```
00007 {
00008     if (damage <= 0)
00009         throw std::invalid_argument("damage must be greater than zero");
00010
00011     setHealth(getHealth() - damage);
00012 }
```

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

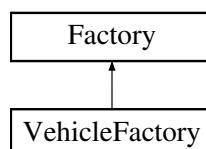
- Vehicle.h
- Vehicle.cpp

4.31 VehicleFactory Class Reference

[VehicleFactory](#) class.

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

Inheritance diagram for VehicleFactory:



Public Member Functions

- [VehicleFactory](#) ([Type](#) *type, [AddOn](#) *addOn)
Instantiates the vehicle factory.
- [Entity](#) * [createEntity](#) ([Alliance](#) *alliance)
Instantiates and returns a vehicle for the given alliance.
- [Factory](#) * [clone](#) ()
Instantiates and returns a clone of the current vehicle factory.

4.31.1 Detailed Description

[VehicleFactory](#) class.

Used to instantiate [Vehicle](#) objects.

Definition at line 10 of file [VehicleFactory.h](#).

4.31.2 Constructor & Destructor Documentation

4.31.2.1 VehicleFactory()

```
VehicleFactory::VehicleFactory (
    Type * type,
    AddOn * addOn )
```

Instantiates the vehicle factory.

Parameters

<i>type</i>	must be a Type *
<i>addOn</i>	must be a AddOn *

Definition at line 4 of file [VehicleFactory.cpp](#).

```
00004 : Factory(type, addOn) {}
```

4.31.3 Member Function Documentation

4.31.3.1 clone()

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

Instantiates and returns a clone of the current vehicle factory.

Postconditions:

- Returns the clone of the current vehicle factory

Returns

Factory* The vehicle factory clone

Implements [Factory](#).

Definition at line 18 of file [VehicleFactory.cpp](#).

```
00018 {
00019     return new VehicleFactory(getType()->clone(), getAddOn()->clone());
00020 }
```

4.31.3.2 createEntity()

```
Entity * VehicleFactory::createEntity (
    Alliance * alliance ) [virtual]
```

Instantiates and returns a vehicle for the given alliance.

Preconditions:

- alliance must be an Alliance*

Postconditions:

- Returns the instantiated vehicle object with specific state

Parameters

<i>alliance</i>	must be a Alliance*
-----------------	---------------------

Returns

Vehicle* The instantiated vehicle

Implements [Factory](#).

Definition at line 6 of file [VehicleFactory.cpp](#).

```
00006 {
00007     Vehicle* v = new Vehicle(getType()->clone());
00008     v->setAlliance(alliance);
00009     if (getAddOn() != NULL) {
00010         AddOn* personnelAddOn = getAddOn()->clone();
00011         personnelAddOn->setEntity(v);
00012         return personnelAddOn;
00013     } else {
00014         return v;
00015     }
00016 }
```

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

- VehicleFactory.h
- VehicleFactory.cpp

4.32 WarEngine Class Reference

```
#include <WarEngine.h>
```

Public Member Functions

- [WarEngineMemento](#) * [saveState](#) ()
Captures current state of simulation via member variables and creates [WarEngineMemento](#) instance storing all relevant members in [WarEngineState](#).
- void [loadSave](#) ([WarEngineMemento](#) *save)
Takes in an instance of saved [WarEngine](#) states and sets current instance's member variables to memento state.
- void [setWarTheatre](#) ([WarTheatre](#) *battleGround)
Sets the state's area to passed in battleGround parameter.
- void [addAlliance](#) ([Alliance](#) *alliance)
- void [simulate](#) ()
Simulates the war.

Static Public Member Functions

- static [WarEngine](#) & [getInstance](#) ()
Function that returns a reference to the current (and only) instance of the class.

Protected Member Functions

- [WarEngine](#) ()
Constructor for class. Is responsible for ensuring only a single instance of class exists.
- [WarEngine](#) (const [WarEngine](#) &)
Parameterized constructor for class.
- [WarEngine](#) & [operator=](#) (const [WarEngine](#) &)
Overloaded operator = for class.
- [~WarEngine](#) ()
Destructor for class responsible for freeing all allocated memory.

4.32.1 Detailed Description

Class that contains all information regarding current simulation. Only one instance of class is allowed.

Definition at line 12 of file [WarEngine.h](#).

4.32.2 Constructor & Destructor Documentation

4.32.2.1 WarEngine() [1/2]

```
WarEngine::WarEngine ( ) [protected]
```

Constructor for class. Is responsible for ensuring only a single instance of class exists.

Definition at line 5 of file [WarEngine.cpp](#).

```
00005 {
00006     this->state = new WarEngineState();
00007 }
```

4.32.2.2 WarEngine() [2/2]

```
WarEngine::WarEngine (
    const WarEngine & ) [inline], [protected]
```

Parameterized constructor for class.

Parameters

<i>warEngine&</i>	An anonymous warEngine reference.
-----------------------	-----------------------------------

Postconditions:

- parameter must be of type [WarEngine&](#)

Definition at line 33 of file [WarEngine.h](#).

```
00033 {};
```

4.32.2.3 ~WarEngine()

```
WarEngine::~WarEngine ( ) [protected]
```

Destructor for class responsible for freeing all allocated memory.

Definition at line 23 of file [WarEngine.cpp](#).

```
00023 {
00024     delete this->state;
00025 }
```

4.32.3 Member Function Documentation

4.32.3.1 addAlliance()

```
void WarEngine::addAlliance (
    Alliance * alliance )
```

Definition at line 87 of file [WarEngine.cpp](#).

```
00087 {
00088     state->alliances.push_back(alliance);
00089 }
```

4.32.3.2 getInstance()

```
WarEngine & WarEngine::getInstance ( ) [static]
```

Function that returns a reference to the current (and only) instance of the class.

Returns

[WarEngine&](#)

Definition at line 18 of file [WarEngine.cpp](#).

```
00018 {
00019     static WarEngine uniqueInstance_;
00020     return uniqueInstance_;
00021 }
```

4.32.3.3 loadSave()

```
void WarEngine::loadSave (
    WarEngineMemento * save )
```

Takes in an instance of saved [WarEngine](#) states and sets current instance's member variables to memento state.

Preconditions:

- Save must be of type [WarEngineMemento*](#)

Postconditions:

- Sets the instance of the class's state member variable to the passed in save parameter.

Parameters

save	must be a WarEngineMemento*
------	---------------------------------------------

Returns

void

Definition at line 13 of file [WarEngine.cpp](#).

```

00013                                     {
00014     delete this->state;
00015     this->state = save->getState();
00016 }
```

4.32.3.4 operator=()

```

WarEngine & WarEngine::operator= (
    const WarEngine & ) [inline], [protected]
```

Overloaded operator = for class.

Definition at line 38 of file [WarEngine.h](#).

```

00038 { return *this; };
```

4.32.3.5 saveState()

```

WarEngineMemento * WarEngine::saveState ( )
```

Captures current state of simulation via member variables and creates [WarEngineMemento](#) instance storing all relevant members in [WarEngineState](#).

Returns

WarEngineMemento*

Definition at line 9 of file [WarEngine.cpp](#).

```

00009                                     {
00010     return new WarEngineMemento(state->clone());
00011 }
```

4.32.3.6 setWarTheatre()

```

void WarEngine::setWarTheatre (
    WarTheatre * battleGround )
```

Sets the state's area to passed in battleGround parameter.

Parameters

<i>battleGround</i>	must be a WarTheatre*
---------------------	-----------------------

Preconditions:

- battleGround must be of type WarTheatre*

Postconditions:

- sets area in WarEngineState to passed in WarTheatre.

Returns

void

Definition at line 83 of file WarEngine.cpp.

```
00083                                     {
00084     state->setArea(battleGround);
00085 }
```

4.32.3.7 simulate()

```
void WarEngine::simulate ( )
```

Simulates the war.

PostConditions:

- Will simulate the war by running the engine

Returns

void

Definition at line 28 of file WarEngine.cpp.

```
00028     {
00029
00030     vector<Alliance*> alliances = this->state->getAlliances();
00031     int numAlliances = alliances.size();
00032     while (numAlliances > 1) {
00033         numAlliances = 0;
00034         for(int i = 0; i < alliances.size(); i++) {
00035             if (alliances[i]->getActive() == 1) {
00036                 numAlliances++;
00037                 RoundStats::clearStats();
00038                 state->getArea()->simulateBattle(alliances[i]);
00039
00040                 double percLoss = (RoundStats::numLosingPoints * 1.0) / (RoundStats::numLosingPoints +
RoundStats::numContestedPoints + RoundStats::numWinningPoints);
00041                 if (percLoss >= 0.7) {
00042                     alliances[i]->surrender();
00043                 } else if (percLoss >= 0.6) {
00044                     alliances[i]->offerPeace();
00045                 }
00046
00047                 cout << "===== " <<
endl << endl;
00048                 cout << "Alliance " << alliances[i]->getID() << ":" << endl;
00049
00050                 if (alliances[i]->getActive() == 2) {
00051                     cout << "Status: Surrendered" << endl;
00052                 } else if (alliances[i]->getActive() == 3) {
00053                     cout << "Status: Found Peace" << endl;
00054                 }
00055             }
00056         }
00057     }
00058 }
```

```

00054         } else {
00055             cout << "Status: Active" << endl;
00056         }
00057
00058         cout << RoundStats::toString() << endl;
00059         cout << "===== " <<
endl;
00060     }
00061 }
00062 }
00063
00064     for(int i = 0; i < alliances.size(); i++) {
00065         cout << "===== " << endl <<
endl;
00066         cout << "Alliance " << alliances[i]->getID() << ":" << endl;
00067
00068         if (alliances[i]->getActive() == 2) {
00069             cout << "Status: Surrendered" << endl;
00070         } else if (alliances[i]->getActive() == 3) {
00071             cout << "Status: Found Peace" << endl;
00072         } else {
00073             cout << "Status: Winner" << endl;
00074         }
00075
00076         cout << endl << "===== " <<
endl;
00077     }
00078 }
00079     cout << "SIMULATION COMPLETE!" << endl;
00080
00081 }

```

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

- WarEngine.h
- WarEngine.cpp

4.33 WarEngineMemento Class Reference

```
#include <WarEngineMemento.h>
```

Friends

- class [WarEngine](#)

4.33.1 Detailed Description

Class that encapsulates and externalises [WarEngine](#) State.

Definition at line 15 of file [WarEngineMemento.h](#).

4.33.2 Friends And Related Function Documentation

4.33.2.1 WarEngine

```
friend class WarEngine [friend]
```

Definition at line 17 of file [WarEngineMemento.h](#).

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

- WarEngineMemento.h
- WarEngineMemento.cpp

4.34 WarEngineState Class Reference

Class for storing current state of entire simulation.

```
#include <WarEngineState.h>
```

Public Member Functions

- [~WarEngineState](#) ()
Destructor for class.

Protected Member Functions

- [WarEngineState](#) ()
Initializes an instance of the [WarEngineState](#) class.
- void [setArea](#) ([Area](#) *area)
Takes in a vector of [Area](#) and sets it to the areas member of the [WarEngineState](#) instance.
- [Area](#) * [getArea](#) ()
Returns the member variable area.
- void [setAlliances](#) (vector< [Alliance](#) * > alliances)
Sets the given vector of [Alliance](#) object pointers to the alliances member variable.
- vector< [Alliance](#) * > [getAlliances](#) ()
Returns the alliances member variable.
- [WarEngineState](#) * [clone](#) ()
Returns a clone of the current [WarEngineMemento](#) object.

Friends

- class [WarEngine](#)

4.34.1 Detailed Description

Class for storing current state of entire simulation.

Class contains member variables areas which stores a vector of all war theatres and keypoints as well as a vector of all alliances in current simulation.

Definition at line 17 of file [WarEngineState.h](#).

4.34.2 Constructor & Destructor Documentation

4.34.2.1 WarEngineState()

```
WarEngineState::WarEngineState ( ) [protected]
```

Initializes an instance of the [WarEngineState](#) class.

Definition at line 3 of file [WarEngineState.cpp](#).

```
00003         {
00004     area = nullptr;
00005 }
```

4.34.2.2 ~WarEngineState()

```
WarEngineState::~~WarEngineState ( )
```

Destructor for class.

Definition at line 48 of file [WarEngineState.cpp](#).

```
00048     {
00049
00050     for(Alliance* alliance : this->alliances){
00051         delete alliance;
00052     }
00053
00054     delete this->area;
00055 }
```

4.34.3 Member Function Documentation

4.34.3.1 clone()

```
WarEngineState * WarEngineState::clone ( ) [protected]
```

Returns a clone of the current [WarEngineMemento](#) object.

Returns

[WarEngineState](#)*

Definition at line 31 of file [WarEngineState.cpp](#).

```
00031     {
00032
00033     WarEngineState* clonedState = new WarEngineState();
00034
00035     clonedState->setArea( this->area->clone() );
00036
00037     for(Alliance* alliance : this->alliances){
00038
00039         Alliance* clonedAlliance = alliance->clone();
00040
00041         clonedState->alliances.push_back(alliance);
00042
00043     }
00044
00045     return clonedState;
00046 }
```

4.34.3.2 getAlliances()

```
vector< Alliance * > WarEngineState::getAlliances ( ) [protected]
```

Returns the alliances member variable.

Returns

vector <Alliance*>

Exceptions

<code>out_of_range</code>	save archive is empty
---------------------------	-----------------------

Definition at line 23 of file [WarEngineState.cpp](#).

```
00023 {
00024
00025     if(alliances.size() == 0)
00026         std::__throw_out_of_range("No Alliances stored.");
00027
00028     return alliances;
00029 }
```

4.34.3.3 getArea()

```
Area * WarEngineState::getArea ( ) [protected]
```

Returns the member variable area.

Postconditions:

- Retrurns the area stored in the state

Returns

Area*

Definition at line 11 of file [WarEngineState.cpp](#).

```
00011 {
00012
00013     if(area == nullptr)
00014         throw "No Areas Stored.";
00015
00016     return this->area;
00017 }
```

4.34.3.4 setAlliances()

```
void WarEngineState::setAlliances (
    vector< Alliance * > alliances ) [protected]
```

Sets the given vector of [Alliance](#) object pointers to the alliances member variable.

Parameters

<code>vector<Alliance*></code>	<code>alliances</code>
--------------------------------------	------------------------

Preconditions:

- alliances must be a vector of Alliance*

Postconditions:

- Sets the instance's alliances member variable to the passed in parameter.

Returns

void

Definition at line 19 of file [WarEngineState.cpp](#).

```
00019                                     {
00020     this->alliances = alliances;
00021 }
```

4.34.3.5 setArea()

```
void WarEngineState::setArea (
    Area * area ) [protected]
```

Takes in a vector of [Area](#) and sets it to the areas member of the [WarEngineState](#) instance.

Preconditions:

- area must be of type Area*

Postconditions:

- Sets the [WarEngineState](#) area member variable to the passed in parameter.

Parameters

<code>area</code>	<code>must be an Area*</code>
-------------------	-------------------------------

Returns

void

Definition at line 7 of file [WarEngineState.cpp](#).

```
00007                                     {
00008     this->area = area;
00009 }
```

4.34.4 Friends And Related Function Documentation

4.34.4.1 WarEngine

```
friend class WarEngine [friend]
```

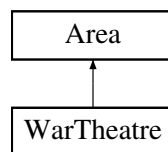
Definition at line 18 of file [WarEngineState.h](#).

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

- WarEngineState.h
- WarEngineState.cpp

4.35 WarTheatre Class Reference

Inheritance diagram for WarTheatre:



Public Member Functions

- [WarTheatre](#) (std::string areaName)
Instantiates the war theatre.
- [~WarTheatre](#) ()
Destroys the war theatre object.
- bool [isKeyPoint](#) ()
Returns area type.
- void [simulateBattle](#) ([Alliance](#) *alliance)
Simulate Battle with troops from the alliance passed in.
- void [addArea](#) ([Area](#) *area)
Adds an area to the war theatre object.
- void [addGeneral](#) ([General](#) *general)
Adds a general to all the points held by the [WarTheatre](#).
- [WarTheatre](#) * [clone](#) ()
Instantiates and returns a clone of the current war theatre.

4.35.1 Detailed Description

Definition at line 10 of file [WarTheatre.h](#).

4.35.2 Constructor & Destructor Documentation

4.35.2.1 WarTheatre()

```
WarTheatre::WarTheatre (
    std::string areaName )
```

Instantiates the war theatre.

Definition at line 5 of file [WarTheatre.cpp](#).

```
00005 :   Area(areaName) {}
```

4.35.2.2 ~WarTheatre()

```
WarTheatre::~~WarTheatre ( )
```

Destroys the war theatre object.

Postconditions:

- All dynamic memory should be deallocated from the war theatre object

Definition at line 7 of file [WarTheatre.cpp](#).

```
00007         {
00008     for (int i = 0; i < areas.size(); i++)
00009         delete areas[i];
00010 }
```

4.35.3 Member Function Documentation

4.35.3.1 addArea()

```
void WarTheatre::addArea (
    Area * area )
```

Adds an area to the war theatre object.

Preconditions:

- area must be an Area*

Postconditions:

- Add area to war theatre object

Parameters

<i>area</i>	must be an Area*
-------------	------------------

Returns

void

Definition at line 21 of file [WarTheatre.cpp](#).

```
00021 {
00022     areas.push_back(area);
00023 }
```

4.35.3.2 addGeneral()

```
void WarTheatre::addGeneral (
    General * general ) [virtual]
```

Adds a general to all the points held by the [WarTheatre](#).

Precoditions:

- general must be a General*

Postconditions:

- Add general to all points

Parameters

<i>general</i>	must be a General*
----------------	--------------------

Implements [Area](#).

Definition at line 34 of file [WarTheatre.cpp](#).

```
00034 {
00035     for (int i = 0; i < areas.size(); i++)
00036         areas[i]->addGeneral(general);
00037 }
```

4.35.3.3 clone()

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

Instantiates and returns a clone of the current war theatre.

Postconditions:

- Returns the clone of the current war theatre

Returns

WarTheatre* The war theatre clone

Implements [Area](#).

Definition at line 25 of file [WarTheatre.cpp](#).

```
00025     {
00026         WarTheatre* w = new WarTheatre(getAreaName());
00027
00028         for (int i = 0; i < areas.size(); i++)
00029             w->addArea(areas[i]->clone());
00030
00031         return w;
00032     }
```

4.35.3.4 isKeyPoint()

```
bool WarTheatre::isKeyPoint ( ) [virtual]
```

Returns area type.

Postconditions:

- Returns false

Returns

bool The area type

Implements [Area](#).

Definition at line 12 of file [WarTheatre.cpp](#).

```
00012     {
00013         return false;
00014     }
```

4.35.3.5 simulateBattle()

```
void WarTheatre::simulateBattle (
    Alliance * alliance ) [virtual]
```

Simulate Battle with troops from the alliance passed in.

Preconditions:

- alliance must be an Alliance*

Postconditions:

- Call attacks function of areas

Parameters

<i>alliance</i>	must be an Alliance*
-----------------	----------------------

Returns

void

Implements [Area](#).

Definition at line 16 of file [WarTheatre.cpp](#).

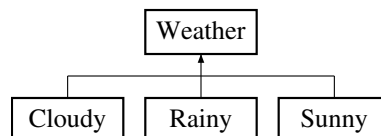
```
00016 {
00017     for (int i = 0; i < areas.size(); i++)
00018         areas[i]->simulateBattle(alliance);
00019 }
```

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

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

4.36 Weather Class Reference

Inheritance diagram for Weather:



Public Member Functions

- [Weather](#) ()
Instantiates the [Weather](#) object.
- [~Weather](#) ()
Destructor for the [Weather](#) object.
- double [getMultiplier](#) ()
Returns double which shows the weather multiplier.
- virtual void [handleChange](#) ([KeyPoint](#) *keypoint)=0
Will change the current state of the weather inside the specific keypoint.
- virtual std::string [getWeather](#) ()=0
Returns a string which tells us the weather.
- virtual [Weather](#) * [clone](#) ()=0
Returns a clone of the [Weather](#) object.

Protected Attributes

- double [multiplier](#)

4.36.1 Detailed Description

Definition at line 9 of file [Weather.h](#).

4.36.2 Constructor & Destructor Documentation

4.36.2.1 Weather()

```
Weather::Weather ( )
```

Instantiates the [Weather](#) object.

Definition at line 3 of file [Weather.cpp](#).
00003 {}

4.36.2.2 ~Weather()

```
Weather::~Weather ( )
```

Destructor for the [Weather](#) object.

Definition at line 5 of file [Weather.cpp](#).
00005 {}

4.36.3 Member Function Documentation

4.36.3.1 clone()

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

Returns a clone of the [Weather](#) object.

Returns

[Weather](#)* Clone of [Weather](#) object

Implemented in [Cloudy](#), [Rainy](#), and [Sunny](#).

4.36.3.2 getMultiplier()

```
double Weather::getMultiplier ( )
```

Returns double which shows the weather multiplier.

Postconditions:

- Returns the double multiplier

Returns

double which is the multiplier

Definition at line 7 of file [Weather.cpp](#).

```
00007 {  
00008     return this->multiplier;  
00009 }
```

4.36.3.3 getWeather()

```
virtual std::string Weather::getWeather ( ) [pure virtual]
```

Returns a string which tells us the weather.

Postconditions:

- Returns the wether of the current state

Returns

std::string which is the current state

Implemented in [Cloudy](#), [Rainy](#), and [Sunny](#).

4.36.3.4 handleChange()

```
virtual void Weather::handleChange (  
    KeyPoint * keypoint ) [pure virtual]
```

Will change the current state of the weather inside the specific keypoint.

Preconditions:

- keypoint must be a KeyPoint*

Postconditions:

- Changes the current weather to the next one in the state pattern ([Cloudy](#))

Parameters

<i>keypoint</i>	must be a KeyPoint*
-----------------	---------------------

Returns

void

Implemented in [Cloudy](#), [Rainy](#), and [Sunny](#).

4.36.4 Member Data Documentation

4.36.4.1 multiplier

```
double Weather::multiplier [protected]
```

Definition at line 12 of file [Weather.h](#).

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

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

Chapter 5

File Documentation

5.1 AddOn.cpp

```
00001 #include "AddOn.h"
00002 #include <stdexcept>
00003
00004 AddOn::AddOn(int value): Entity() {
00005     this->value = value;
00006     entity = NULL;
00007 }
00008
00009 void AddOn::setValue(int value) {
00010
00011     if (value <= 0)
00012         throw std::invalid_argument("value must be greater than zero");
00013
00014     this->value = value;
00015 }
00016
00017 int AddOn::getValue() {
00018     return value;
00019 }
00020
00021 void AddOn::setEntity(Entity* entity) {
00022     this->entity = entity;
00023 }
00024
00025 Entity* AddOn::getEntity() {
00026     return this->entity;
00027 }
00028
00029 Type* AddOn::getType() {
00030     return entity->getType();
00031 }
00032
00033 void AddOn::setType(Type* type) {
00034     entity->setType(type);
00035 }
00036
00037 Alliance* AddOn::getAlliance() {
00038     return entity->getAlliance();
00039 }
00040
00041 void AddOn::setAlliance(Alliance* alliance) {
00042     entity->setAlliance(alliance);
00043 }
00044
00045 int AddOn::getHealth() {
00046     return entity->getHealth();
00047 }
00048
00049 void AddOn::setHealth(int health) {
00050     entity->setHealth(health);
00051 }
00052
00053 int AddOn::getDamage() {
00054     return entity->getDamage();
00055 }
00056
00057 void AddOn::setDamage(int damage) {
00058     entity->setDamage(damage);
00059 }
```

5.2 AddOn.h

```

00001 #ifndef ADDON_H
00002 #define ADDON_H
00003 #include "Entity.h"
00004
00010 class AddOn : public Entity {
00011
00012 protected:
00013     int value;
00014     Entity* entity;
00015
00016 public:
00022     AddOn(int value);
00023
00036     void setValue(int value);
00037
00046     int getValue();
00047
00060     void setEntity(Entity* entity);
00061
00070     Entity* getEntity();
00071
00080     Type* getType();
00081
00094     void setType(Type* type);
00095
00104     Alliance* getAlliance();
00105
00118     void setAlliance(Alliance* alliance);
00119
00128     int getHealth();
00129
00142     void setHealth(int health);
00143
00152     int getDamage();
00153
00166     void setDamage(int damage);
00167
00180     virtual void takeDamage(int damage) = 0;
00181
00194     virtual void dealDamage(Entity* entity) = 0;
00195
00204     virtual AddOn* clone() = 0;
00205 };
00206
00207 #endif

```

5.3 AddOnTest.h

```

00001 #include <stdexcept>
00002 #include "AddOn.h"
00003 #include "Piercing.h"
00004 #include "Armour.h"
00005 #include "Personnel.h"
00006 #include "TerrainType.h"
00007 #include "gtest/gtest.h"
00008
00009 namespace {
00010
00011     // Tests General AddOn Functionality
00012
00013     // Tests AddOn setValue()
00014     // ===== Precondition Testing =====
00015     // Test Precondition Negative
00016     TEST(AddOnSetValueTest, TestPreconditionNegative) {
00017         Armour* a = new Armour(10);
00018         try {
00019             a->setValue(-5);
00020             FAIL();
00021         } catch (std::invalid_argument& err) {
00022             EXPECT_EQ(err.what(), std::string("value must be greater than zero"));
00023         } catch (...) {
00024             FAIL();
00025         }
00026     }
00027
00028     // Test Precondition Positive
00029     TEST(AddOnSetValueTest, TestPreconditionPositive) {
00030         Armour* a = new Armour(5);
00031         a->setValue(5);
00032         EXPECT_EQ(5, a->getValue());
00033     }

```

```

00034
00035 // Test Precondition Bounds
00036 TEST(AddOnSetValueTest, TestPreconditionBounds) {
00037     Armour* a = new Armour(5);
00038     try {
00039         a->setValue(0);
00040         FAIL();
00041     } catch (std::invalid_argument& err) {
00042         EXPECT_EQ(err.what(), std::string("value must be greater than zero"));
00043     } catch (...) {
00044         FAIL();
00045     }
00046 }
00047
00048 // ===== Positive Testing =====
00049 // Test Preconditions Bounds
00050 TEST(AddOnSetValueTest, PositiveTesting) {
00051     Armour* a = new Armour(5);
00052
00053     a->setValue(5);
00054     EXPECT_EQ(5, a->getValue());
00055
00056     a->setValue(10);
00057     EXPECT_EQ(10, a->getValue());
00058
00059     a->setValue(20);
00060     EXPECT_EQ(20, a->getValue());
00061
00062     a->setValue(55);
00063     EXPECT_EQ(55, a->getValue());
00064
00065     a->setValue(3);
00066     EXPECT_EQ(3, a->getValue());
00067
00068     a->setValue(100);
00069     EXPECT_EQ(100, a->getValue());
00070 }
00071
00072 // Tests AddOn setEntity()
00073 // ===== Positive Testing =====
00074 // Test Preconditions Bounds
00075 TEST(AddOnSetEntityTest, PositiveTesting) {
00076     Armour* a = new Armour(5);
00077
00078     Personnel* p = new Personnel(new TerrainType(), 100, 10);
00079     a->setEntity(p);
00080     EXPECT_EQ(p, a->getEntity());
00081
00082     Personnel* m = new Personnel(new TerrainType(), 100, 10);
00083     a->setEntity(m);
00084     EXPECT_EQ(m, a->getEntity());
00085
00086     Personnel* n = new Personnel(new TerrainType(), 100, 10);
00087     a->setEntity(n);
00088     EXPECT_EQ(n, a->getEntity());
00089 }
00090
00091 // Tests Armour AddOn Functionality
00092
00093 // Tests Armour takeDamage()
00094 // ===== Precondition Testing =====
00095 // Test Precondition Negative
00096 TEST(ArmourTakeDamageTest, TestPreconditionNegative) {
00097     Armour* a = new Armour(10);
00098     try {
00099         a->takeDamage(-5);
00100         FAIL();
00101     } catch (std::invalid_argument& err) {
00102         EXPECT_EQ(err.what(), std::string("damage must be greater than zero"));
00103     } catch (...) {
00104         FAIL();
00105     }
00106 }
00107
00108 // Test Precondition Positive
00109 TEST(ArmourTakeDamageTest, TestPreconditionPositive) {
00110     Armour* a = new Armour(10);
00111     Personnel* p = new Personnel(new TerrainType(), 100, 10);
00112
00113     a->setEntity(p);
00114     a->takeDamage(10);
00115     EXPECT_EQ(0, a->getValue());
00116     EXPECT_EQ(100, a->getHealth());
00117     EXPECT_EQ(100, p->getHealth());
00118 }
00119
00120 // Test Precondition Bounds

```

```

00121     TEST(ArmourTakeDamageTest, TestPreconditionBounds) {
00122         Armour* a = new Armour(10);
00123         try {
00124             a->takeDamage(0);
00125             FAIL();
00126         } catch (std::invalid_argument& err) {
00127             EXPECT_EQ(err.what(), std::string("damage must be greater than zero"));
00128         } catch (...) {
00129             FAIL();
00130         }
00131     }
00132
00133     // ===== Positive Testing =====
00134     // Test Preconditions Bounds
00135     TEST(ArmourTakeDamageTest, PositiveTesting) {
00136         Armour* a = new Armour(20);
00137         Personnel* p = new Personnel(new TerrainType(), 100, 10);
00138         a->setEntity(p);
00139
00140         a->takeDamage(10);
00141         EXPECT_EQ(10, a->getValue());
00142         EXPECT_EQ(100, a->getHealth());
00143         EXPECT_EQ(100, p->getHealth());
00144
00145         a->takeDamage(10);
00146         EXPECT_EQ(0, a->getValue());
00147         EXPECT_EQ(100, a->getHealth());
00148         EXPECT_EQ(100, p->getHealth());
00149
00150         a->takeDamage(10);
00151         EXPECT_EQ(0, a->getValue());
00152         EXPECT_EQ(90, a->getHealth());
00153         EXPECT_EQ(90, p->getHealth());
00154     }
00155
00156     // Tests Armour dealDamage()
00157     // ===== Positive Testing =====
00158     // Test Preconditions Bounds
00159     TEST(ArmourDealDamageTest, PositiveTesting) {
00160         Armour* a = new Armour(10);
00161         Personnel* p = new Personnel(new TerrainType(), 100, 10);
00162         a->setEntity(p);
00163         Personnel* x = new Personnel(new TerrainType(), 100, 10);
00164
00165         a->dealDamage(x);
00166         EXPECT_EQ(10, a->getValue());
00167         EXPECT_EQ(90, x->getHealth());
00168
00169         a->dealDamage(x);
00170         EXPECT_EQ(10, a->getValue());
00171         EXPECT_EQ(80, x->getHealth());
00172
00173         a->dealDamage(x);
00174         EXPECT_EQ(10, a->getValue());
00175         EXPECT_EQ(70, x->getHealth());
00176     }
00177
00178     // Tests Piercing AddOn Functionality
00179
00180     // Tests Piercing takeDamage()
00181     // ===== Positive Testing =====
00182     // Test Preconditions Bounds
00183     TEST(PiercingTakeDamageTest, PositiveTesting) {
00184         Piercing* pi = new Piercing(10);
00185         Personnel* p = new Personnel(new TerrainType(), 100, 10);
00186         pi->setEntity(p);
00187
00188         pi->takeDamage(10);
00189         EXPECT_EQ(10, pi->getValue());
00190         EXPECT_EQ(90, pi->getHealth());
00191         EXPECT_EQ(90, p->getHealth());
00192
00193         pi->takeDamage(10);
00194         EXPECT_EQ(10, pi->getValue());
00195         EXPECT_EQ(80, pi->getHealth());
00196         EXPECT_EQ(80, p->getHealth());
00197
00198         pi->takeDamage(10);
00199         EXPECT_EQ(10, pi->getValue());
00200         EXPECT_EQ(70, pi->getHealth());
00201         EXPECT_EQ(70, p->getHealth());
00202     }
00203
00204     // Tests Piercing dealDamage()
00205     // ===== Positive Testing =====
00206     // Test Preconditions Bounds
00207     TEST(PiercingDealDamageTest, PositiveTesting) {

```



```

00208     Piercing* pi = new Piercing(10);
00209     Personnel* p = new Personnel(new TerrainType(), 100, 10);
00210     pi->setEntity(p);
00211     Personnel* x = new Personnel(new TerrainType(), 100, 10);
00212
00213     pi->dealDamage(x);
00214     EXPECT_EQ(10, pi->getValue());
00215     EXPECT_EQ(80, x->getHealth());
00216
00217     pi->dealDamage(x);
00218     EXPECT_EQ(10, pi->getValue());
00219     EXPECT_EQ(60, x->getHealth());
00220
00221     pi->dealDamage(x);
00222     EXPECT_EQ(10, pi->getValue());
00223     EXPECT_EQ(40, x->getHealth());
00224 }
00225 }

```

5.4 AerialType.cpp

```

00001 #include "AerialType.h"
00002
00003 AerialType::AerialType() {}
00004
00005 string AerialType::getTypeDesc() {
00006     return "Aerial";
00007 }
00008
00009 Type* AerialType::clone() {
00010     return new AerialType();
00011 }

```

5.5 AerialType.h

```

00001 #ifndef AERIALTYPE_H
00002 #define AERIALTYPE_H
00003
00004 #include "Type.h"
00005
00011 class AerialType : public Type {
00012
00013 public:
00017     AerialType();
00018
00027     string getTypeDesc();
00028
00029
00038     Type* clone();
00039 };
00040
00041 #endif

```

5.6 Aggressive.cpp

```

00001 #include "Aggressive.h"
00002 #include "KeyPoint.h"
00003
00004 Aggressive::Aggressive() {}
00005
00006 void Aggressive::performStrat(KeyPoint* keyPoint, Alliance* alliance) {
00007     int randomNumber = (rand() % 10) + 5;
00008     keyPoint->moveEntitiesInto(alliance, randomNumber);
00009 }
00010
00011 Strategy* Aggressive::clone() {
00012     return new Aggressive();
00013 }

```

5.7 Aggressive.h

```

00001 #ifndef AGGRESSIVE_H
00002 #define AGGRESSIVE_H
00003 #include "Strategy.h"
00004
00005 class Aggressive : public Strategy {
00006
00007 public:
00008     Aggressive();
00009
00025     void performStrat(KeyPoint* keyPoint, Alliance* alliance);
00026
00032     Strategy* clone();
00033 };
00034
00035 #endif

```

5.8 Alliance.cpp

```

00001 #include "Alliance.h"
00002 #include "Negotiator.h"
00003 #include "Entity.h"
00004 #include "RoundStats.h"
00005 #include <time.h>
00006 #include <iostream>
00007
00008 using namespace std;
00009
00010 int Alliance::totalNum = 0;
00011
00012 Alliance::Alliance() {
00013     this->active = 1;
00014     this->aID = totalNum++;
00015     this->negotiator = NULL;
00016     srand(time(0));
00017 }
00018
00019 Alliance::Alliance(Alliance& alliance) {
00020     this->active = alliance.active;
00021     this->aID = alliance.aID;
00022
00023     for (int i = 0; i < alliance.members.size(); i++)
00024         this->addCountry(alliance.members[i]->clone());
00025
00026     for (int i = 0; i < alliance.production.size(); i++)
00027         this->addFactory(alliance.production[i]->clone());
00028
00029     for (int i = 0; i < alliance.reserveEntities.size(); i++)
00030         this->addReserveEntity(alliance.reserveEntities[i]->clone());
00031
00032     this->negotiator = NULL;
00033 }
00034
00035 Alliance::~Alliance() {
00036
00037     for (int i = 0; i < members.size(); i++)
00038         //delete members[i];
00039
00040     if (this->negotiator != NULL) {
00041         this->negotiator->removeAlliance(this);
00042
00043         if (this->negotiator->getNumAlliances() == 1)
00044             delete this->negotiator;
00045     }
00046 }
00047
00048 void Alliance::setNegotiator(Negotiator* negotiator) {
00049     this->negotiator = negotiator;
00050 }
00051
00052 void Alliance::addCountry(Country* nation) {
00053     members.push_back(nation);
00054 }
00055
00056 vector<Entity*> Alliance::getReserveEntities(int number) {
00057     vector<Entity*> out;
00058     for (int i = 0; i < number && i < reserveEntities.size(); i++) {
00059         out.push_back(reserveEntities[i]);
00060         reserveEntities.erase(reserveEntities.begin() + i);
00061     }
00062
00063     return out;

```

```

00064 }
00065
00066 void Alliance::addReserveEntity(Entity* entity) {
00067     reserveEntities.push_back(entity);
00068 }
00069
00070 int Alliance::numRemainingEntities() {
00071     return reserveEntities.size();
00072 }
00073
00074 bool Alliance::considerPeace() {
00075     return (rand() % 2 == 0);
00076 }
00077
00078 void Alliance::addFactory(Factory* factory) {
00079     production.push_back(factory);
00080 }
00081
00082 void Alliance::runFactories() {
00083     for (int i = 0; i < production.size(); i++) {
00084         RoundStats::numEntitiesCreated++;
00085         reserveEntities.push_back(production[i]->createEntity(this));
00086     }
00087 }
00088
00089 void Alliance::surrender() {
00090     this->active = 2; //Number 2 means that Alliance has surrendered
00091
00092     this->negotiator->removeAlliance(this);
00093 }
00094
00095 int Alliance::getID() {
00096     return this->aID;
00097 }
00098
00099 bool Alliance::offerPeace() {
00100
00101     if (this->negotiator->sendPeace(this)) //Send the peace deal to all the alliances fighting against
00102     {
00103         this->active = 3; //Number 3 means that Alliance chose to peacefully pull out of war
00104         return true;
00105     }
00106
00107     return false;
00108 }
00109
00110 int Alliance::getActive() {
00111     return active;
00112 }
00113
00114 Alliance* Alliance::clone() {
00115     return new Alliance(*this);
00116 }

```

5.9 Alliance.h

```

00001 #ifndef ALLIANCE_H
00002 #define ALLIANCE_H
00003 #include "Country.h"
00004 #include "Factory.h"
00005 #include "Country.h"
00006 #include <vector>
00007
00008 class Negotiator;
00009 class Entity;
00010
00011 using namespace std;
00012
00013 class Alliance {
00014 private:
00015     static int totalNum;
00016     int aID;
00017     vector<Factory*> production;
00018     Negotiator* negotiator;
00019     vector<Country*> members;
00020     int active;
00021     vector<Entity*> reserveEntities;
00022
00023 public:
00024     Alliance();
00025
00026     Alliance(Alliance& alliance);

```

```

00034
00038     ~Alliance();
00039
00052     void setNegotiator(Negotiator* newNegotiator);
00053
00066     void addCountry(Country* nation);
00067
00081     vector<Entity*> getReserveEntities(int number);
00082
00095     void addReserveEntity(Entity* entity);
00096
00108     bool considerPeace();
00109
00122     void addFactory(Factory* factory);
00123
00133     void surrender();
00134
00143     int getID();
00144
00153     bool offerPeace();
00154
00163     Alliance* clone();
00164
00176     void setActiveStatus(int active);
00177
00186     int getActive();
00187
00197     int numRemainingEntities();
00198
00207     void runFactories();
00208 };
00209
00210 #endif

```

5.10 AquaticType.cpp

```

00001 #include "AquaticType.h"
00002
00003 using namespace std;
00004
00005 AquaticType::AquaticType() {}
00006
00007 string AquaticType::getTypeDesc() {
00008     return "Aquatic";
00009 }
00010
00011 Type* AquaticType::clone() {
00012     return new AquaticType();
00013 }

```

5.11 AquaticType.h

```

00001 #ifndef AQUATICTYPE_H
00002 #define AQUATICTYPE_H
00003
00004 #include "Type.h"
00005
00011 class AquaticType : public Type {
00012 public:
00017     AquaticType();
00018
00027     string getTypeDesc();
00028
00037     Type* clone();
00038 };
00039
00040 #endif

```

5.12 Area.cpp

```

00001 #include "Area.h"
00002
00003 using namespace std;
00004

```

```

00005 Area::Area(string areaName) {
00006     this->areaName = areaName;
00007 }
00008
00009 Area::~Area() {}
00010
00011 std::string Area::getAreaName() const {
00012     return areaName;
00013 }

```

5.13 Area.h

```

00001 #ifndef AREA_H
00002 #define AREA_H
00003 #include <string>
00004 #include "Alliance.h"
00005
00006 class General;
00007
00008 class Area {
00009 private:
00010     std::string areaName;
00011 public:
00012     Area(std::string areaName);
00013     virtual ~Area();
00014     virtual bool isKeyPoint() = 0;
00015     virtual void simulateBattle(Alliance* alliance) = 0;
00016     std::string getAreaName() const;
00017     virtual Area* clone() = 0;
00018     virtual void addGeneral(General* general) = 0;
00019 };
00020 #endif

```

5.14 Armour.cpp

```

00001 #include "Armour.h"
00002 #include <stdexcept>
00003
00004 Armour::Armour(int value) : AddOn(value) {}
00005
00006 void Armour::takeDamage(int damage) {
00007     if (damage <= 0)
00008         throw std::invalid_argument("damage must be greater than zero");
00009     if (value > 0) {
00010         value -= damage;
00011     } else {
00012         entity->takeDamage(damage);
00013     }
00014 }
00015
00016 void Armour::dealDamage(Entity* entity) {
00017     this->entity->dealDamage(entity);
00018 }
00019
00020 AddOn* Armour::clone() {
00021     Armour* armour = new Armour(value);
00022     if (getEntity() != NULL)
00023         armour->setEntity(entity->clone());
00024     return armour;
00025 }

```

5.15 Armour.h

```

00001 #ifndef ARMOUR_H

```

```

00002 #define ARMOUR_H
00003 #include "AddOn.h"
00004 #include "Entity.h"
00005
00011 class Armour : public AddOn {
00012
00013
00014 public:
00020     Armour(int value);
00021
00037     void takeDamage(int damage);
00038
00051     void dealDamage(Entity* entity);
00052
00061     AddOn* clone();
00062 };
00063
00064 #endif

```

5.16 Cloudy.cpp

```

00001 #include "Cloudy.h"
00002 #include "Rainy.h"
00003
00004 Cloudy::Cloudy(): Weather() {
00005     this->multiplier = 0.75;
00006 }
00007
00008 std::string Cloudy::getWeather() {
00009     return "Cloudy";
00010 }
00011
00012 void Cloudy::handleChange(KeyPoint* k) {
00013     Rainy* newWeather = new Rainy();
00014     k->setWeather(newWeather);
00015 }
00016
00017 Weather* Cloudy::clone() {
00018     return new Cloudy();
00019 }

```

5.17 Cloudy.h

```

00001 #ifndef CLOUDY_H
00002 #define CLOUDY_H
00003 #include "Weather.h"
00004 #include <string>
00005
00006 class Cloudy : public Weather {
00007
00008 public:
00012     Cloudy();
00013
00022     std::string getWeather();
00023
00036     void handleChange(KeyPoint* keypoint);
00037
00043     Weather* clone();
00044 };
00045
00046 #endif

```

5.18 Country.cpp

```

00001 #include "Country.h"
00002
00003 using namespace std;
00004
00005 Country::Country(std::string name){
00006     this->name = name;
00007     this->id = rand() % 1000;
00008 }
00009
00010
00011 Country* Country::clone() {

```

```

00012         return new Country(this->name);
00013     }
00014
00015     string Country::getName() const{
00016         return this->name;
00017     }
00018
00019     int Country::getID() const{
00020         return this->id;
00021     }
00022
00023

```

5.19 Country.h

```

00001 #ifndef COUNTRY_H
00002 #define COUNTRY_H
00003 #include <string>
00004
00005 class Country {
00006
00007 private:
00008     std::string name;
00009     int id;
00010
00011 public:
00012     Country(std::string name);
00013
00014     Country* clone();
00015
00016     void setName(std::string name);
00017
00018     void setID(int id);
00019
00020     std::string getName() const;
00021
00022     int getID() const;
00023 };
00024
00025 #endif

```

5.20 Defensive.cpp

```

00001 #include "Defensive.h"
00002
00003 Defensive::Defensive() {
00004
00005 }
00006
00007 void Defensive::performStrat(KeyPoint* keyPoint, Alliance* alliance) {
00008
00009     int randomNumber = (rand() % 5) + 1;
00010     keyPoint->moveEntitiesInto(alliance, randomNumber);
00011 }
00012
00013 Strategy* Defensive::clone() {
00014     return new Defensive();
00015 }

```

5.21 Defensive.h

```

00001 #ifndef DEFENSIVE_H
00002 #define DEFENSIVE_H
00003 #include "Strategy.h"
00004 #include "KeyPoint.h"
00005 #include "Alliance.h"
00006 #include "Personnel.h"
00007 class Defensive : public Strategy {
00008
00009
00010 public:
00011     Defensive();
00012

```

```

00022     void performStrat(KeyPoint* keyPoint, Alliance* alliance);
00023
00029     Strategy* clone();
00030 };
00031
00032 #endif

```

5.22 EasySetup.cpp

```

00001 #include "EasySetup.h"
00002 #include <string.h>
00003 #include "Alliance.h"
00004 #include "Country.h"
00005 #include "AquaticType.h"
00006 #include "AerialType.h"
00007 #include "TerrainType.h"
00008 #include "Piercing.h"
00009 #include "Armour.h"
00010 #include "PersonnelFactory.h"
00011 #include "VehicleFactory.h"
00012 #include "SupportFactory.h"
00013 #include "KeyPoint.h"
00014 #include "WarTheatre.h"
00015 #include "Passive.h"
00016 #include "Aggressive.h"
00017 #include "Defensive.h"
00018 #include "WarEngine.h"
00019 #include "Negotiator.h"
00020
00021 EasySetup::EasySetup() {
00022     saveArchive = new SaveArchive();
00023 }
00024
00025 void EasySetup::setupSimulation() {
00026     while (true)
00027     {
00028         cout << "Load simulation (L) or New Simulation (N): ";
00029         string selectedOption;
00030         cin >> selectedOption;
00031         cin.ignore();
00032
00033         if(toupper(selectedOption[0]) == 'L')
00034         {
00035             string saveName;
00036             cout << "Please enter the name of the save to be re-simulated" << endl;
00037             getline(cin, saveName); // getting the name of the save-archive
00038             try {
00039                 this->loadSpecificSave(saveName); // loading the save-archive
00040                 return; // will return if the above the function does not throw an exception
00041             } catch(const std::exception& exception) {
00042                 cout << "Error: " << exception.what() << endl;
00043
00044                 if (strcmp(exception.what(), "Save archive is empty") == 0) {
00045                     cout << "Please create new simulation" << endl;
00046                     goto setup;
00047                 } else if (strcmp(exception.what(), "No save with given name exists") == 0) {
00048                     cout << "Please enter the correct name of save-archive and try again or create new
00049 simulation" << endl;
00050                 }
00051             }
00052         }
00053         } else if(toupper(selectedOption[0]) == 'N') {
00054             // setting up a new simulation
00055             goto setup;
00056         } else {
00057             cout << "Incorrect input: Please enter (L) or (N)" << endl;
00058         }
00059     }
00060 }
00061
00062 setup:
00063     // Creating alliances and generals
00064     int numAlliesAndGenerals;
00065     cout << "Enter number of alliances: ";
00066     cin >> numAlliesAndGenerals;
00067
00068     Alliance** alliances = new Alliance*[numAlliesAndGenerals];
00069     General** generals = new General*[numAlliesAndGenerals];
00070
00071     int numCountries,
00072         numFactories;
00073     string countryName,

```



```

00074         factoryType,
00075         selectedFactory,
00076         selectedAddOn;
00077     Country* country;
00078     Type* type;
00079     AddOn* addOn;
00080     Factory* factory;
00081
00082     Negotiator* negotiator = new Negotiator();
00083
00084     for (int i = 0; i < numAlliesAndGenerals; i++) {
00085         alliances[i] = new Alliance();
00086         negotiator->addAlliance(alliances[i]);
00087         alliances[i]->setNegotiator(negotiator);
00088         WarEngine::getInstance().addAlliance(alliances[i]);
00089
00090         cout << "Enter number of countries for Alliance " << alliances[i]->getID() << ": ";
00091         cin >> numCountries;
00092         cin.ignore();
00093
00094         for (int k = 0; k < numCountries; k++) {
00095             cout << "Enter name of county " << k+1 << ": ";
00096             getline(cin, countryName);
00097             country = new Country(countryName);
00098             alliances[i]->addCountry(country);
00099         }
00100
00101         cout << "Enter number of factories for Alliance " << alliances[i]->getID() << ": ";
00102         cin >> numFactories;
00103
00104         for (int k = 0; k < numFactories; k++) {
00105             retryType:
00106             cout << "Factory " << k+1 << " is of type Aquatic(Q), Aerial(E), or Terrain(T) : ";
00107             cin >> factoryType;
00108             cin.ignore();
00109
00110             if (toupper(factoryType[0]) == 'Q') {
00111                 type = new AerialType;
00112             } else if (toupper(factoryType[0]) == 'E') {
00113                 type = new AerialType;
00114             } else if (toupper(factoryType[0]) == 'T') {
00115                 type = new TerrainType;
00116             } else {
00117                 cout << "Invalid type input! Try again" << endl;
00118                 goto retryType;
00119             }
00120
00121             retryAddOn:
00122             cout << "Select AddOn for factory " << k+1 << " Armour(A), Piercing(P) or None(N) : ";
00123             getline(cin, selectedAddOn);
00124             if (toupper(selectedAddOn[0]) == 'A') {
00125                 int value;
00126                 cout << "Enter armour value: ";
00127                 cin >> value;
00128                 cin.ignore();
00129                 addOn = new Armour(value);
00130             } else if (toupper(selectedAddOn[0]) == 'P') {
00131                 int value;
00132                 cout << "Enter piercing value: ";
00133                 cin >> value;
00134                 cin.ignore();
00135                 addOn = new Piercing(value);
00136             } else if (toupper(selectedAddOn[0]) == 'N') {
00137                 addOn = NULL;
00138             } else {
00139                 cout << "Invalid AddOn input! Try again" << endl;
00140                 goto retryAddOn;
00141             }
00142
00143             retryFactory:
00144             cout << "Which factory is factory " << k+1 << " Vehicle(V), Personnel(P), or Support(S) : ";
00145             getline(cin, selectedFactory);
00146             if (toupper(selectedFactory[0]) == 'V') {
00147                 factory = new VehicleFactory(type, addOn);
00148             } else if (toupper(selectedFactory[0]) == 'P') {
00149                 factory = new PersonnelFactory(type, addOn);
00150             } else if (toupper(selectedFactory[0]) == 'S') {
00151                 factory = new SupportFactory(type, addOn);
00152             } else {
00153                 cout << "Invalid factory input! Try again" << endl;
00154                 goto retryFactory;
00155             }
00156
00157             alliances[i]->addFactory(factory);
00158         }
00159     }

```

```

00160         string selectedStrat;
00161         Strategy* strat;
00162
00163         retryStrat:
00164         cout << "What is this Alliances generals strategy Passive(P), Defensive(D), or
Aggressive(A) : ";
00165         getline(cin, selectedStrat);
00166         if (toupper(selectedStrat[0]) == 'P') {
00167             strat = new Passive();
00168         } else if (toupper(selectedStrat[0]) == 'D') {
00169             strat = new Defensive();
00170         } else if (toupper(selectedStrat[0]) == 'A') {
00171             strat = new Aggressive();
00172         } else {
00173             cout << "Invalid strategy input! Try again" << endl;
00174             goto retryStrat;
00175         }
00176
00177         generals[i] = new General(alliances[i], strat);
00178     }
00179
00180     int factoryRun;
00181     cout << "How many production runs do you wish to perform: ";
00182     cin >> factoryRun;
00183     cin.ignore();
00184     for (int i = 0; i < numAlliesAndGenerals; i++) {
00185         for (int j = 0; j < factoryRun; j++) {
00186             alliances[i]->runFactories();
00187         }
00188     }
00189
00190     // Creating main WarTheatre
00191     WarTheatre* mainBattleGround;
00192     cout << "Creating the main battle ground" << endl;
00193     string battleGroundName;
00194     cout << "Set main battle ground's name: ";
00195     getline(cin, battleGroundName);
00196     mainBattleGround = new WarTheatre(battleGroundName);
00197
00198     int sizeOfGrounds;
00199     cout << "Enter number of battle grounds in " << battleGroundName << " battle ground: ";
00200     cin >> sizeOfGrounds;
00201     cin.ignore();
00202     WarTheatre** battleGrounds = new WarTheatre*[sizeOfGrounds];
00203
00204     // Creating sub WarTheatres
00205     for (int i = 0; i < sizeOfGrounds; i++) {
00206         battleGroundName.clear();
00207         cout << "Set battle ground " << i+1 << "'s name: ";
00208         getline(cin, battleGroundName);
00209         battleGrounds[i] = new WarTheatre(battleGroundName);
00210     }
00211
00212     vector<int> numKeyPoints;
00213     int numKeyPoint = 0;
00214
00215     for (int i = 0; i < sizeOfGrounds; i++) {
00216         cout << "Enter number of key points in " << battleGrounds[i]->getAreaName() << " battle
ground: ";
00217         cin >> numKeyPoint;
00218         cin.ignore();
00219         numKeyPoints.push_back(numKeyPoint);
00220         numKeyPoint = 0;
00221     }
00222
00223     KeyPoint* keyPoint;
00224     string keyPointName;
00225     int numEntitiesInKeyPt;
00226
00227     // Creating KeyPoints for the sub WarTheatres
00228     for (int i = 0; i < sizeOfGrounds; i++) {
00229         numKeyPoint = numKeyPoints[i];
00230         cout << "For " << battleGrounds[i]->getAreaName() << "'s key points" << endl;
00231
00232         for (int k = 0; k < numKeyPoint; k++) {
00233             cout << "Set key point " << i+1 << "'s name: ";
00234             getline(cin, keyPointName);
00235             keyPoint = new KeyPoint(keyPointName);
00236
00237             for (int a = 0; a < numAlliesAndGenerals; a++) {
00238                 tryAgain:
00239                 cout << "There are " << alliances[a]->numRemainingEntities() << " for Alliance " <<
a+1 << endl;
00240                 cout << "How many would you like to place in " << keyPointName << " keypoint? ";
00241                 cin >> numEntitiesInKeyPt;
00242                 cin.ignore();
00243

```

```

00244         if (alliances[a]->numRemainingEntities() > 0 &&
alliances[a]->numRemainingEntities() < numEntitiesInKeyPt) {
00245             cout << "You selected more than the available amount. Try again " << endl;
00246             goto tryAgain;
00247         } else if (alliances[a]->numRemainingEntities() <= 0) {
00248             continue;
00249         } else {
00250             keyPoint->moveEntitiesInto(alliances[a], numEntitiesInKeyPt);
00251         }
00252     }
00253
00254     battleGrounds[i]->addArea(keyPoint);
00255 }
00256
00257     mainBattleGround->addArea(battleGrounds[i]);
00258 }
00259
00260     for (int i = 0; i < numAlliesAndGenerals; i++) {
00261         mainBattleGround->addGeneral(generals[i]);
00262     }
00263
00264     WarEngine::getInstance().setWarTheatre(mainBattleGround);
00265 }
00266
00267 void EasySetup::runSimulation() {
00268     WarEngine::getInstance().simulate();
00269 }
00270
00271 void EasySetup::saveSimulationSetup() {
00272     // Getting the name of the save
00273     cout << "Please enter name of save: ";
00274     string saveName;
00275     getline(cin, saveName);
00276
00277     // saving the current state of the simulation
00278     saveArchive->addNewSave(saveName, WarEngine::getInstance().saveState());
00279 }
00280
00281 void EasySetup::loadPrevSave() {
00282     try{
00283         WarEngineMemento* saveFile = saveArchive->getLastSave();
00284         WarEngine::getInstance().loadSave(saveFile);
00285     }
00286     catch(const std::exception& error){
00287         std::cerr << error.what() << "\n";
00288     }
00289 }
00290
00291 void EasySetup::loadSpecificSave(string name) {
00292     try{
00293         WarEngineMemento* saveFile = saveArchive->getSave(name);
00294         WarEngine::getInstance().loadSave(saveFile);
00295     }
00296     catch(const std::out_of_range& range_error){
00297         std::cerr << range_error.what() << "\n";
00298     }
00299 }
00300
00301 #endif
00302
00303 #define EASYSETUP_H
00304 #include <iostream>
00305 #include <cctype>
00306 #include <string>
00307 #include <vector>
00308 #include "SaveArchive.h"
00309
00310 using namespace std;
00311
00312 class EasySetup

```

5.23 EasySetup.h

```

00001 #ifndef EASYSETUP_H
00002 #define EASYSETUP_H
00003 #include <iostream>
00004 #include <cctype>
00005 #include <string>
00006 #include <vector>
00007 #include "SaveArchive.h"
00008
00009 using namespace std;
00010
00011 class EasySetup

```

```

00012 {
00013     private:
00014         SaveArchive* saveArchive;
00015     public:
00016         EasySetup();
00017         void setupSimulation();
00018         void runSimulation();
00019         void loadPrevSave();
00020         void loadSpecificSave(std::string name);
00021         void saveSimulationSetup();
00022 };
00023 };
00024
00025 #endif

```

5.24 Entity.cpp

```

00001 #include "Entity.h"
00002 #include "Alliance.h"
00003
00004
00005 Entity::Entity() {
00006     health = 0;
00007     damage = 0;
00008     type = NULL;
00009 }
00010
00011 Entity::Entity(Type* type, int health, int damage) {
00012     this->health = health;
00013     this->damage = damage;
00014     this->type = type;
00015 }
00016
00017 Type* Entity::getType() {
00018     return this->type;
00019 }
00020
00021 void Entity::setType(Type* type) {
00022     this->type = type;
00023 }
00024
00025 Alliance* Entity::getAlliance() {
00026     return this->alliance;
00027 }
00028
00029 void Entity::setAlliance(Alliance* alliance) {
00030     this->alliance = alliance;
00031 }
00032
00033 int Entity::getHealth() {
00034     return this->health;
00035 }
00036
00037 void Entity::setHealth(int health) {
00038     this->health = health;
00039 }
00040
00041 int Entity::getDamage() {
00042     return this->damage;
00043 }
00044
00045 void Entity::setDamage(int damage) {
00046     this->damage = damage;
00047 }

```

5.25 Entity.h

```

00001 #ifndef ENTITY_H
00002 #define ENTITY_H
00003
00004 #include "Type.h"
00005
00006 class Alliance;
00007
00013 class Entity {
00014
00015 private:
00016     Type* type;
00017     Alliance* alliance;

```

```

00018     int health;
00019     int damage;
00020
00021 public:
00022     Entity();
00023
00029     Entity(Type* type, int health, int damage);
00030
00039     virtual Type* getType();
00040
00053     virtual void setType(Type* type);
00054
00063     virtual Alliance* getAlliance();
00064
00077     virtual void setAlliance(Alliance* alliance);
00078
00087     virtual int getHealth();
00088
00101     virtual void setHealth(int health);
00102
00111     virtual int getDamage();
00112
00125     virtual void setDamage(int damage);
00126
00153     virtual void takeDamage(int damage) = 0;
00154
00167     virtual void dealDamage(Entity* entity) = 0;
00168
00177     virtual Entity* clone() = 0;
00178 };
00179
00180 #endif

```

5.26 Factory.cpp

```

00001 #include "Factory.h"
00002
00003 Factory::Factory(Type* type, AddOn* addOn) {
00004     this->type = type;
00005     this->addOn = addOn;
00006 }
00007
00008 Factory::~Factory() {
00009     delete type;
00010     delete addOn;
00011 }
00012
00013 Type* Factory::getType() {
00014     return this->type;
00015 }
00016
00017 void Factory::setType(Type* type) {
00018     this->type = type;
00019 }
00020
00021 AddOn* Factory::getAddOn() {
00022     return this->addOn;
00023 }
00024
00025 void Factory::setAddOns(AddOn* addOn) {
00026     this->addOn = addOn;
00027 }

```

5.27 Factory.h

```

00001 #ifndef FACTORY_H
00002 #define FACTORY_H
00003
00004 #include "Type.h"
00005 #include "AddOn.h"
00006
00012 class Factory {
00013
00014 private:
00015     Type* type;
00016     AddOn* addOn;
00017
00018 public:
00025     Factory(Type* type, AddOn* addOn);

```

```

00026
00033     ~Factory();
00034
00035     virtual Entity* createEntity(Alliance* alliance) = 0;
00036
00045     Type* getType();
00046
00047
00060     void setType(Type* type);
00061
00062
00071     AddOn* getAddOn();
00072
00085     void setAddOns(AddOn* addOn);
00086
00095     virtual Factory* clone() = 0;
00096 };
00097
00098 #endif

```

5.28 General.cpp

```

00001 #include "General.h"
00002
00003 General::General(Alliance* alliance, Strategy* strategy) {
00004     this->alliance = alliance;
00005     this->strategy = strategy;
00006     numDeaths = 0;
00007 }
00008
00009 void General::initiateStrategy(KeyPoint* keyPoint) {
00010     numDeaths++;
00011     if (numDeaths >= 5) {
00012         strategy->performStrat(keyPoint, this->alliance);
00013         numDeaths = 0;
00014     }
00015 }
00016
00017 bool General::setStrategy(Strategy* strategy){
00018     this->strategy = strategy;
00019     return true;
00020 }
00021
00022 Alliance* General::getAlliance(){
00023     return this->alliance;
00024 }

```

5.29 General.h

```

00001 #ifndef GENERAL_H
00002 #define GENERAL_H
00003 #include "Alliance.h"
00004 #include "Strategy.h"
00005
00006 class KeyPoint;
00007
00008 class General {
00009
00010 private:
00011     Alliance* alliance;
00012     Strategy* strategy;
00013     int numDeaths;
00014
00015 public:
00022     General(Alliance* alliance, Strategy* strategy);
00023
00033     void initiateStrategy(KeyPoint* keyPoint);
00034
00049     bool setStrategy(Strategy* strategy);
00050
00059     Alliance* getAlliance();
00060 };
00061
00062 #endif

```

5.30 KeyPoint.cpp

```

00001 #include "KeyPoint.h"
00002 #include "Weather.h"
00003 #include "RoundStats.h"
00004 #include "Sunny.h"
00005 #include <time.h>
00006 #include <cstdlib>
00007 #include <iostream>
00008
00009 using namespace std;
00010
00011 KeyPoint::KeyPoint(string areaName): Area(areaName) {
00012     weather = new Sunny();
00013 }
00014
00015 KeyPoint::KeyPoint(KeyPoint& keyPoint): Area(keyPoint.getAreaName()) {
00016     for (int i = 0; i < keyPoint.entities.size(); i++)
00017         this->addEntity(keyPoint.entities[i]->clone());
00018
00019     weather = keyPoint.weather->clone();
00020 }
00021
00022 KeyPoint::~KeyPoint() {
00023     for (int i = 0; i < entities.size(); i++)
00024         delete entities[i];
00025
00026     for (int i = 0; i < generals.size(); i++)
00027         delete generals[i];
00028
00029     delete weather;
00030 }
00031
00032 bool KeyPoint::isKeyPoint() {
00033     return true;
00034 }
00035
00036 void KeyPoint::simulateBattle(Alliance* alliance) {
00037     int numUnits = 0;
00038     for (int i = 0; i < entities.size(); i++) {
00039         if (entities[i]->getAlliance() == alliance) {
00040             numUnits++;
00041         }
00042     }
00043
00044     if (numUnits != entities.size()) {
00045         for (int i = 0; i < entities.size(); i++) {
00046             if (entities[i]->getAlliance() == alliance) {
00047                 int random;
00048                 do {
00049                     random = rand() % entities.size();
00050                 } while (entities[random]->getAlliance() == alliance);
00051
00052                 if (rand() % (int)(weather->getMultiplier() * 100) <= (int)(weather->getMultiplier() *
100))
00053                     entities[i]->dealDamage(entities[random]);
00054             }
00055         }
00056     }
00057
00058     clearBattlefield(alliance);
00059 }
00060
00061 void KeyPoint::clearBattlefield(Alliance* alliance) {
00062     int destroyed = 0;
00063     double numUnits = 0;
00064     for (vector<Entity*>::iterator it = entities.begin(); it != entities.end(); ++it) {
00065         if ((*it)->getHealth() <= 0) {
00066             destroyed++;
00067             for (int i = 0; i < generals.size(); i++) {
00068                 if (generals[i]->getAlliance() == (*it)->getAlliance()) {
00069                     generals[i]->initiateStrategy(this);
00070                     delete *it;
00071                     entities.erase(it);
00072                 }
00073             }
00074         } else if ((*it)->getAlliance() == alliance) {
00075             numUnits++;
00076         }
00077     }
00078
00079     // saving stats
00080     string stats = getAreaName() + ":\n";
00081     stats += "Key Point Status: ";
00082     if (numUnits / entities.size() >= 0.6) {
00083         stats += "Winning\n";
00084         RoundStats::numWinningPoints++;
    
```

```

00085     } else if (numUnits / entities.size() >= 0.35) {
00086         stats += "Contested\n";
00087         RoundStats::numContestedPoints++;
00088     } else {
00089         stats += "Losing\n";
00090         RoundStats::numLosingPoints++;
00091     }
00092
00093     stats += "Number of Entities Destroyed by Alliance: " + to_string(destroyed) + "\n";
00094     stats += "Number of Entities/Total Amount of Entities: " + to_string((int)numUnits) + "/" +
to_string(entities.size());
00095
00096     RoundStats::keyPointInformation.push_back(stats);
00097     RoundStats::numEntitiesDestroyed += destroyed;
00098 }
00099
00100 void KeyPoint::moveEntitiesInto(Alliance* alliance, int numTroops) {
00101     vector<Entity*> troops = alliance->getReserveEntities(numTroops);
00102     for (int i = 0; i < troops.size(); i++)
00103         entities.push_back(troops[i]);
00104
00105     string stats = "Alliance " + to_string(alliance->getID()) + " moved " + to_string(troops.size()) +
" entities into " + getAreaName();
00106     RoundStats::entityMovementInformation.push_back(stats);
00107 }
00108
00109 void KeyPoint::moveEntitiesOutOf(Alliance* alliance, int numTroops) {
00110     int numMoved = 0;
00111     for (vector<Entity*>::iterator it = entities.begin(); it != entities.end() && numMoved !=
numTroops; ++it) {
00112         if ((*it)->getAlliance() == alliance) {
00113             numMoved++;
00114             alliance->addReserveEntity(*it);
00115             entities.erase(it);
00116         }
00117     }
00118
00119     string stats = "Alliance " + to_string(alliance->getID()) + " moved " + to_string(numMoved) + "
entities out of " + getAreaName();
00120     RoundStats::entityMovementInformation.push_back(stats);
00121 }
00122
00123 void KeyPoint::addEntity(Entity* entity) {
00124     entities.push_back(entity);
00125 }
00126
00127 void KeyPoint::addGeneral(General* general) {
00128     generals.push_back(general);
00129 }
00130
00131 void KeyPoint::removeGeneral(General* general) {
00132     for (vector<General*>::iterator it = generals.begin(); it != generals.end(); ++it) {
00133         if (*it == general) {
00134             delete *it;
00135             generals.erase(it);
00136             return;
00137         }
00138     }
00139 }
00140
00141 Area* KeyPoint::clone() {
00142     return new KeyPoint(*this);
00143 }
00144
00145 void KeyPoint::setWeather(Weather* weather) {
00146     delete this->weather;
00147     this->weather = weather;
00148 }
00149
00150 void KeyPoint::changeWeather() {
00151
00152     srand(time(0));
00153
00154     int randomNum = 1 + (rand() % 10);
00155     std::string currWeather = this->weather->getWeather();
00156
00157     if (currWeather == "Sunny" && randomNum > 6) // 60% chance of not changing weather from Sunny and
staying
00158         this->weather->handleChange(this);
00159     else if (currWeather == "Cloudy" && randomNum > 3) // 30% chance of not changing weather from
Cloudy and staying
00160         this->weather->handleChange(this);
00161     else if (currWeather == "Rainy" && randomNum > 1) // 10% chance of not changing weather from Rainy
and staying
00162         this->weather->handleChange(this);
00163
00164

```



```

00165 }
00166
00167 std::string KeyPoint::getWeather() const {
00168     return this->weather->getWeather();
00169 }

```

5.31 KeyPoint.h

```

00001 #ifndef KEYPOINT_H
00002 #define KEYPOINT_H
00003
00004 #include "Alliance.h"
00005 #include "Area.h"
00006 #include "Entity.h"
00007 #include "General.h"
00008 #include <vector>
00009
00010 class Weather;
00011
00017 class KeyPoint : public Area {
00018
00019 private:
00020     vector<Entity*> entities;
00021     vector<General*> generals;
00022     Weather* weather;
00023
00024 public:
00030     KeyPoint(std::string areaName);
00031
00037     KeyPoint(KeyPoint& keyPoint);
00038
00039     ~KeyPoint();
00040
00049     bool isKeyPoint();
00050
00063     void simulateBattle(Alliance* alliance);
00064
00074     void clearBattlefield(Alliance* alliance);
00075
00090     void moveEntitiesInto(Alliance* alliance, int numTroops);
00091
00106     void moveEntitiesOutOf(Alliance* alliance, int numTroops);
00107
00120     void addEntity(Entity* entity);
00121
00133     void addGeneral(General* general);
00134
00146     void removeGeneral(General* general);
00147
00156     Area* clone();
00157
00162     void changeWeather();
00163
00176     void setWeather(Weather* weather);
00177
00183     std::string getWeather() const;
00184
00185 };
00186
00187 #endif

```

5.32 Negotiator.cpp

```

00001 #include "Negotiator.h"
00002 #include <bits/stdc++.h>
00003
00004 Negotiator::Negotiator() {}
00005
00006 Negotiator::~Negotiator() {
00007     alliances.clear();
00008 }
00009
00010 bool Negotiator::sendPeace(Alliance* offerAlliance) {
00011
00012     for (int yy = 0; yy < alliances.size(); yy++)
00013     {
00014         if (alliances[yy] != offerAlliance) {
00015             if (alliances[yy]->considerPeace() == false)

```

```

00016         return false; // There is at least one enemy alliances that does not want the peace
00017     deal
00018     }
00019 }
00020
00021     return true; // All the alliances being fought against agreed to the peace deal
00022 }
00023
00024 void Negotiator::removeAlliance(Alliance* oldAlliance) {
00025
00026     for (int xx = 0; xx < alliances.size(); xx++)
00027     {
00028         if (alliances[xx]->getID() == oldAlliance->getID())
00029             alliances.erase( alliances.begin() + xx ); // Removes the specific alliances from this
00030         negotiator
00031     }
00032 }
00033
00034 void Negotiator::addAlliance(Alliance* newAlliance) {
00035
00036     if (std::find(alliances.begin(), alliances.end(), newAlliance) != alliances.end())
00037         alliances.push_back(newAlliance);
00038 }
00039 }
00040
00041 int Negotiator::getNumAlliances() {
00042     return this->alliances.size();
00043 }

```

5.33 Negotiator.h

```

00001 #ifndef NEGOTIATOR_H
00002 #define NEGOTIATOR_H
00003 #include <vector>
00004 #include "Alliance.h"
00005
00006 class Negotiator {
00007
00008 private:
00009     vector<Alliance*> alliances;
00010
00011 public:
00012     Negotiator();
00013     ~Negotiator();
00014
00015     bool sendPeace(Alliance* offerAlliance);
00016
00017     void removeAlliance(Alliance* oldAlliance);
00018
00019     void addAlliance(Alliance* newAlliance);
00020
00021     int getNumAlliances();
00022 };
00023
00024 #endif

```

5.34 NegotiatorTest.h

```

00001 #include <stdexcept>
00002 #include "Negotiator.h"
00003 #include "Alliance.h"
00004 #include "gtest/gtest.h"
00005
00006 namespace {
00007
00008     // Tests Negotiator Functionality
00009
00010     // Tests AddOn setEntity()
00011     // ===== Positive Testing =====
00012     // Test Preconditions Bounds
00013     TEST(NegotiatorOfferPeace, PositiveTesting) {
00014         Alliance* a = new Alliance();
00015         Alliance* b = new Alliance();
00016         Negotiator* n = new Negotiator();
00017         n->addAlliance(a);
00018         n->addAlliance(b);

```

```

00019         a->setNegotiator(n);
00020         b->setNegotiator(n);
00021
00022         if (a->offerPeace()) {
00023             EXPECT_EQ(3, a->getActive());
00024         } else {
00025             EXPECT_EQ(1, a->getActive());
00026         }
00027     }
00028
00029     TEST(NegotiatorSurrender, PositiveTesting) {
00030         Alliance* a = new Alliance();
00031         Alliance* b = new Alliance();
00032         Alliance* c = new Alliance();
00033         Alliance* d = new Alliance();
00034         Alliance* e = new Alliance();
00035         Negotiator* n = new Negotiator();
00036         n->addAlliance(a);
00037         n->addAlliance(b);
00038         n->addAlliance(c);
00039         n->addAlliance(d);
00040         n->addAlliance(e);
00041         a->setNegotiator(n);
00042         b->setNegotiator(n);
00043         c->setNegotiator(n);
00044         d->setNegotiator(n);
00045         e->setNegotiator(n);
00046
00047         a->surrender();
00048         EXPECT_EQ(2, a->getActive());
00049
00050         b->surrender();
00051         EXPECT_EQ(2, a->getActive());
00052
00053         c->surrender();
00054         EXPECT_EQ(2, a->getActive());
00055
00056         d->surrender();
00057         EXPECT_EQ(2, a->getActive());
00058     }
00059 }

```

5.35 Passive.cpp

```

00001 #include "Passive.h"
00002
00003 using namespace std;
00004
00005 Passive::Passive() {}
00006
00007 void Passive::performStrat(KeyPoint* keyPoint, Alliance* alliance) {
00008
00009     int randomNumber = (rand() % 10) + 5;
00010     keyPoint->moveEntitiesOutOf(alliance, randomNumber);
00011 }
00012
00013 Strategy* Passive::clone() {
00014     return new Passive();
00015 }

```

5.36 Passive.h

```

00001 #ifndef PASSIVE_H
00002 #define PASSIVE_H
00003 #include "Strategy.h"
00004 #include "KeyPoint.h"
00005
00006 class Passive : public Strategy {
00007
00008 public:
00009     Passive();
00010
00011     void performStrat(KeyPoint* keyPoint, Alliance* alliance);
00012
00013     Strategy* clone();
00014 };
00015
00016 #endif

```

5.37 Personnel.cpp

```

00001 #include "Personnel.h"
00002 #include "RoundStats.h"
00003 #include <iostream>
00004 #include <stdexcept>
00005
00006 Personnel::Personnel(Type* type, int health, int damage): Entity(type, health, damage) {}
00007
00008 void Personnel::takeDamage(int damage) {
00009     if (damage <= 0)
00010         throw std::invalid_argument("damage must be greater than zero");
00011
00012     setHealth(getHealth() - damage);
00013 }
00014
00015 void Personnel::dealDamage(Entity* entity) {
00016     RoundStats::damageDone += getDamage();
00017     entity->takeDamage(getDamage());
00018 }
00019
00020 Entity* Personnel::clone() {
00021     Personnel* p;
00022     if (this->getType() == NULL) {
00023         p = new Personnel(NULL, this->getHealth(), this->getDamage());
00024     } else {
00025         p = new Personnel(this->getType()->clone(), this->getHealth(), this->getDamage());
00026     }
00027
00028     p->setAlliance(this->getAlliance());
00029
00030     return p;
00031 }

```

5.38 Personnel.h

```

00001 #ifndef PERSONNEL_H
00002 #define PERSONNEL_H
00003
00004 #include "Entity.h"
00005
00011 class Personnel : public Entity {
00012
00013 public:
00021     Personnel(Type* type, int health = 100, int damage = 10);
00022
00038     void takeDamage(int damage);
00039
00052     void dealDamage(Entity* entity);
00053
00062     Entity* clone();
00063 };
00064
00065 #endif

```

5.39 PersonnelFactory.cpp

```

00001 #include "PersonnelFactory.h"
00002 #include "Personnel.h"
00003 #include <iostream>
00004
00005 PersonnelFactory::PersonnelFactory(Type* type, AddOn* addOn): Factory(type, addOn) {}
00006
00007 Entity* PersonnelFactory::createEntity(Alliance* alliance) {
00008     Personnel* p = new Personnel(getType()->clone());
00009     p->setAlliance(alliance);
00010     if (getAddOn() != NULL) {
00011         AddOn* personnelAddOn = getAddOn()->clone();
00012         personnelAddOn->setEntity(p);
00013         return personnelAddOn;
00014     } else {
00015         return p;
00016     }
00017 }
00018
00019 Factory* PersonnelFactory::clone() {
00020     return new PersonnelFactory(getType(), getAddOn());
00021 }

```

5.40 PersonnelFactory.h

```

00001 #ifndef PERSONNELFACTORY_H
00002 #define PERSONNELFACTORY_H
00003
00004 #include "Factory.h"
00005
00011 class PersonnelFactory : public Factory {
00012
00013 public:
00020     PersonnelFactory(Type* type, AddOn* addOn);
00021
00034     Entity* createEntity(Alliance* alliance);
00035
00044     Factory* clone();
00045 };
00046
00047 #endif

```

5.41 Piercing.cpp

```

00001 #include "Piercing.h"
00002 #include "RoundStats.h"
00003 #include <stdexcept>
00004
00005 Piercing::Piercing(int value) : AddOn(value) {}
00006
00007 void Piercing::takeDamage(int damage) {
00008     if (damage <= 0)
00009         throw std::invalid_argument("damage must be greater than zero");
00010
00011     entity->takeDamage(damage);
00012 }
00013
00014 void Piercing::dealDamage(Entity* entity) {
00015     int sumValue = this->entity->getDamage() + value;
00016     entity->takeDamage(sumValue);
00017     RoundStats::damageDone += sumValue;
00018 }
00019
00020 AddOn* Piercing::clone() {
00021     Piercing* piercing = new Piercing(value);
00022     if (getEntity() != NULL)
00023         piercing->setEntity(entity->clone());
00024     return piercing;
00025 }

```

5.42 Piercing.h

```

00001 #ifndef PIERCING_H
00002 #define PIERCING_H
00003 #include "AddOn.h"
00004 #include "Entity.h"
00005
00011 class Piercing : public AddOn {
00012
00013 public:
00020     Piercing(int value);
00021
00037     void takeDamage(int damage);
00038
00051     void dealDamage(Entity* entity);
00052
00061     AddOn* clone();
00062 };
00063
00064 #endif

```

5.43 Rainy.cpp

```

00001 #include "Rainy.h"
00002 #include "Sunny.h"
00003
00004 Rainy::Rainy(): Weather() {

```

```

00005     this->multiplier = 0.5;
00006 }
00007
00008 std::string Rainy::getWeather() {
00009     return "Rainy";
00010 }
00011
00012 void Rainy::handleChange(KeyPoint* keypoint) {
00013     Sunny* newWeather = new Sunny();
00014     keypoint->setWeather(newWeather);
00015 }
00016
00017 Weather* Rainy::clone() {
00018     return new Rainy();
00019 }

```

5.44 Rainy.h

```

00001 #ifndef RAINY_H
00002 #define RAINY_H
00003 #include "Weather.h"
00004 #include "KeyPoint.h"
00005
00006 class Rainy : public Weather {
00007
00008
00009 public:
00010     Rainy();
00011
00012     std::string getWeather();
00013
00014     void handleChange(KeyPoint* keypoint);
00015
00016     Weather* clone();
00017 };
00018 #endif

```

5.45 RoundStats.cpp

```

00001 #include "RoundStats.h"
00002
00003 int RoundStats::numEntitiesCreated = 0;
00004 int RoundStats::numEntitiesDestroyed = 0;
00005 int RoundStats::damageDone = 0;
00006 int RoundStats::numLosingPoints = 0;
00007 int RoundStats::numContestedPoints = 0;
00008 int RoundStats::numWinningPoints = 0;
00009 vector<string> RoundStats::keyPointInformation;
00010 vector<string> RoundStats::entityMovementInformation;
00011
00012 void RoundStats::clearStats() {
00013     numEntitiesCreated = 0;
00014     numEntitiesDestroyed = 0;
00015     damageDone = 0;
00016     numLosingPoints = 0;
00017     numContestedPoints = 0;
00018     numWinningPoints = 0;
00019     keyPointInformation.clear();
00020     entityMovementInformation.clear();
00021 }
00022
00023 string RoundStats::toString() {
00024     string out = "Number of Key Points Winning/Contested/Losing: " + to_string(numWinningPoints) +
00025         "/" + to_string(numContestedPoints) + "/" + to_string(numLosingPoints) + "\n";
00026     out += "Number of Entities Created: " + to_string(numEntitiesCreated) + "\n";
00027     out += "Number of Entities Destroyed by Alliance: " + to_string(numEntitiesDestroyed) + "\n";
00028     out += "Damage Given by Alliance: " + to_string(damageDone) + "\n";
00029
00030     out += "\nKey Point Round Information:\n";
00031     for (int i = 0; i < keyPointInformation.size(); i++)
00032         out += keyPointInformation[i] + "\n";
00033
00034     out += "\nMovement Round Information:\n";
00035     for (int i = 0; i < entityMovementInformation.size(); i++)
00036         out += entityMovementInformation[i] + "\n";
00037     return out;
00038 }

```

5.46 RoundStats.h

```

00001 #ifndef ROUNDSTATS_H
00002 #define ROUNDSTATS_H
00003
00004 #include <vector>
00005 #include <string>
00006
00007 using namespace std;
00008
00009 class RoundStats {
00010     public:
00011         static int numEntitiesCreated;
00012         static int numEntitiesDestroyed;
00013         static int damageDone;
00014         static int numLosingPoints;
00015         static int numContestedPoints;
00016         static int numWinningPoints;
00017         static vector<string> keyPointInformation;
00018         static vector<string> entityMovementInformation;
00019
00020         static void clearStats();
00021         static string toString();
00022 };
00023
00024 #endif

```

5.47 SaveArchive.cpp

```

00001 #include "SaveArchive.h"
00002
00003 SaveArchive::SaveArchive() {}
00004
00005 void SaveArchive::addNewSave(std::string newSaveName, WarEngineMemento* newSave) {
00006     saveList.insert({newSaveName, newSave});
00007 }
00008
00009 WarEngineMemento* SaveArchive::getLastSave() {
00010
00011     if(saveList.size() == 0){
00012         throw "Save archive is empty.";
00013     }
00014
00015     WarEngineMemento* lastSave = saveList.begin()->second;
00016
00017     saveList.erase( saveList.begin() );
00018
00019     return lastSave;
00020 }
00021
00022 WarEngineMemento* SaveArchive::getSave(std::string name) {
00023     if(saveList.size() == 0){
00024         std::__throw_out_of_range("Save archive is empty");
00025     }
00026
00027     auto iter = saveList.find(name);
00028
00029     if(iter == saveList.end())
00030         std::__throw_invalid_argument("No save with given name exists");
00031
00032     return iter->second;
00033 }
00034
00035 void SaveArchive::clearSaveList() {
00036     saveList.clear();
00037 }
00038
00039 void SaveArchive::deleteSave(std::string name) {
00040     if(saveList.size() == 0){
00041         std::__throw_out_of_range("Save archive is empty");
00042     }
00043
00044     auto iter = saveList.find(name) ;
00045
00046     if(iter == saveList.end())
00047         return;
00048
00049     saveList.erase( iter );
00050 }

```

5.48 SaveArchive.h

```

00001 #ifndef SAVEARCHIVE_H
00002 #define SAVEARCHIVE_H
00003 #include <unordered_map>
00004 #include <string>
00005 #include "WarEngineMemento.h"
00006
00011 class SaveArchive {
00012
00013 private:
00014     std::unordered_map<std::string, WarEngineMemento*> saveList;
00015
00016 public:
00020     SaveArchive();
00021
00035     void addNewSave(std::string newSaveName, WarEngineMemento* newSave);
00036
00047     WarEngineMemento* getLastSave();
00048
00063     WarEngineMemento* getSave(std::string name);
00064
00072     void clearSaveList();
00073
00087     void deleteSave(std::string name);
00088 };
00089
00090 #endif

```

5.49 Strategy.cpp

```

00001 #include "Strategy.h"
00002 #include "KeyPoint.h"
00003 #include "Alliance.h"
00004
00005 using namespace std;
00006
00007 Strategy::Strategy() {}
00008
00009 Strategy::~Strategy() {}
00010

```

5.50 Strategy.h

```

00001 #ifndef STRATEGY_H
00002 #define STRATEGY_H
00003 #include <string>
00004 #include <ctime>
00005 #include <cstdlib>
00006
00007 class KeyPoint;
00008 class Alliance;
00009
00010 class Strategy {
00011
00012 protected:
00013     std::string strategy;
00014
00015 public:
00020     Strategy();
00021
00026     ~Strategy();
00027
00037     virtual void performStrat(KeyPoint* keyPoint, Alliance* alliance) = 0;
00038
00039     virtual Strategy* clone() = 0;
00049
00050 };
00051
00052 #endif

```

5.51 Sunny.cpp

```

00001 #include "Sunny.h"
00002 #include "Cloudy.h"

```



```

00003
00004 Sunny::Sunny() {
00005     this->multiplier = 1.0;
00006 }
00007
00008 std::string Sunny::getWeather() {
00009     return "Sunny";
00010 }
00011
00012 void Sunny::handleChange(KeyPoint* k) {
00013     Cloudy* newWeather = new Cloudy();
00014     k->setWeather(newWeather);
00015 }
00016
00017 Weather* Sunny::clone() {
00018     return new Sunny();
00019 }

```

5.52 Sunny.h

```

00001 #ifndef SUNNY_H
00002 #define SUNNY_H
00003 #include "Weather.h"
00004 #include "KeyPoint.h"
00005
00006 #include "Weather.h"
00007
00008 class Sunny : public Weather {
00009 public:
00010     Sunny();
00011
00012     virtual std::string getWeather();
00013
00014     virtual void handleChange(KeyPoint* keypoint);
00015
00016     Weather* clone();
00017 };
00018
00019 #endif

```

5.53 Support.cpp

```

00001 #include "Support.h"
00002 #include "RoundStats.h"
00003 #include <stdexcept>
00004
00005 Support::Support(Type* type, int health, int damage): Entity(type, health, damage) {}
00006
00007 void Support::dealDamage(Entity* entity) {
00008     RoundStats::damageDone += getDamage();
00009     entity->takeDamage(getDamage());
00010 }
00011
00012 void Support::takeDamage(int damage) {
00013     if (damage <= 0)
00014         throw std::invalid_argument("damage must be greater than zero");
00015
00016     this->setHealth(this->getHealth() - damage);
00017 }
00018
00019 Entity* Support::clone() {
00020     Support* s;
00021     if (this->getType() == NULL) {
00022         s = new Support(NULL, this->getHealth(), this->getDamage());
00023     } else {
00024         s = new Support(this->getType()->clone(), this->getHealth(), this->getDamage());
00025     }
00026
00027     s->setAlliance(this->getAlliance());
00028
00029     return s;
00030 }

```

5.54 Support.h

```

00001 #ifndef SUPPORT_H

```

```

00002 #define SUPPORT_H
00003
00004 #include "Entity.h"
00005
00011 class Support : public Entity {
00012
00013 public:
00021     Support(Type* type, int health = 1000, int damage = 30);
00022
00038     void takeDamage(int damage);
00039
00052     void dealDamage(Entity* entity);
00053
00059     Entity* clone();
00060 };
00061
00062 #endif

```

5.55 SupportFactory.cpp

```

00001 #include "SupportFactory.h"
00002 #include "Support.h"
00003
00004 SupportFactory::SupportFactory(Type* type, AddOn* addOn): Factory(type, addOn) {}
00005
00006 Entity* SupportFactory::createEntity(Alliance* alliance) {
00007     Support* s = new Support(getType()->clone());
00008     s->setAlliance(alliance);
00009     if (getAddOn() != NULL) {
00010         AddOn* personnelAddOn = getAddOn()->clone();
00011         personnelAddOn->setEntity(s);
00012         return personnelAddOn;
00013     } else {
00014         return s;
00015     }
00016 }
00017
00018 Factory* SupportFactory::clone() {
00019     return new SupportFactory(getType()->clone(), getAddOn()->clone());
00020 }

```

5.56 SupportFactory.h

```

00001 #ifndef SUPPORTFACTORY_H
00002 #define SUPPORTFACTORY_H
00003
00004 #include "Factory.h"
00005
00011 class SupportFactory : public Factory {
00012
00013 public:
00020     SupportFactory(Type* type, AddOn* addOn);
00021
00034     Entity* createEntity(Alliance* alliance);
00035
00044     Factory* clone();
00045 };
00046
00047 #endif

```

5.57 TerrainType.cpp

```

00001 #include "TerrainType.h"
00002
00003 TerrainType::TerrainType() {}
00004
00005 string TerrainType::getTypeDesc() {
00006     return "Terrain";
00007 }
00008
00009 Type* TerrainType::clone() {
00010     return new TerrainType();
00011 }

```

5.58 TerrainType.h

```

00001 #ifndef TERRAINTYPE_H
00002 #define TERRAINTYPE_H
00003
00004 #include "Type.h"
00005
00011 class TerrainType : public Type {
00012
00013 public:
00017     TerrainType();
00018
00027     string getTypeDesc();
00028
00037     Type* clone();
00038 };
00039
00040 #endif

```

5.59 testmain.cpp

```

00001 #include "NegotiatorTest.h"
00002 #include "AddOnTest.h"
00003 #include <gtest/gtest.h>
00004 #include "EasySetup.h"
00005 #include "WarEngine.h"
00006 #include "KeyPoint.h"
00007 #include "Negotiator.h"
00008 #include <iostream>
00009
00010 void setupWarEngine() {
00011     Alliance* a1 = new Alliance();
00012     a1->addCountry(new Country("Germany"));
00013
00014     Alliance* a2 = new Alliance();
00015     a2->addCountry(new Country("Finland"));
00016
00017     Negotiator* n = new Negotiator();
00018     n->addAlliance(a1);
00019     n->addAlliance(a2);
00020     a1->setNegotiator(n);
00021     a2->setNegotiator(n);
00022
00023     WarEngine::getInstance().addAlliance(a1);
00024     WarEngine::getInstance().addAlliance(a2);
00025
00026     KeyPoint* k1 = new KeyPoint("West");
00027     KeyPoint* k2 = new KeyPoint("North");
00028     KeyPoint* k3 = new KeyPoint("East");
00029
00030     for (int i = 0; i < 100; i++) {
00031         Personnel* p1 = new Personnel(NULL);
00032         p1->setAlliance(a1);
00033         k1->addEntity(p1->clone());
00034         k2->addEntity(p1->clone());
00035         k3->addEntity(p1->clone());
00036
00037         Personnel* p2 = new Personnel(NULL);
00038         p2->setAlliance(a2);
00039         k1->addEntity(p2->clone());
00040         k2->addEntity(p2->clone());
00041     }
00042
00043     WarTheatre* w = new WarTheatre("Europe");
00044     w->addArea(k1);
00045     w->addArea(k2);
00046     w->addArea(k3);
00047
00048     WarEngine::getInstance().setWarTheatre(w);
00049 }
00050
00051 void showTests(int &argc, char** argv){}
00052
00053 int startWarEngine(int &argc, char** argv){
00054
00055     bool continueLoop = true;
00056
00057     while(continueLoop){
00058
00059         cout << "Welcome to the War Simulator!\n" << "Please select an option:" << endl;
00060
00061         cout << "1) Run Google Tests\n" << "2) Setup Simulation\n" << "3)Quit\n" << endl;
00062

```

```

00063         std::string userStringInput;
00064
00065         cin » userStringInput;
00066
00067         int userOption = stoi(userStringInput);
00068
00069         switch(userOption){
00070             case 1:
00071                 testing::InitGoogleTest(&argc, argv);
00072                 RUN_ALL_TESTS();
00073                 cout « "\n" « endl;
00074                 break;
00075             case 2:
00076                 setupWarEngine();
00077                 WarEngine::getInstance().simulate();
00078                 cout « "\n" « endl;
00079                 break;
00080             case 3:
00081                 continueLoop = false;
00082                 cout « "\n" « endl;
00083                 break;
00084             default:
00085                 cout « "Please try again. Enter a valid option.\n\n" « endl;
00086         }
00087     }
00088
00089     return 0;
00090 }
00091 }
00092
00093 int main(int argc, char **argv) {
00094     //setupWarEngine();
00095
00096     //WarEngine::getInstance().simulate();
00097
00098     //testing::InitGoogleTest(&argc, argv);
00099     //return RUN_ALL_TESTS();
00100     startWarEngine(argc, argv);
00101
00102     return 0;
00103 }

```

5.60 Type.cpp

```

00001 #include "Type.h"
00002
00003 Type::Type() {}

```

5.61 Type.h

```

00001 #ifndef TYPE_H
00002 #define TYPE_H
00003
00004 #include <string>
00005
00006 using namespace std;
00007
00013 class Type {
00014 public:
00015     Type();
00019
00020     virtual string getTypeDesc() = 0;
00029
00030     virtual Type* clone() = 0;
00039
00040 };
00041 };
00042
00043 #endif

```

5.62 Vehicle.cpp

```

00001 #include "Vehicle.h"
00002 #include "RoundStats.h"
00003 #include <stdexcept>

```

```

00004
00005 Vehicle::Vehicle(Type* type, int health, int damage): Entity(type, health, damage) {}
00006
00007 void Vehicle::takeDamage(int damage) {
00008     if (damage <= 0)
00009         throw std::invalid_argument("damage must be greater than zero");
00010
00011     setHealth(getHealth() - damage);
00012 }
00013
00014 void Vehicle::dealDamage(Entity* entity) {
00015     RoundStats::damageDone += getDamage();
00016     entity->takeDamage(getDamage());
00017 }
00018
00019 Entity* Vehicle::clone() {
00020     Vehicle* v;
00021     if (this->getType() == NULL) {
00022         v = new Vehicle(NULL, this->getHealth(), this->getDamage());
00023     } else {
00024         v = new Vehicle(this->getType()->clone(), this->getHealth(), this->getDamage());
00025     }
00026
00027     v->setAlliance(this->getAlliance());
00028
00029     return v;
00030 }

```

5.63 Vehicle.h

```

00001 #ifndef VEHICLE_H
00002 #define VEHICLE_H
00003
00004 #include "Entity.h"
00005
00011 class Vehicle : public Entity {
00012
00013 public:
00021     Vehicle(Type* type, int health = 500, int damage = 10);
00022
00038     void takeDamage(int damage);
00039
00052     void dealDamage(Entity* entity);
00053
00059     Entity* clone();
00060 };
00061
00062 #endif

```

5.64 VehicleFactory.cpp

```

00001 #include "VehicleFactory.h"
00002 #include "Vehicle.h"
00003
00004 VehicleFactory::VehicleFactory(Type* type, AddOn* addOn): Factory(type, addOn) {}
00005
00006 Entity* VehicleFactory::createEntity(Alliance* alliance) {
00007     Vehicle* v = new Vehicle(getType()->clone());
00008     v->setAlliance(alliance);
00009     if (getAddOn() != NULL) {
00010         AddOn* personnelAddOn = getAddOn()->clone();
00011         personnelAddOn->setEntity(v);
00012         return personnelAddOn;
00013     } else {
00014         return v;
00015     }
00016 }
00017
00018 Factory* VehicleFactory::clone() {
00019     return new VehicleFactory(getType()->clone(), getAddOn()->clone());
00020 }

```

5.65 VehicleFactory.h

```

00001 #ifndef VEHICLEFACTORY_H

```

```

00002 #define VEHICLEFACTORY_H
00003 #include "Factory.h"
00004
00010 class VehicleFactory : public Factory {
00011
00012 public:
00019     VehicleFactory(Type* type, AddOn* addOn);
00020
00033     Entity* createEntity(Alliance* alliance);
00034
00043     Factory* clone();
00044 };
00045
00046 #endif

```

5.66 WarEngine.cpp

```

00001 #include "WarEngine.h"
00002 #include "RoundStats.h"
00003 #include <iostream>
00004
00005 WarEngine::WarEngine() {
00006     this->state = new WarEngineState();
00007 }
00008
00009 WarEngineMemento* WarEngine::saveState() {
00010     return new WarEngineMemento(state->clone());
00011 }
00012
00013 void WarEngine::loadSave(WarEngineMemento* save) {
00014     delete this->state;
00015     this->state = save->getState();
00016 }
00017
00018 WarEngine& WarEngine::getInstance() {
00019     static WarEngine uniqueInstance_;
00020     return uniqueInstance_;
00021 }
00022
00023 WarEngine::~WarEngine() {
00024     delete this->state;
00025 }
00026
00027
00028 void WarEngine::simulate() {
00029
00030     vector<Alliance*> alliances = this->state->getAlliances();
00031     int numAlliances = alliances.size();
00032     while (numAlliances > 1) {
00033         numAlliances = 0;
00034         for(int i = 0; i < alliances.size(); i++) {
00035             if (alliances[i]->getActive() == 1) {
00036                 numAlliances++;
00037                 RoundStats::clearStats();
00038                 state->getArea()->simulateBattle(alliances[i]);
00039
00040                 double percLoss = (RoundStats::numLosingPoints * 1.0) / (RoundStats::numLosingPoints +
RoundStats::numContestedPoints + RoundStats::numWinningPoints);
00041                 if (percLoss >= 0.7) {
00042                     alliances[i]->surrender();
00043                 } else if (percLoss >= 0.6) {
00044                     alliances[i]->offerPeace();
00045                 }
00046
00047                 cout << "===== " <<
endl << endl;
00048                 cout << "Alliance " << alliances[i]->getID() << ":" << endl;
00049
00050                 if (alliances[i]->getActive() == 2) {
00051                     cout << "Status: Surrendered" << endl;
00052                 } else if (alliances[i]->getActive() == 3) {
00053                     cout << "Status: Found Peace" << endl;
00054                 } else {
00055                     cout << "Status: Active" << endl;
00056                 }
00057
00058                 cout << RoundStats::toString() << endl;
00059                 cout << "===== " <<
endl;
00060             }
00061         }
00062     }
00063 }

```

```

00064     for(int i = 0; i < alliances.size(); i++) {
00065         cout << "===== " << endl <<
endl;
00066         cout << "Alliance " << alliances[i]->getID() << ":" << endl;
00067
00068         if (alliances[i]->getActive() == 2) {
00069             cout << "Status: Surrendered" << endl;
00070         } else if (alliances[i]->getActive() == 3) {
00071             cout << "Status: Found Peace" << endl;
00072         } else {
00073             cout << "Status: Winner" << endl;
00074         }
00075
00076         cout << endl << "===== " <<
endl;
00077     }
00078
00079     cout << "SIMULATION COMPLETE!" << endl;
00080 }
00081 }
00082
00083 void WarEngine::setWarTheatre(WarTheatre* battleGround) {
00084     state->setArea(battleGround);
00085 }
00086
00087 void WarEngine::addAlliance(Alliance* alliance) {
00088     state->alliances.push_back(alliance);
00089 }

```

5.67 WarEngine.h

```

00001 #ifndef WARENGINE_H
00002 #define WARENGINE_H
00003
00004 #include "WarEngineState.h"
00005 #include "WarEngineMemento.h"
00006 #include "WarTheatre.h"
00007
00012 class WarEngine {
00013
00014 private:
00015     WarEngineState* state;
00016     bool gameOver;
00017
00018 protected:
00023     WarEngine();
00024
00033     WarEngine(const WarEngine&){};
00034
00038     WarEngine& operator=(const WarEngine&){ return *this; };
00039
00044     ~WarEngine();
00045
00046 public:
00051     WarEngineMemento* saveState();
00052
00065     void loadSave(WarEngineMemento* save);
00066
00072     static WarEngine& getInstance();
00073
00087     void setWarTheatre(WarTheatre* battleGround);
00088
00089     void addAlliance(Alliance* alliance);
00090
00099     void simulate();
00100
00101 };
00102
00103 #endif

```

5.68 WarEngineMemento.cpp

```

00001 #include "WarEngineMemento.h"
00002
00003 WarEngineMemento::WarEngineMemento(WarEngineState * state){
00004     this->state = state;
00005 }
00006
00007 void WarEngineMemento::setState(WarEngineState* state){

```

```

00008     this->state = state;
00009 }
00010
00011 WarEngineState* WarEngineMemento::getState() {
00012     return state;
00013 }

```

5.69 WarEngineMemento.h

```

00001 #ifndef WARENGINEMEMENTO_H
00002 #define WARENGINEMEMENTO_H
00003
00004 #include "WarEngineState.h"
00005 #include <string>
00006 #include <vector>
00007
00008 class WarEngine;
00009
00015 class WarEngineMemento {
00016
00017     friend class WarEngine;
00018
00019 private:
00020     WarEngineState* state;
00021
00028     WarEngineMemento(WarEngineState* state);
00029
00039     void setState(WarEngineState* state);
00040
00048     WarEngineState* getState();
00049
00050 };
00051
00052 #endif

```

5.70 WarEngineState.cpp

```

00001 #include "WarEngineState.h"
00002
00003 WarEngineState::WarEngineState() {
00004     area = nullptr;
00005 }
00006
00007 void WarEngineState::setArea(Area* area) {
00008     this->area = area;
00009 }
00010
00011 Area* WarEngineState::getArea() {
00012
00013     if(area == nullptr)
00014         throw "No Areas Stored.";
00015
00016     return this->area;
00017 }
00018
00019 void WarEngineState::setAlliances(vector<Alliance*> alliances) {
00020     this->alliances = alliances;
00021 }
00022
00023 vector<Alliance*> WarEngineState::getAlliances() {
00024
00025     if(alliances.size() == 0)
00026         std::__throw_out_of_range("No Alliances stored.");
00027
00028     return alliances;
00029 }
00030
00031 WarEngineState* WarEngineState::clone() {
00032
00033     WarEngineState* clonedState = new WarEngineState();
00034
00035     clonedState->setArea( this->area->clone() );
00036
00037     for(Alliance* alliance : this->alliances){
00038
00039         Alliance* clonedAlliance = alliance->clone();
00040
00041         clonedState->alliances.push_back(alliance);
00042

```



```

00043     }
00044
00045     return clonedState;
00046 }
00047
00048 WarEngineState::~WarEngineState(){
00049
00050     for(Alliance* alliance : this->alliances){
00051         delete alliance;
00052     }
00053
00054     delete this->area;
00055 }

```

5.71 WarEngineState.h

```

00001 #ifndef WARENGINESTATE_H
00002 #define WARENGINESTATE_H
00003 #include "Alliance.h"
00004 #include "Area.h"
00005 #include <vector>
00006
00007 class WarEngine;
00008
00009 using namespace std;
00010
00017 class WarEngineState {
00018 friend class WarEngine;
00019
00020 private:
00021     Area* area;
00022     vector<Alliance*> alliances;
00023
00024 protected:
00029     WarEngineState();
00030
00043     void setArea(Area* area);
00044
00053     Area* getArea();
00054
00067     void setAlliances(vector<Alliance*> alliances);
00068
00077     vector<Alliance*> getAlliances();
00078
00084     WarEngineState* clone();
00085
00086 public:
00090     ~WarEngineState();
00091 };
00092
00093 #endif

```

5.72 WarTheatre.cpp

```

00001 #include "WarTheatre.h"
00002
00003 using namespace std;
00004
00005 WarTheatre::WarTheatre(string areaName): Area(areaName) {}
00006
00007 WarTheatre::~WarTheatre() {
00008     for (int i = 0; i < areas.size(); i++)
00009         delete areas[i];
00010 }
00011
00012 bool WarTheatre::isKeyPoint() {
00013     return false;
00014 }
00015
00016 void WarTheatre::simulateBattle(Alliance* alliance) {
00017     for (int i = 0; i < areas.size(); i++)
00018         areas[i]->simulateBattle(alliance);
00019 }
00020
00021 void WarTheatre::addArea(Area* area) {
00022     areas.push_back(area);
00023 }
00024
00025 WarTheatre* WarTheatre::clone() {

```

```

00026     WarTheatre* w = new WarTheatre(getAreaName());
00027
00028     for (int i = 0; i < areas.size(); i++)
00029         w->addArea(areas[i]->clone());
00030
00031     return w;
00032 }
00033
00034 void WarTheatre::addGeneral(General* general) {
00035     for (int i = 0; i < areas.size(); i++)
00036         areas[i]->addGeneral(general);
00037 }

```

5.73 WarTheatre.h

```

00001 #ifndef WARTHEATRE_H
00002 #define WARTHEATRE_H
00003
00004 #include "Area.h"
00005 #include "Alliance.h"
00006 #include <vector>
00007
00008 using namespace std;
00009
00010 class WarTheatre : public Area {
00011
00012 private:
00013     vector<Area*> areas;
00014
00015 public:
00016     WarTheatre(std::string areaName);
00017
00018     ~WarTheatre();
00019
00020     bool isKeyPoint();
00021
00022     void simulateBattle(Alliance* alliance);
00023
00024     void addArea(Area* area);
00025
00026     void addGeneral(General* general);
00027
00028     WarTheatre* clone();
00029 };
00030
00031 #endif

```

5.74 Weather.cpp

```

00001 #include "Weather.h"
00002
00003 Weather::Weather() {}
00004
00005 Weather::~Weather() {}
00006
00007 double Weather::getMultiplier() {
00008     return this->multiplier;
00009 }

```

5.75 Weather.h

```

00001 #ifndef WEATHER_H
00002 #define WEATHER_H
00003 #include <string>
00004 #include "Weather.h"
00005 #include "KeyPoint.h"
00006
00007 class KeyPoint;
00008
00009 class Weather {
00010
00011 protected:
00012     double multiplier;
00013
00014 public:

```

```
00018     Weather();
00019
00023     ~Weather();
00024
00033     double getMultiplier();
00034
00047     virtual void handleChange(KeyPoint* keypoint) = 0;
00048
00057     virtual std::string getWeather() = 0;
00058
00064     virtual Weather* clone() = 0;
00065 };
00066
00067 #endif
```


Index

- ~Alliance
 - Alliance, [22](#)
- ~Area
 - Area, [32](#)
- ~Factory
 - Factory, [59](#)
- ~KeyPoint
 - KeyPoint, [66](#)
- ~Negotiator
 - Negotiator, [74](#)
- ~Strategy
 - Strategy, [97](#)
- ~WarEngine
 - WarEngine, [117](#)
- ~WarEngineState
 - WarEngineState, [123](#)
- ~WarTheatre
 - WarTheatre, [127](#)
- ~Weather
 - Weather, [131](#)
- addAlliance
 - Negotiator, [75](#)
 - WarEngine, [117](#)
- addArea
 - WarTheatre, [127](#)
- addCountry
 - Alliance, [23](#)
- addEntity
 - KeyPoint, [66](#)
- addFactory
 - Alliance, [23](#)
- addGeneral
 - Area, [33](#)
 - KeyPoint, [67](#)
 - WarTheatre, [128](#)
- addNewSave
 - SaveArchive, [93](#)
- AddOn, [7](#)
 - AddOn, [8](#)
 - clone, [9](#)
 - dealDamage, [9](#)
 - entity, [17](#)
 - getAlliance, [10](#)
 - getDamage, [10](#)
 - getEntity, [10](#)
 - getHealth, [11](#)
 - getType, [11](#)
 - getValue, [12](#)
 - setAlliance, [12](#)
 - setDamage, [13](#)
 - setEntity, [13](#)
 - setHealth, [14](#)
 - setType, [15](#)
 - setValue, [15](#)
 - takeDamage, [16](#)
 - value, [17](#)
- addReserveEntity
 - Alliance, [24](#)
- AerialType, [17](#)
 - AerialType, [18](#)
 - clone, [18](#)
 - getTypeDesc, [18](#)
- Aggressive, [19](#)
 - Aggressive, [20](#)
 - clone, [20](#)
 - performStrat, [20](#)
- Alliance, [21](#)
 - ~Alliance, [22](#)
 - addCountry, [23](#)
 - addFactory, [23](#)
 - addReserveEntity, [24](#)
 - Alliance, [22](#)
 - clone, [25](#)
 - considerPeace, [25](#)
 - getActive, [25](#)
 - getID, [26](#)
 - getReserveEntities, [26](#)
 - numRemainingEntities, [27](#)
 - offerPeace, [27](#)
 - runFactories, [27](#)
 - setActiveStatus, [28](#)
 - setNegotiator, [28](#)
 - surrender, [29](#)
- AquaticType, [30](#)
 - AquaticType, [30](#)
 - clone, [30](#)
 - getTypeDesc, [31](#)
- Area, [32](#)
 - ~Area, [32](#)
 - addGeneral, [33](#)
 - Area, [32](#)
 - clone, [33](#)
 - getAreaName, [33](#)
 - isKeyPoint, [34](#)
 - simulateBattle, [34](#)
- Armour, [34](#)
 - Armour, [35](#)
 - clone, [35](#)

- dealDamage, 36
- takeDamage, 36
- changeWeather
 - KeyPoint, 67
- clearBattlefield
 - KeyPoint, 68
- clearSaveList
 - SaveArchive, 93
- clearStats
 - RoundStats, 90
- clone
 - AddOn, 9
 - AerialType, 18
 - Aggressive, 20
 - Alliance, 25
 - AquaticType, 30
 - Area, 33
 - Armour, 35
 - Cloudy, 38
 - Country, 41
 - Defensive, 44
 - Entity, 51
 - Factory, 59
 - KeyPoint, 69
 - Passive, 78
 - Personnel, 80
 - PersonnelFactory, 83
 - Piercing, 85
 - Rainy, 88
 - Strategy, 97
 - Sunny, 99
 - Support, 102
 - SupportFactory, 105
 - TerrainType, 107
 - Type, 109
 - Vehicle, 111
 - VehicleFactory, 114
 - WarEngineState, 123
 - WarTheatre, 128
 - Weather, 131
- Cloudy, 37
 - clone, 38
 - Cloudy, 38
 - getWeather, 38
 - handleChange, 39
- considerPeace
 - Alliance, 25
- Country, 40
 - clone, 41
 - Country, 40
 - getID, 41
 - getName, 41
 - setID, 42
 - setName, 42
- createEntity
 - Factory, 59
 - PersonnelFactory, 83
 - SupportFactory, 105
 - VehicleFactory, 115
- damageDone
 - RoundStats, 90
- dealDamage
 - AddOn, 9
 - Armour, 36
 - Entity, 51
 - Personnel, 80
 - Piercing, 85
 - Support, 102
 - Vehicle, 112
- Defensive, 43
 - clone, 44
 - Defensive, 43
 - performStrat, 44
- deleteSave
 - SaveArchive, 94
- EasySetup, 45
 - EasySetup, 45
 - loadPrevSave, 45
 - loadSpecificSave, 46
 - runSimulation, 46
 - saveSimulationSetup, 46
 - setupSimulation, 46
- Entity, 50
 - clone, 51
 - dealDamage, 51
 - Entity, 51
 - getAlliance, 52
 - getDamage, 52
 - getHealth, 53
 - getType, 53
 - setAlliance, 54
 - setDamage, 55
 - setHealth, 55
 - setType, 56
 - takeDamage, 56
- entity
 - AddOn, 17
- entityMovementInformation
 - RoundStats, 90
- Factory, 58
 - ~Factory, 59
 - clone, 59
 - createEntity, 59
 - Factory, 58
 - getAddOn, 60
 - getType, 60
 - setAddOns, 60
 - setType, 61
- General, 62
 - General, 62
 - getAlliance, 62
 - initiateStrategy, 63
 - setStrategy, 63

getActive
 Alliance, 25
getAddOn
 Factory, 60
getAlliance
 AddOn, 10
 Entity, 52
 General, 62
getAlliances
 WarEngineState, 123
getArea
 WarEngineState, 124
getAreaName
 Area, 33
getDamage
 AddOn, 10
 Entity, 52
getEntity
 AddOn, 10
getHealth
 AddOn, 11
 Entity, 53
getID
 Alliance, 26
 Country, 41
getInstance
 WarEngine, 118
getLastSave
 SaveArchive, 94
getMultiplier
 Weather, 131
getName
 Country, 41
getNumAlliances
 Negotiator, 75
getReserveEntities
 Alliance, 26
getSave
 SaveArchive, 95
getType
 AddOn, 11
 Entity, 53
 Factory, 60
getTypeDesc
 AerialType, 18
 AquaticType, 31
 TerrainType, 107
 Type, 109
getValue
 AddOn, 12
getWeather
 Cloudy, 38
 KeyPoint, 69
 Rainy, 88
 Sunny, 99
 Weather, 132
handleChange
 Cloudy, 39
 Rainy, 88
 Sunny, 100
 Weather, 132
initiateStrategy
 General, 63
isKeyPoint
 Area, 34
 KeyPoint, 69
 WarTheatre, 129
KeyPoint, 64
 ~KeyPoint, 66
 addEntity, 66
 addGeneral, 67
 changeWeather, 67
 clearBattlefield, 68
 clone, 69
 getWeather, 69
 isKeyPoint, 69
 KeyPoint, 65, 66
 moveEntitiesInto, 70
 moveEntitiesOutOf, 71
 removeGeneral, 71
 setWeather, 72
 simulateBattle, 73
keyPointInformation
 RoundStats, 91
loadPrevSave
 EasySetup, 45
loadSave
 WarEngine, 118
loadSpecificSave
 EasySetup, 46
moveEntitiesInto
 KeyPoint, 70
moveEntitiesOutOf
 KeyPoint, 71
multiplier
 Weather, 133
Negotiator, 74
 ~Negotiator, 74
 addAlliance, 75
 getNumAlliances, 75
 Negotiator, 74
 removeAlliance, 75
 sendPeace, 76
numContestedPoints
 RoundStats, 91
numEntitiesCreated
 RoundStats, 91
numEntitiesDestroyed
 RoundStats, 91
numLosingPoints
 RoundStats, 91
numRemainingEntities

- Alliance, [27](#)
- numWinningPoints
 - RoundStats, [91](#)
- offerPeace
 - Alliance, [27](#)
- operator=
 - WarEngine, [119](#)
- Passive, [77](#)
 - clone, [78](#)
 - Passive, [77](#)
 - performStrat, [78](#)
- performStrat
 - Aggressive, [20](#)
 - Defensive, [44](#)
 - Passive, [78](#)
 - Strategy, [97](#)
- Personnel, [79](#)
 - clone, [80](#)
 - dealDamage, [80](#)
 - Personnel, [79](#)
 - takeDamage, [81](#)
- PersonnelFactory, [82](#)
 - clone, [83](#)
 - createEntity, [83](#)
 - PersonnelFactory, [82](#)
- Piercing, [84](#)
 - clone, [85](#)
 - dealDamage, [85](#)
 - Piercing, [85](#)
 - takeDamage, [86](#)
- Rainy, [87](#)
 - clone, [88](#)
 - getWeather, [88](#)
 - handleChange, [88](#)
 - Rainy, [88](#)
- removeAlliance
 - Negotiator, [75](#)
- removeGeneral
 - KeyPoint, [71](#)
- RoundStats, [89](#)
 - clearStats, [90](#)
 - damageDone, [90](#)
 - entityMovementInformation, [90](#)
 - keyPointInformation, [91](#)
 - numContestedPoints, [91](#)
 - numEntitiesCreated, [91](#)
 - numEntitiesDestroyed, [91](#)
 - numLosingPoints, [91](#)
 - numWinningPoints, [91](#)
 - toString, [90](#)
- runFactories
 - Alliance, [27](#)
- runSimulation
 - EasySetup, [46](#)
- SaveArchive, [92](#)
- addNewSave, [93](#)
- clearSaveList, [93](#)
- deleteSave, [94](#)
- getLastSave, [94](#)
- getSave, [95](#)
- SaveArchive, [92](#)
- saveSimulationSetup
 - EasySetup, [46](#)
- saveState
 - WarEngine, [119](#)
- sendPeace
 - Negotiator, [76](#)
- setActiveStatus
 - Alliance, [28](#)
- setAddOns
 - Factory, [60](#)
- setAlliance
 - AddOn, [12](#)
 - Entity, [54](#)
- setAlliances
 - WarEngineState, [124](#)
- setArea
 - WarEngineState, [125](#)
- setDamage
 - AddOn, [13](#)
 - Entity, [55](#)
- setEntity
 - AddOn, [13](#)
- setHealth
 - AddOn, [14](#)
 - Entity, [55](#)
- setID
 - Country, [42](#)
- setName
 - Country, [42](#)
- setNegotiator
 - Alliance, [28](#)
- setStrategy
 - General, [63](#)
- setType
 - AddOn, [15](#)
 - Entity, [56](#)
 - Factory, [61](#)
- setupSimulation
 - EasySetup, [46](#)
- setValue
 - AddOn, [15](#)
- setWarTheatre
 - WarEngine, [119](#)
- setWeather
 - KeyPoint, [72](#)
- simulate
 - WarEngine, [120](#)
- simulateBattle
 - Area, [34](#)
 - KeyPoint, [73](#)
 - WarTheatre, [129](#)
- Strategy, [96](#)

- ~Strategy, 97
 - clone, 97
 - performStrat, 97
 - Strategy, 97
 - strategy, 98
- strategy
 - Strategy, 98
- Sunny, 98
 - clone, 99
 - getWeather, 99
 - handleChange, 100
 - Sunny, 99
- Support, 101
 - clone, 102
 - dealDamage, 102
 - Support, 101
 - takeDamage, 103
- SupportFactory, 104
 - clone, 105
 - createEntity, 105
 - SupportFactory, 104
- surrender
 - Alliance, 29
- takeDamage
 - AddOn, 16
 - Armour, 36
 - Entity, 56
 - Personnel, 81
 - Piercing, 86
 - Support, 103
 - Vehicle, 112
- TerrainType, 106
 - clone, 107
 - getTypeDesc, 107
 - TerrainType, 107
- toString
 - RoundStats, 90
- Type, 108
 - clone, 109
 - getTypeDesc, 109
 - Type, 109
- value
 - AddOn, 17
- Vehicle, 110
 - clone, 111
 - dealDamage, 112
 - takeDamage, 112
 - Vehicle, 111
- VehicleFactory, 113
 - clone, 114
 - createEntity, 115
 - VehicleFactory, 114
- WarEngine, 116
 - ~WarEngine, 117
 - addAlliance, 117
 - getInstance, 118
 - loadSave, 118
 - operator=, 119
 - saveState, 119
 - setWarTheatre, 119
 - simulate, 120
 - WarEngine, 116, 117
 - WarEngineMemento, 121
 - WarEngineState, 126
- WarEngineMemento, 121
 - WarEngine, 121
- WarEngineState, 122
 - ~WarEngineState, 123
 - clone, 123
 - getAlliances, 123
 - getArea, 124
 - setAlliances, 124
 - setArea, 125
 - WarEngine, 126
 - WarEngineState, 123
- WarTheatre, 126
 - ~WarTheatre, 127
 - addArea, 127
 - addGeneral, 128
 - clone, 128
 - isKeyPoint, 129
 - simulateBattle, 129
 - WarTheatre, 127
- Weather, 130
 - ~Weather, 131
 - clone, 131
 - getMultiplier, 131
 - getWeather, 132
 - handleChange, 132
 - multiplier, 133
 - Weather, 131