

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	8
4.1.3.1 clone()	8
4.1.3.2 dealDamage()	8
4.1.3.3 getEntity()	9
4.1.3.4 getValue()	9
4.1.3.5 setEntity()	9
4.1.3.6 setValue()	10
4.1.3.7 takeDamage()	10
4.1.4 Member Data Documentation	11
4.1.4.1 entity	11
4.1.4.2 value	11
4.2 AerialType Class Reference	11
4.2.1 Detailed Description	12
4.2.2 Constructor & Destructor Documentation	12
4.2.2.1 AerialType()	12
4.2.3 Member Function Documentation	12
4.2.3.1 clone()	12
4.2.3.2 getTypeDesc()	12
4.3 Aggressive Class Reference	13
4.3.1 Detailed Description	13
4.3.2 Constructor & Destructor Documentation	13
4.3.2.1 Aggressive()	13
4.3.3 Member Function Documentation	13
4.3.3.1 clone()	14
4.3.3.2 performStrat()	14
4.4 Alliance Class Reference	15
4.4.1 Detailed Description	15
4.4.2 Constructor & Destructor Documentation	15
4.4.2.1 Alliance() [1/2]	16

4.4.2.2 Alliance() [2/2]	16
4.4.2.3 ~Alliance()	16
4.4.3 Member Function Documentation	17
4.4.3.1 addCountry()	17
4.4.3.2 addFactory()	17
4.4.3.3 addReserveEntity()	18
4.4.3.4 clone()	18
4.4.3.5 considerPeace()	19
4.4.3.6 getActive()	19
4.4.3.7 getID()	20
4.4.3.8 getReserveEntities()	20
4.4.3.9 offerPeace()	21
4.4.3.10 setActiveStatus()	21
4.4.3.11 setNegotiator()	22
4.4.3.12 surrender()	22
4.5 AquaticType Class Reference	23
4.5.1 Detailed Description	23
4.5.2 Constructor & Destructor Documentation	23
4.5.2.1 AquaticType()	23
4.5.3 Member Function Documentation	23
4.5.3.1 clone()	24
4.5.3.2 getTypeDesc()	24
4.6 Area Class Reference	24
4.6.1 Detailed Description	25
4.6.2 Constructor & Destructor Documentation	25
4.6.2.1 Area()	25
4.6.2.2 ~Area()	25
4.6.3 Member Function Documentation	25
4.6.3.1 clone()	26
4.6.3.2 getAreaName()	26
4.6.3.3 isKeyPoint()	26
4.6.3.4 simulateBattle()	26
4.7 Armour Class Reference	27
4.7.1 Detailed Description	27
4.7.2 Constructor & Destructor Documentation	27
4.7.2.1 Armour()	27
4.7.3 Member Function Documentation	28
4.7.3.1 clone()	28
4.7.3.2 dealDamage()	28
4.7.3.3 takeDamage()	29
4.8 Cloudy Class Reference	30
4.8.1 Detailed Description	30

4.8.2 Constructor & Destructor Documentation	30
4.8.2.1 Cloudy()	30
4.8.3 Member Function Documentation	30
4.8.3.1 clone()	31
4.8.3.2 getWeather()	31
4.8.3.3 handleChange()	31
4.9 Country Class Reference	32
4.9.1 Detailed Description	32
4.9.2 Constructor & Destructor Documentation	32
4.9.2.1 Country()	33
4.9.3 Member Function Documentation	33
4.9.3.1 clone()	33
4.9.3.2 getID()	33
4.9.3.3 getName()	34
4.9.3.4 setID()	34
4.9.3.5 setName()	34
4.10 Defensive Class Reference	35
4.10.1 Detailed Description	35
4.10.2 Constructor & Destructor Documentation	36
4.10.2.1 Defensive()	36
4.10.3 Member Function Documentation	36
4.10.3.1 clone()	36
4.10.3.2 performStrat()	36
4.11 Entity Class Reference	37
4.11.1 Detailed Description	38
4.11.2 Constructor & Destructor Documentation	38
4.11.2.1 Entity() [1/2]	38
4.11.2.2 Entity() [2/2]	38
4.11.3 Member Function Documentation	39
4.11.3.1 clone()	39
4.11.3.2 dealDamage()	39
4.11.3.3 getAlliance()	39
4.11.3.4 getDamage()	40
4.11.3.5 getHealth()	40
4.11.3.6 getType()	40
4.11.3.7 setAlliance()	41
4.11.3.8 setDamage()	41
4.11.3.9 setHealth()	42
4.11.3.10 setType()	42
4.11.3.11 takeDamage()	43
4.12 Factory Class Reference	43
4.12.1 Detailed Description	44

4.12.2 Constructor & Destructor Documentation	44
4.12.2.1 Factory()	44
4.12.2.2 ~Factory()	44
4.12.3 Member Function Documentation	44
4.12.3.1 clone()	45
4.12.3.2 createEntity()	45
4.12.3.3 getAddOn()	45
4.12.3.4 getType()	45
4.12.3.5 setAddOns()	46
4.12.3.6 setType()	46
4.13 General Class Reference	47
4.13.1 Detailed Description	47
4.13.2 Constructor & Destructor Documentation	47
4.13.2.1 General()	47
4.13.3 Member Function Documentation	48
4.13.3.1 getAlliance()	48
4.13.3.2 initiateStrategy()	48
4.13.3.3 setStrategy()	49
4.14 KeyPoint Class Reference	50
4.14.1 Detailed Description	50
4.14.2 Constructor & Destructor Documentation	51
4.14.2.1 KeyPoint() [1/2]	51
4.14.2.2 KeyPoint() [2/2]	51
4.14.2.3 ~KeyPoint()	51
4.14.3 Member Function Documentation	52
4.14.3.1 addEntity()	52
4.14.3.2 addGeneral()	52
4.14.3.3 changeWeather()	53
4.14.3.4 clearBattlefield()	53
4.14.3.5 clone()	54
4.14.3.6 getWeather()	54
4.14.3.7 isKeyPoint()	54
4.14.3.8 moveEntitiesInto()	55
4.14.3.9 moveEntitiesOutOf()	55
4.14.3.10 removeGeneral()	56
4.14.3.11 setWeather()	56
4.14.3.12 simulateBattle()	57
4.15 Negotiator Class Reference	58
4.15.1 Detailed Description	58
4.15.2 Constructor & Destructor Documentation	58
4.15.2.1 Negotiator()	58
4.15.2.2 ~Negotiator()	58

4.15.3 Member Function Documentation	59
4.15.3.1 addAlliance()	59
4.15.3.2 getNumAlliances()	59
4.15.3.3 removeAlliance()	60
4.15.3.4 sendPeace()	60
4.16 Passive Class Reference	61
4.16.1 Detailed Description	61
4.16.2 Constructor & Destructor Documentation	61
4.16.2.1 Passive()	62
4.16.3 Member Function Documentation	62
4.16.3.1 clone()	62
4.16.3.2 performStrat()	62
4.17 Personnel Class Reference	63
4.17.1 Detailed Description	63
4.17.2 Constructor & Destructor Documentation	63
4.17.2.1 Personnel()	63
4.17.3 Member Function Documentation	64
4.17.3.1 clone()	64
4.17.3.2 dealDamage()	64
4.17.3.3 takeDamage()	65
4.18 PersonnelFactory Class Reference	66
4.18.1 Detailed Description	66
4.18.2 Constructor & Destructor Documentation	66
4.18.2.1 PersonnelFactory()	66
4.18.3 Member Function Documentation	67
4.18.3.1 clone()	67
4.18.3.2 createEntity()	67
4.19 Piercing Class Reference	68
4.19.1 Detailed Description	69
4.19.2 Constructor & Destructor Documentation	69
4.19.2.1 Piercing()	69
4.19.3 Member Function Documentation	69
4.19.3.1 clone()	69
4.19.3.2 dealDamage()	70
4.19.3.3 takeDamage()	70
4.20 Rainy Class Reference	71
4.20.1 Detailed Description	71
4.20.2 Constructor & Destructor Documentation	72
4.20.2.1 Rainy()	72
4.20.3 Member Function Documentation	72
4.20.3.1 clone()	72
4.20.3.2 getWeather()	72

4.20.3.3 handleChange()	73
4.21 SaveArchive Class Reference	73
4.21.1 Detailed Description	74
4.21.2 Constructor & Destructor Documentation	74
4.21.2.1 SaveArchive()	74
4.21.3 Member Function Documentation	74
4.21.3.1 addNewSave()	74
4.21.3.2 clearSaveList()	75
4.21.3.3 deleteSave()	75
4.21.3.4 getLastSave()	76
4.21.3.5 getSave()	77
4.22 Strategy Class Reference	77
4.22.1 Detailed Description	78
4.22.2 Constructor & Destructor Documentation	78
4.22.2.1 Strategy()	78
4.22.2.2 ~Strategy()	78
4.22.3 Member Function Documentation	78
4.22.3.1 clone()	79
4.22.3.2 performStrat()	79
4.22.4 Member Data Documentation	79
4.22.4.1 strategy	79
4.23 Sunny Class Reference	79
4.23.1 Detailed Description	80
4.23.2 Constructor & Destructor Documentation	80
4.23.2.1 Sunny()	80
4.23.3 Member Function Documentation	80
4.23.3.1 clone()	80
4.23.3.2 getWeather()	81
4.23.3.3 handleChange()	81
4.24 Support Class Reference	82
4.24.1 Detailed Description	82
4.24.2 Constructor & Destructor Documentation	82
4.24.2.1 Support()	82
4.24.3 Member Function Documentation	83
4.24.3.1 clone()	83
4.24.3.2 dealDamage()	83
4.24.3.3 takeDamage()	84
4.25 SupportFactory Class Reference	85
4.25.1 Detailed Description	85
4.25.2 Constructor & Destructor Documentation	85
4.25.2.1 SupportFactory()	85
4.25.3 Member Function Documentation	86

4.25.3.1 clone()	86
4.25.3.2 createEntity()	86
4.26 TerrainType Class Reference	87
4.26.1 Detailed Description	87
4.26.2 Constructor & Destructor Documentation	88
4.26.2.1 TerrainType()	88
4.26.3 Member Function Documentation	88
4.26.3.1 clone()	88
4.26.3.2 getTypeDesc()	88
4.27 Type Class Reference	89
4.27.1 Detailed Description	89
4.27.2 Constructor & Destructor Documentation	89
4.27.2.1 Type()	89
4.27.3 Member Function Documentation	89
4.27.3.1 getTypeDesc()	90
4.28 Vehicle Class Reference	90
4.28.1 Detailed Description	90
4.28.2 Constructor & Destructor Documentation	90
4.28.2.1 Vehicle()	90
4.28.3 Member Function Documentation	91
4.28.3.1 clone()	91
4.28.3.2 dealDamage()	91
4.28.3.3 takeDamage()	92
4.29 VehicleFactory Class Reference	93
4.29.1 Detailed Description	93
4.29.2 Constructor & Destructor Documentation	93
4.29.2.1 VehicleFactory()	93
4.29.3 Member Function Documentation	94
4.29.3.1 clone()	94
4.29.3.2 createEntity()	94
4.30 WarEngine Class Reference	95
4.30.1 Detailed Description	96
4.30.2 Constructor & Destructor Documentation	96
4.30.2.1 WarEngine() [1/2]	96
4.30.2.2 WarEngine() [2/2]	96
4.30.2.3 ~WarEngine()	96
4.30.3 Member Function Documentation	97
4.30.3.1 getInstance()	97
4.30.3.2 loadState()	97
4.30.3.3 operator=()	98
4.30.3.4 saveState()	98
4.30.3.5 setWarTheatre()	98

4.30.3.6 simulate()	99
4.31 WarEngineMemento Class Reference	99
4.31.1 Detailed Description	99
4.31.2 Friends And Related Function Documentation	100
4.31.2.1 WarEngine	100
4.32 WarEngineState Class Reference	100
4.32.1 Detailed Description	100
4.32.2 Constructor & Destructor Documentation	101
4.32.2.1 WarEngineState()	101
4.32.2.2 ~WarEngineState()	101
4.32.3 Member Function Documentation	101
4.32.3.1 clone()	101
4.32.3.2 getAlliances()	102
4.32.3.3 getArea()	102
4.32.3.4 setAlliances()	102
4.32.3.5 setArea()	103
4.33 WarTheatre Class Reference	104
4.33.1 Detailed Description	104
4.33.2 Constructor & Destructor Documentation	104
4.33.2.1 WarTheatre()	105
4.33.2.2 ~WarTheatre()	105
4.33.3 Member Function Documentation	105
4.33.3.1 addArea()	105
4.33.3.2 clone()	106
4.33.3.3 isKeyPoint()	106
4.33.3.4 simulateBattle()	107
4.34 Weather Class Reference	107
4.34.1 Detailed Description	108
4.34.2 Constructor & Destructor Documentation	108
4.34.2.1 Weather()	108
4.34.2.2 ~Weather()	108
4.34.3 Member Function Documentation	108
4.34.3.1 clone()	109
4.34.3.2 getMultiplier()	109
4.34.3.3 getWeather()	109
4.34.3.4 handleChange()	109
4.34.4 Member Data Documentation	109
4.34.4.1 multiplier	109
5 File Documentation	111
5.1 AddOn.cpp	111
5.2 AddOn.h	111

5.3 AddOnTest.h	112
5.4 AerialType.cpp	114
5.5 AerialType.h	114
5.6 Aggressive.cpp	115
5.7 Aggressive.h	115
5.8 Alliance.cpp	115
5.9 Alliance.h	116
5.10 AquaticType.cpp	117
5.11 AquaticType.h	117
5.12 Area.cpp	117
5.13 Area.h	118
5.14 Armour.cpp	118
5.15 Armour.h	118
5.16 Cloudy.cpp	119
5.17 Cloudy.h	119
5.18 Country.cpp	119
5.19 Country.h	120
5.20 Defensive.cpp	120
5.21 Defensive.h	120
5.22 Entity.cpp	121
5.23 Entity.h	121
5.24 Factory.cpp	122
5.25 Factory.h	122
5.26 General.cpp	123
5.27 General.h	123
5.28 KeyPoint.cpp	123
5.29 KeyPoint.h	125
5.30 Negotiator.cpp	125
5.31 Negotiator.h	126
5.32 NegotiatorTest.h	126
5.33 Passive.cpp	127
5.34 Passive.h	127
5.35 Personnel.cpp	128
5.36 Personnel.h	128
5.37 PersonnelFactory.cpp	128
5.38 PersonnelFactory.h	129
5.39 Piercing.cpp	129
5.40 Piercing.h	129
5.41 Rainy.cpp	129
5.42 Rainy.h	130
5.43 SaveArchive.cpp	130
5.44 SaveArchive.h	131

5.45 Strategy.cpp	131
5.46 Strategy.h	131
5.47 Sunny.cpp	132
5.48 Sunny.h	132
5.49 Support.cpp	132
5.50 Support.h	132
5.51 SupportFactory.cpp	133
5.52 SupportFactory.h	133
5.53 TerrainType.cpp	133
5.54 TerrainType.h	133
5.55 testmain.cpp	134
5.56 Type.cpp	134
5.57 Type.h	134
5.58 Vehicle.cpp	134
5.59 Vehicle.h	135
5.60 VehicleFactory.cpp	135
5.61 VehicleFactory.h	135
5.62 WarEngine.cpp	135
5.63 WarEngine.h	136
5.64 WarEngineMemento.cpp	136
5.65 WarEngineMemento.h	137
5.66 WarEngineState.cpp	137
5.67 WarEngineState.h	138
5.68 WarTheatre.cpp	138
5.69 WarTheatre.h	139
5.70 Weather.cpp	139
5.71 Weather.h	139

Index	141
--------------	------------

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Alliance	15
Area	24
KeyPoint	50
WarTheatre	104
Country	32
Entity	37
AddOn	7
Armour	27
Piercing	68
Personnel	63
Support	82
Vehicle	90
Factory	43
PersonnelFactory	66
SupportFactory	85
VehicleFactory	93
General	47
Negotiator	58
SaveArchive	73
Strategy	77
Aggressive	13
Defensive	35
Passive	61
Type	89
AerialType	11
AquaticType	23
TerrainType	87
WarEngine	95
WarEngineMemento	99
WarEngineState	100
Weather	107
Cloudy	30
Rainy	71
Sunny	79

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	11
Aggressive		13
Alliance		15
AquaticType		
AquaticType	class	23
Area		24
Armour		
Armour	class	27
Cloudy		30
Country		32
Defensive		35
Entity		
Entity	class	37
Factory		
Factory	class	43
General		47
KeyPoint		
Keypoint	class	50
Negotiator		58
Passive		61
Personnel		
Personnel	class	63
PersonnelFactory		
PersonnelFactory	class	66
Piercing		
Piercing	class	68
Rainy		71
SaveArchive		
Stores a list of mementos containing simulation state		73
Strategy		77
Sunny		79
Support		
Support	class	82

SupportFactory	
SupportFactory class	85
TerrainType	
TerrainType class	87
Type	
Type class	89
Vehicle	
Vehicle class	90
VehicleFactory	
VehicleFactory class	93
WarEngine	95
WarEngineMemento	99
WarEngineState	
Class for storing current state of entire simulation	100
WarTheatre	104
Weather	107

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	??
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	??
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

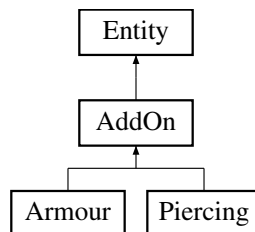
Class Documentation

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.
- virtual void [takeDamage](#) (int damage)=0
- virtual void [dealDamage](#) ([Entity](#) *entity)=0
- virtual [AddOn](#) * [clone](#) ()=0

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

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

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

```
00003      : Entity() {
00004      this->value = value;
00005      entity = NULL;
00006 }
```

4.1.3 Member Function Documentation

4.1.3.1 clone()

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

Implements [Entity](#).

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

4.1.3.2 dealDamage()

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

Implements [Entity](#).

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

4.1.3.3 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 20 of file [AddOn.cpp](#).

```
00020 {  
00021     return this->entity;  
00022 }
```

4.1.3.4 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 12 of file [AddOn.cpp](#).

```
00012 {  
00013     return value;  
00014 }
```

4.1.3.5 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 16 of file [AddOn.cpp](#).

```
00016 {
00017     this->entity = entity;
00018 }
```

4.1.3.6 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 8 of file [AddOn.cpp](#).

```
00008 {
00009     this->value = value;
00010 }
```

4.1.3.7 takeDamage()

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

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:

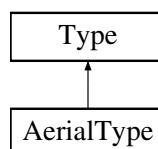
- [AddOn.h](#)
- [AddOn.cpp](#)

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](#) ()

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]
```

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](#).

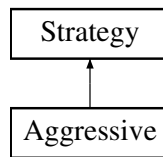
```
00005 {  
00006     return "Aerial";  
00007 }
```

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 [Aggressive 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 [Aggressive Strategy](#) object.

Returns

Strategy* The clone of the [Aggressive 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.

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 a copy of an [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](#) (bool active)
Sets variable active to the passed in parameter.
- int [getActive](#) ()

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 8 of file [Alliance.cpp](#).

```
00008      {
00009          this->active = 1;
00010          this->aID = totalNum++;
00011          this->negotiator = NULL;
00012          srand(time(0));
00013      }
```

4.4.2.2 Alliance() [2/2]

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

Instantiates a copy of an [Alliance](#).

Parameters

<i>alliance</i>	must be an alliance instance
-----------------	------------------------------

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

```
00015      {
00016          this->active = alliance.active;
00017          this->aID = alliance.aID;
00018
00019          for (int i = 0; i < alliance.members.size(); i++)
00020              this->addCountry(alliance.members[i]->clone());
00021
00022          for (int i = 0; i < alliance.production.size(); i++)
00023              this->addFactory(alliance.production[i]->clone());
00024
00025          for (int i = 0; i < alliance.reserveEntities.size(); i++)
00026              this->addReserveEntity(alliance.reserveEntities[i]->clone());
00027
00028          this->negotiator = NULL;
00029      }
```

4.4.2.3 ~Alliance()

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

Destructor for the [Alliance](#) object.

Definition at line 31 of file [Alliance.cpp](#).

```
00031      {
00032
00033          this->negotiator->removeAlliance(this);
00034
00035          if (this->negotiator->getNumAlliances() == 1)
00036              delete this->negotiator;
00037      }
```

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

nation	must be an Country*
--------	---------------------

Returns

void

Definition at line 43 of file Alliance.cpp.

```
00043                                     {
00044     members.push_back(nation);
00045 }
```

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 65 of file [Alliance.cpp](#).

```
00065                                     {  
00066     production.push_back(factory);  
00067 }
```

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 57 of file [Alliance.cpp](#).

```
00057                                     {  
00058     reserveEntities.push_back(entity);  
00059 }
```

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 94 of file [Alliance.cpp](#).

```
00094         {  
00095     return new Alliance (*this);  
00096 }
```

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 61 of file [Alliance.cpp](#).

```
00061     {  
00062     return (rand() % 2 == 0);  
00063 }
```

4.4.3.6 getActive()

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

Definition at line 90 of file [Alliance.cpp](#).

```
00090     {  
00091     return active;  
00092 }
```

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 75 of file [Alliance.cpp](#).

```
00075     {
00076     return this->aID;
00077 }
```

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 47 of file [Alliance.cpp](#).

```
00047     {
00048     vector<Entity*> out;
00049     for (int i = 0; i < number && i < reserveEntities.size(); i++) {
00050         out.push_back(reserveEntities[i]);
```



```

00051         reserveEntities.erase(reserveEntities.begin() + i);
00052     }
00053
00054     return out;
00055 }

```

4.4.3.9 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 79 of file [Alliance.cpp](#).

```

00079         {
00080
00081         if (this->negotiator->sendPeace(this)) //Send the peace deal to all the alliances fighting against
00082         {
00083             this->active = 3; //Number 3 means that Alliance chose to peacefully pull out of war
00084             return true;
00085         }
00086
00087         return false;
00088     }

```

4.4.3.10 setActiveStatus()

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

Sets variable active to the passed in parameter.

PreConditions:

- active must be an a bool

PostConditions:

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

Parameters

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

4.4.3.11 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

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

Returns

void

Definition at line 39 of file Alliance.cpp.

```
00039      {
00040          this->negotiator = negotiator;
00041      }
```

4.4.3.12 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 69 of file Alliance.cpp.

```
00069      {
00070          this->active = 2; //Number 2 means that Alliance has surrendered
00071
00072          this->negotiator->removeAlliance(this);
00073      }
```

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

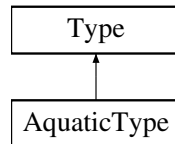
- Alliance.h
- Alliance.cpp

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](#) ()

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]
```

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](#).

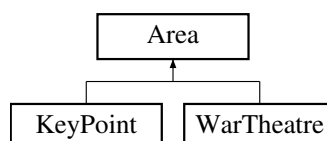
```
00007     {  
00008     return "Aquatic";  
00009 }
```

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

4.6.1 Detailed Description

Definition at line 6 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 clone()

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

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

4.6.3.2 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 }
```

4.6.3.3 isKeyPoint()

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

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

4.6.3.4 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:

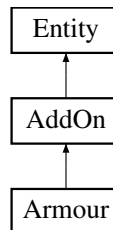
- [Area.h](#)
- [Area.cpp](#)

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 3 of file [Armour.cpp](#).

```
00003 : 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 17 of file [Armour.cpp](#).

```
00017 {  
00018     Armour* armour = new Armour(value);  
00019     armour->setEntity(entity->clone());  
00020     return armour;  
00021 }
```

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 13 of file [Armour.cpp](#).

```
00013 {  
00014     this->entity->dealDamage(entity);  
00015 }
```

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

Parameters

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

Returns

void

Implements [AddOn](#).

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

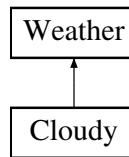
```
00005 {  
00006     if (value > 0) {  
00007         value -= damage;  
00008     } else {  
00009         entity->takeDamage(damage);  
00010     }  
00011 }
```

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](#) *k)
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 * k ) [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](#) ()
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 ( )
```

Instantiates the [Country](#).

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

```
00005 {}
```

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 8 of file [Country.cpp](#).

```
00008 {
00009
00010     Country* countryClone = new Country();
00011     countryClone->setID(this->id);
00012     countryClone->setName(this->name);
00013
00014     return countryClone;
00015 }
```

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 29 of file [Country.cpp](#).

```
00029 {
00030     return this->id;
00031 }
```

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 25 of file [Country.cpp](#).

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

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>	
-----------	--

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

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

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>	
-------------	--

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

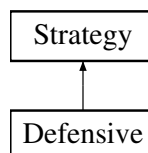
```
00021 {
00022     this->name = name;
00023 }
```

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 Deffensive [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:

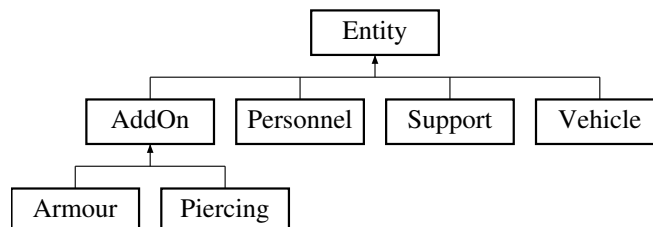
- Defensive.h
- Defensive.cpp

4.11 Entity Class Reference

[Entity](#) class.

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

Inheritance diagram for Entity:



Public Member Functions

- [Entity](#) ()
Instantiates the entity.
- [Entity](#) (Type *type, int health, int damage)
Instantiates the entity.
- 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
- virtual void [dealDamage](#) ([Entity](#) *entity)=0
- virtual [Entity](#) * [clone](#) ()=0

4.11.1 Detailed Description

[Entity](#) class.

Used to simulate war entity objects.

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

4.11.2 Constructor & Destructor Documentation

4.11.2.1 [Entity\(\)](#) [1/2]

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

Instantiates the entity.

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

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

4.11.2.2 [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 }
```

4.11.3 Member Function Documentation

4.11.3.1 clone()

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

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

4.11.3.2 dealDamage()

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

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

4.11.3.3 getAlliance()

```
Alliance * Entity::getAlliance ( )
```

Returns entities alliance.

Postconditions:

- Returns the alliance

Returns

Type* The alliance of the entity object

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

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

4.11.3.4 getDamage()

```
int Entity::getDamage ( )
```

Returns entities damage.

Postconditions:

- Returns the damage

Returns

int The damage of the entity object

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

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

4.11.3.5 getHealth()

```
int Entity::getHealth ( )
```

Returns entities health.

Postconditions:

- Returns the health

Returns

int The health of the entity object

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

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

4.11.3.6 getType()

```
Type * Entity::getType ( )
```

Returns entities type state.

Postconditions:

- Returns the type

Returns

Type* The type state of the entity object

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

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

4.11.3.7 setAlliance()

```
void Entity::setAlliance (
    Alliance * alliance )
```

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

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

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

4.11.3.8 setDamage()

```
void Entity::setDamage (
    int damage )
```

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

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

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

4.11.3.9 setHealth()

```
void Entity::setHealth (  
    int health )
```

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

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

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

4.11.3.10 setType()

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

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

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

```

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

```

4.11.3.11 takeDamage()

```

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

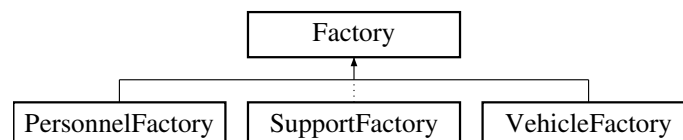
```

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

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

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

4.12 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

4.12.1 Detailed Description

[Factory](#) class.

Used to instantiate [Entity](#) objects.

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

4.12.2 Constructor & Destructor Documentation

4.12.2.1 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 }
```

4.12.2.2 ~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 }
```

4.12.3 Member Function Documentation

4.12.3.1 clone()

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

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

4.12.3.2 createEntity()

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

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

4.12.3.3 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     }
```

4.12.3.4 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     }
```

4.12.3.5 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 }
```

4.12.3.6 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](#)

4.13 General Class Reference

Public Member Functions

- [General](#) ([Alliance](#) *alliance, [Strategy](#) *stratety)
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.

4.13.1 Detailed Description

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

4.13.2 Constructor & Destructor Documentation

4.13.2.1 General()

```

General::General (
    Alliance * alliance,
    Strategy * stratety )

```

Construct a new [General](#) object.

Parameters

<i>alliance</i>	must be an Alliance*
<i>stratety</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     }
```

4.13.3 Member Function Documentation

4.13.3.1 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     }
```

4.13.3.2 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 }
```

4.13.3.3 setStrategy()

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

Set the [Strategy](#) object.

PreConditions:

- 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:

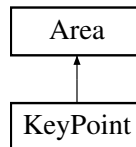
- General.h
- General.cpp

4.14 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](#) ()
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)
- void [removeGeneral](#) ([General](#) *general)
- [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.

4.14.1 Detailed Description

Keypoint class.

Used to emulate strategic positions.

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

4.14.2 Constructor & Destructor Documentation

4.14.2.1 KeyPoint() [1/2]

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

Instantiates the key point.

Parameters

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

4.14.2.2 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 10 of file [KeyPoint.cpp](#).

```
00010                                     : Area(keyPoint.getAreaName()) {
00011     for (int i = 0; i < keyPoint.entities.size(); i++)
00012         this->addEntity(keyPoint.entities[i]->clone());
00013
00014     weather = keyPoint.weather->clone();
00015 }
```

4.14.2.3 ~KeyPoint()

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

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

```
00017     {
00018     for (int i = 0; i < entities.size(); i++)
00019         delete entities[i];
00020
00021     for (int i = 0; i < generals.size(); i++)
00022         delete generals[i];
00023
00024     delete weather;
00025 }
```

4.14.3 Member Function Documentation

4.14.3.1 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 77 of file [KeyPoint.cpp](#).

```
00077                                     {
00078     entities.push_back(entity);
00079 }
```

4.14.3.2 addGeneral()

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

Definition at line 81 of file [KeyPoint.cpp](#).

```
00081                                     {
00082     generals.push_back(general);
00083 }
```


4.14.3.3 changeWeather()

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

Switches the [Weather](#) object to the next state.

Definition at line 104 of file [KeyPoint.cpp](#).

```
00104         {
00105
00106             srand(time(0));
00107
00108             int randomNum = 1 + (rand() % 10);
00109             std::string currWeather = this->weather->getWeather();
00110
00111             if (currWeather == "Sunny" && randomNum > 6) // 60% chance of not changing weather from Sunny and
                staying
00112                 this->weather->handleChange(this);
00113             else if (currWeather == "Cloudy" && randomNum > 3) // 30% chance of not changing weather from
                Cloudy and staying
00114                 this->weather->handleChange(this);
00115             else if (currWeather == "Rainy" && randomNum > 1) // 10% chance of not changing weather from Rainy
                and staying
00116                 this->weather->handleChange(this);
00117
00118
00119 }
```

4.14.3.4 clearBattlefield()

```
void KeyPoint::clearBattlefield ( )
```

Clears the battlefield of all deceased troops.

Postconditions:

- Notify command centers of each troop who is killed

Returns

void

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

```
00045         {
00046             for (vector<Entity*>::iterator it = entities.begin(); it != entities.end(); ++it) {
00047                 if ((*it)->getHealth() <= 0) {
00048                     for (int i = 0; i < generals.size(); i++) {
00049                         if (generals[i]->getAlliance() == (*it)->getAlliance()) {
00050                             generals[i]->initiateStrategy(this);
00051                             delete *it;
00052                             entities.erase(it);
00053                         }
00054                     }
00055                 }
00056             }
00057 }
```

4.14.3.5 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 95 of file [KeyPoint.cpp](#).

```
00095         {  
00096     return new KeyPoint (*this);  
00097 }
```

4.14.3.6 getWeather()

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

The weather at the current state is returned.

Returns

string The weather state

Definition at line 121 of file [KeyPoint.cpp](#).

```
00121     {  
00122     return this->weather->getWeather();  
00123 }
```

4.14.3.7 isKeyPoint()

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

Returns area type.

Postconditions:

- Returns true

Returns

bool The area type

Implements [Area](#).

Definition at line 27 of file [KeyPoint.cpp](#).

```
00027     {  
00028     return true;  
00029 }
```

4.14.3.8 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 59 of file [KeyPoint.cpp](#).

```
00059                                     {
00060     vector<Entity*> troops = alliance->getReserveEntities(numTroops);
00061     for (int i = 0; i < troops.size(); i++)
00062         entities.push_back(troops[i]);
00063 }
```

4.14.3.9 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 65 of file [KeyPoint.cpp](#).

```

00065                                     {
00066     vector<Entity*>::iterator it = entities.begin();
00067     for (int i = 0; i < numTroops && it != entities.end(); i++) {
00068         for (; it != entities.end(); ++it) {
00069             if ((*it)->getAlliance() == alliance) {
00070                 alliance->addReserveEntity(*it);
00071                 entities.erase(it);
00072             }
00073         }
00074     }
00075 }
```

4.14.3.10 removeGeneral()

```

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

Definition at line 85 of file [KeyPoint.cpp](#).

```

00085                                     {
00086     for (vector<General*>::iterator it = generals.begin(); it != generals.end(); ++it) {
00087         if (*it == general) {
00088             delete *it;
00089             generals.erase(it);
00090             return;
00091         }
00092     }
00093 }
```

4.14.3.11 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 99 of file [KeyPoint.cpp](#).

```

00099                                     {
00100         delete this->weather;
00101         this->weather = weather;
00102     }
```

4.14.3.12 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 31 of file [KeyPoint.cpp](#).

```

00031                                     {
00032         for (int i = 0; i < entities.size(); i++) {
00033             if (entities[i]->getAlliance() == alliance) {
00034                 int random;
00035                 do {
00036                     random = rand() % entities.size();
00037                 } while (entities[random]->getAlliance() == alliance);
00038             }
00039             if (rand() % (int)(weather->getMultiplier() * 100) <= (int)(weather->getMultiplier() *
100))
00040                 entities[i]->dealDamage(entities[random]);
00041         }
00042     }
00043 }
```

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

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

4.15 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

4.15.1 Detailed Description

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

4.15.2 Constructor & Destructor Documentation

4.15.2.1 Negotiator()

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

Instantiates the [Negotiator](#).

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

```
00004 {}
```

4.15.2.2 ~Negotiator()

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

Destructor for the [Negotiator](#) object.

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

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

4.15.3 Member Function Documentation

4.15.3.1 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 }
```

4.15.3.2 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 }
```

4.15.3.3 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 }
```

4.15.3.4 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](#).

```

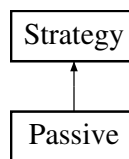
00010                                     {
00011
00012     for (int yy = 0; yy < alliances.size(); yy++)
00013     {
00014         if (alliances[yy] != offerAlliance) {
00015             if (alliances[yy]->considerPeace() == false)
00016                 return false; // There is at least one enemy alliances that does not want the peace
00017         }
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](#)

4.16 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

4.16.1 Detailed Description

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

4.16.2 Constructor & Destructor Documentation

4.16.2.1 Passive()

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

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

```
00005 {}
```

4.16.3 Member Function Documentation

4.16.3.1 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 }
```

4.16.3.2 performStrat()

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

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

Author

Antwi-Antwi

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:

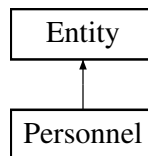
- [Passive.h](#)
- [Passive.cpp](#)

4.17 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](#) ()
Returns the clone of the [Personnel](#) object.

4.17.1 Detailed Description

[Personnel](#) class.

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

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

4.17.2 Constructor & Destructor Documentation

4.17.2.1 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 3 of file [Personnel.cpp](#).

```
00003 : Entity(type, health, damage) {}
```

4.17.3 Member Function Documentation

4.17.3.1 clone()

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

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

Returns

Entity* Clone of personnel object

Implements [Entity](#).

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

```
00013 {  
00014     return new Personnel(this->getType()->clone(), this->getHealth(), this->getDamage());  
00015 }
```

4.17.3.2 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 9 of file [Personnel.cpp](#).

```
00009 {  
00010     entity->takeDamage (getDamage () );  
00011 }
```

4.17.3.3 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

Parameters

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

Returns

void

Implements [Entity](#).

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

```
00005 {  
00006     setHealth (getHealth () - damage);  
00007 }
```

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

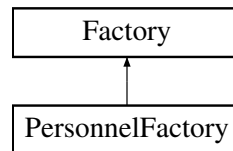
- Personnel.h
- Personnel.cpp

4.18 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.

4.18.1 Detailed Description

[PersonnelFactory](#) class.

Used to instantiate [Personnel](#) objects.

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

4.18.2 Constructor & Destructor Documentation

4.18.2.1 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 4 of file [PersonnelFactory.cpp](#).

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

4.18.3 Member Function Documentation

4.18.3.1 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 17 of file [PersonnelFactory.cpp](#).

```
00017 {
00018     return new PersonnelFactory(getType()->clone(), getAddOn()->clone());
00019 }
```

4.18.3.2 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 6 of file [PersonnelFactory.cpp](#).

```

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

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

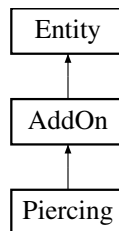
- PersonnelFactory.h
- PersonnelFactory.cpp

4.19 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

4.19.1 Detailed Description

[Piercing](#) class.

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

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

4.19.2 Constructor & Destructor Documentation

4.19.2.1 Piercing()

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

Instantiates an [Piercing](#).

Parameters

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

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

```
00003 :   AddOn(value) {}
```

4.19.3 Member Function Documentation

4.19.3.1 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 14 of file [Piercing.cpp](#).

```
00014         {
00015     Piercing* piercing = new Piercing(value);
00016     piercing->setEntity(entity->clone());
00017     return piercing;
00018 }
```

4.19.3.2 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 9 of file [Piercing.cpp](#).

```
00009 {
00010     int sumValue = this->entity->getDamage() + value;
00011     entity->takeDamage(sumValue);
00012 }
```

4.19.3.3 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

Parameters

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

Returns

void

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

```

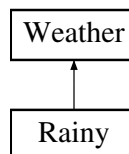
00005 {
00006     entity->takeDamage (damage);
00007 }
```

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

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

4.20 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](#) *k)
Will change the current state of the weather inside the specific keypoint.
- [Weather](#) * [clone](#) ()
Returns a clone of the [Rainy](#) object.

Additional Inherited Members

4.20.1 Detailed Description

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

4.20.2 Constructor & Destructor Documentation

4.20.2.1 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 }
```

4.20.3 Member Function Documentation

4.20.3.1 clone()

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

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

Returns

Weather* The clone of the rainy object

Implements [Weather](#).

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

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

4.20.3.2 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 }
```

4.20.3.3 handleChange()

```
void Rainy::handleChange (
    KeyPoint * k ) [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 ([Sunny](#))

Parameters

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

Returns

void

Implements [Weather](#).

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

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

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

- Rainy.h
- Rainy.cpp

4.21 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:

4.21.1 Detailed Description

Stores a list of mementos containing simulation state.

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

4.21.2 Constructor & Destructor Documentation

4.21.2.1 SaveArchive()

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

Instantiates the [SaveArchive](#) class.

Definition at line 3 of file [SaveArchive.cpp](#).
00003 {}

4.21.3 Member Function Documentation

4.21.3.1 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 }
```

4.21.3.2 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 }
```

4.21.3.3 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 }
```

4.21.3.4 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 }
```


4.21.3.5 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

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

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

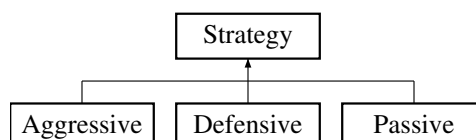
```
00022                                     {
00023     if(saveList.size() == 0){
00024         std::__throw_out_of_range("Save archive is empty");
00025     }
00026
00027     auto iter = saveList.find(name);
00028
00029     if(iter == saveList.end())
00030         std::__throw_invalid_argument("No 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

4.22 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
- virtual [Strategy](#) * [clone](#) ()=0

Protected Attributes

- std::string [strategy](#)

4.22.1 Detailed Description

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

4.22.2 Constructor & Destructor Documentation

4.22.2.1 Strategy()

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

Construct a new [Strategy](#) object.

Definition at line 7 of file [Strategy.cpp](#).
00007 {}

4.22.2.2 ~Strategy()

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

Destroy the [Strategy](#) object.

Definition at line 9 of file [Strategy.cpp](#).
00009 {}

4.22.3 Member Function Documentation

4.22.3.1 clone()

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

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

4.22.3.2 performStrat()

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

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

4.22.4 Member Data Documentation

4.22.4.1 strategy

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

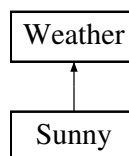
Definition at line 13 of file [Strategy.h](#).

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

- Strategy.h
- Strategy.cpp

4.23 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](#) *k)
Will change the current state of the weather inside the specific keypoint.
- [Weather](#) * [clone](#) ()
Returns the clone of the [Sunny](#) object.

Additional Inherited Members

4.23.1 Detailed Description

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

4.23.2 Constructor & Destructor Documentation

4.23.2.1 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 }
```

4.23.3 Member Function Documentation

4.23.3.1 clone()

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

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

Returns

[Weather](#)* Returns the clone of the sunny object

Implements [Weather](#).

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

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

4.23.3.2 `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 }
```

4.23.3.3 `handleChange()`

```
void Sunny::handleChange (
    KeyPoint * k ) [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>k</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:

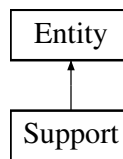
- Sunny.h
- Sunny.cpp

4.24 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.

4.24.1 Detailed Description

[Support](#) class.

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

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

4.24.2 Constructor & Destructor Documentation

4.24.2.1 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 3 of file [Support.cpp](#).

```
00003 : Entity(type, health, damage) {}
```

4.24.3 Member Function Documentation

4.24.3.1 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 13 of file [Support.cpp](#).

```
00013 {  
00014     return new Support(this->getType()->clone(), this->getHealth(), this->getDamage());  
00015 }
```

4.24.3.2 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 5 of file [Support.cpp](#).

```
00005 {  
00006     entity->takeDamage (getDamage());  
00007 }
```

4.24.3.3 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

Parameters

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

Returns

void

Implements [Entity](#).Definition at line 9 of file [Support.cpp](#).

```
00009 {  
00010     this->setHealth(this->getHealth() - damage);  
00011 }
```

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

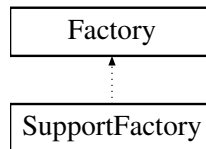
- Support.h
- Support.cpp

4.25 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.

4.25.1 Detailed Description

[SupportFactory](#) class.

Used to instantiate [Support](#) objects.

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

4.25.2 Constructor & Destructor Documentation

4.25.2.1 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) {}
```

4.25.3 Member Function Documentation

4.25.3.1 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 17 of file [SupportFactory.cpp](#).

```
00017 {
00018     return new SupportFactory(getType()->clone(), getAddOn()->clone());
00019 }
```

4.25.3.2 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 instantiated support

Implements [Factory](#).

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

```

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

```

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

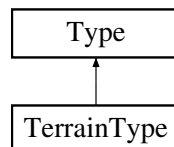
- SupportFactory.h
- SupportFactory.cpp

4.26 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](#) ()

4.26.1 Detailed Description

[TerrainType](#) class.

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

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

4.26.2 Constructor & Destructor Documentation

4.26.2.1 TerrainType()

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

Instantiates the terrain type.

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

```
00003 {}
```

4.26.3 Member Function Documentation

4.26.3.1 clone()

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

Implements [Type](#).

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

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

4.26.3.2 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:

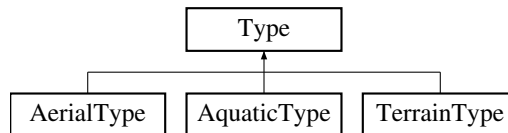
- [TerrainType.h](#)
- [TerrainType.cpp](#)

4.27 Type Class Reference

Type class.

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

Inheritance diagram for Type:



Public Member Functions

- [Type](#) ()
Instantiates the type.
- virtual string [getTypeDesc](#) ()=0
- virtual [Type](#) * [clone](#) ()=0

4.27.1 Detailed Description

Type class.

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

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

4.27.2 Constructor & Destructor Documentation

4.27.2.1 Type()

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

Instantiates the type.

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

```
00003 {}
```

4.27.3 Member Function Documentation

4.27.3.1 getTypeDesc()

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

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

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

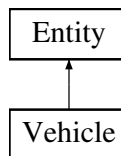
- [Type.h](#)
- [Type.cpp](#)

4.28 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.

4.28.1 Detailed Description

[Vehicle](#) class.

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

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

4.28.2 Constructor & Destructor Documentation

4.28.2.1 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 3 of file [Vehicle.cpp](#).

```
00003 : Entity(type, health, damage) {}
```

4.28.3 Member Function Documentation

4.28.3.1 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 13 of file [Vehicle.cpp](#).

```
00013 {  
00014     return new Vehicle(this->getType()->clone(), this->getHealth(), this->getDamage());  
00015 }
```

4.28.3.2 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 9 of file [Vehicle.cpp](#).

```
00009 {
00010     entity->takeDamage (getDamage());
00011 }
```

4.28.3.3 takeDamage()

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

Removes health from the vehicle object.

Preconditions:

- damage must be an int

Postconditions:

- Reduces the health of the vehicle object

Parameters

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

Returns

void

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

```
00005 {
00006     setHealth (getHealth() - damage);
00007 }
```

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

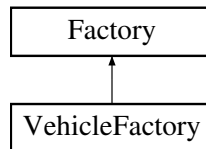
- Vehicle.h
- Vehicle.cpp

4.29 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.

4.29.1 Detailed Description

[VehicleFactory](#) class.

Used to instantiate [Vehicle](#) objects.

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

4.29.2 Constructor & Destructor Documentation

4.29.2.1 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) {}
```

4.29.3 Member Function Documentation

4.29.3.1 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 17 of file [VehicleFactory.cpp](#).

```
00017 {
00018     return new VehicleFactory(getType()->clone(), getAddOn()->clone());
00019 }
```

4.29.3.2 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 instatiated vehicle

Implements [Factory](#).

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

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

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

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

4.30 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 [loadState](#) ([WarEngineState](#) *save)
Takes in an instance of saved [WarEngine](#) states and sets current instance's member variables to memento state.
- void [simulate](#) ()
Simulates battle on WarTheatres.
- void [setWarTheatre](#) ([WarTheatre](#) *battleGround)
Sets the state's area to passed in battleGround parameter.

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](#) ()
Destrctor for class responsible for freeing all allocated memory.

4.30.1 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](#).

4.30.2 Constructor & Destructor Documentation

4.30.2.1 WarEngine() [1/2]

```
WarEngine::WarEngine ( ) [protected]
```

Constructor for class. Is responsible for ensuring only a single instance of class exists.

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

```
00003     {
00004         this->state = new WarEngineState();
00005     }
```

4.30.2.2 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 {};
```

4.30.2.3 ~WarEngine()

```
WarEngine::~WarEngine ( ) [protected]
```

Destructor for class responsible for freeing all allocated memory.

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

```
00021     {
00022         delete this->state;
00023     }
```

4.30.3 Member Function Documentation

4.30.3.1 getInstance()

`WarEngine & WarEngine::getInstance () [static]`

Function that returns a reference to the current (and only) instance of the class.

Returns

`WarEngine&`

Definition at line 16 of file `WarEngine.cpp`.

```
00016     {
00017         static WarEngine uniqueInstance_;
00018         return uniqueInstance_;
00019     }
```

4.30.3.2 loadState()

```
void WarEngine::loadState (
    WarEngineState * save )
```

Takes in an instance of saved `WarEngine` states and sets current instance's member variables to memento state.

Parameters

save	
------	--

Preconditions:

- Save must be of type `WarEngineState*`

Postconditions:

- Sets the instance of the class' state member variable to the passed in save parameter.

Returns

`void`

Definition at line 11 of file `WarEngine.cpp`.

```
00011     {
00012         delete this->state;
00013         this->state = save;
00014     }
```

4.30.3.3 operator=()

```
WarEngine & WarEngine::operator= (
    const WarEngine & ) [inline], [protected]
```

Overloaded operator = for class.

Definition at line 38 of file [WarEngine.h](#).

```
00038 { return *this; };
```

4.30.3.4 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 7 of file [WarEngine.cpp](#).

```
00007 {
00008     return new WarEngineMemento(state->clone());
00009 }
```

4.30.3.5 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 35 of file [WarEngine.cpp](#).

```

00035         {
00036             state->setArea(battleGround);
00037     }

```

4.30.3.6 simulate()

```
void WarEngine::simulate ( )
```

Simulates battle on WarTheatres.

Returns

void

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

```

00025         {
00026
00027             vector<Alliance*> alliances = this->state->getAlliances();
00028
00029             for(int j = 0; j < alliances.size(); j++) {
00030                 state->getArea()->simulateBattle(alliances[j]);
00031             }
00032
00033     }

```

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

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

4.31 WarEngineMemento Class Reference

#include <WarEngineMemento.h>

Friends

- class [WarEngine](#)

4.31.1 Detailed Description

Class that encapsulates and externalises [WarEngine](#) State.Definition at line 15 of file [WarEngineMemento.h](#).

4.31.2 Friends And Related Function Documentation

4.31.2.1 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

4.32 WarEngineState Class Reference

Class for storing current state of entire simulation.

```
#include <WarEngineState.h>
```

Public 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.
- [~WarEngineState](#) ()
Destructor for class.

4.32.1 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 15 of file [WarEngineState.h](#).

4.32.2 Constructor & Destructor Documentation

4.32.2.1 WarEngineState()

```
WarEngineState::WarEngineState ( )
```

Initializes an instance of the [WarEngineState](#) class.

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

```
00003         {  
00004     area = nullptr;  
00005 }
```

4.32.2.2 ~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  
00056 }
```

4.32.3 Member Function Documentation

4.32.3.1 clone()

```
WarEngineState * WarEngineState::clone ( )
```

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         Alliance* clonedAlliance = alliance->clone();  
00039  
00040         clonedState->alliances.push_back(alliance);  
00041  
00042     }  
00043  
00044     return clonedState;  
00045  
00046 }
```

4.32.3.2 getAlliances()

```
vector< Alliance * > WarEngineState::getAlliances ( )
```

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 }
```

4.32.3.3 getArea()

```
Area * WarEngineState::getArea ( )
```

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 }
```

4.32.3.4 setAlliances()

```
void WarEngineState::setAlliances (
    vector< Alliance * > alliances )
```

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 }
```

4.32.3.5 setArea()

```
void WarEngineState::setArea (
    Area * area )
```

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](#).

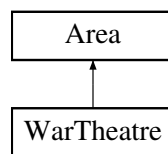
```
00007                                     {  
00008     this->area = area;  
00009 }
```

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

- WarEngineState.h
- WarEngineState.cpp

4.33 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.
- [WarTheatre](#) * [clone](#) ()
Instantiates and returns a clone of the current war theatre.

4.33.1 Detailed Description

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

4.33.2 Constructor & Destructor Documentation

4.33.2.1 WarTheatre()

```
WarTheatre::WarTheatre (
    std::string areaName )
```

Instantiates the war theatre.

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

```
00005 :   Area(areaName) {}
```

4.33.2.2 ~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 }
```

4.33.3 Member Function Documentation

4.33.3.1 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 }
```

4.33.3.2 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 }
```

4.33.3.3 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 }
```

4.33.3.4 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](#).

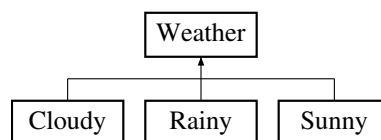
```
00016 {
00017     for (int i = 0; i < areas.size(); i++)
00018         areas[i]->simulateBattle(alliance);
00019 }
```

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

- WarTheatre.h
- WarTheatre.cpp

4.34 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](#) *k)=0
- virtual std::string [getWeather](#) ()=0
- virtual [Weather](#) * [clone](#) ()=0

Protected Attributes

- double [multiplier](#)

4.34.1 Detailed Description

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

4.34.2 Constructor & Destructor Documentation

4.34.2.1 [Weather](#)()

```
Weather::Weather ( )
```

Instantiates the [Weather](#) object.

Definition at line 3 of file [Weather.cpp](#).
00003 { }

4.34.2.2 [~Weather](#)()

```
Weather::~~Weather ( )
```

Destructor for the [Weather](#) object.

Definition at line 5 of file [Weather.cpp](#).
00005 { }

4.34.3 Member Function Documentation

4.34.3.1 clone()

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

Implemented in [Cloudy](#), [Rainy](#), and [Sunny](#).

4.34.3.2 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 }
```

4.34.3.3 getWeather()

```
virtual std::string Weather::getWeather ( ) [pure virtual]
```

Implemented in [Cloudy](#), [Rainy](#), and [Sunny](#).

4.34.3.4 handleChange()

```
virtual void Weather::handleChange (  
    KeyPoint * k ) [pure virtual]
```

Implemented in [Cloudy](#), [Rainy](#), and [Sunny](#).

4.34.4 Member Data Documentation

4.34.4.1 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
00003 AddOn::AddOn(int value): Entity() {
00004     this->value = value;
00005     entity = NULL;
00006 }
00007
00008 void AddOn::setValue(int value) {
00009     this->value = value;
00010 }
00011
00012 int AddOn::getValue() {
00013     return value;
00014 }
00015
00016 void AddOn::setEntity(Entity* entity) {
00017     this->entity = entity;
00018 }
00019
00020 Entity* AddOn::getEntity() {
00021     return this->entity;
00022 }
```

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
00072     virtual void takeDamage(int damage) = 0;
00073
00074     virtual void dealDamage(Entity* entity) = 0;
00075
00076     virtual AddOn* clone() = 0;
00077 };
00078
00079 #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, p->getHealth());
00117     }
00118
00119     // Test Precondition Bounds
00120     TEST(ArmourTakeDamageTest, TestPreconditionBounds) {
00121         Armour* a = new Armour(10);
00122         try {
00123             a->takeDamage(0);
00124             FAIL();
00125         } catch (std::invalid_argument& err) {
00126             EXPECT_EQ(err.what(), std::string("damage must be greater than zero"));
00127         } catch (...) {
00128             FAIL();
00129         }
00130     }
00131
00132     // ===== Positive Testing =====
00133     // Test Preconditions Bounds
00134     TEST(ArmourTakeDamageTest, PositiveTesting) {
00135         Armour* a = new Armour(20);
00136         Personnel* p = new Personnel(new TerrainType(), 100, 10);
00137         a->setEntity(p);
00138
00139         a->takeDamage(10);
00140         EXPECT_EQ(10, a->getValue());
00141         EXPECT_EQ(100, p->getHealth());
00142
00143         a->takeDamage(10);
00144         EXPECT_EQ(0, a->getValue());
00145         EXPECT_EQ(100, p->getHealth());
00146
00147         a->takeDamage(10);
00148         EXPECT_EQ(0, a->getValue());
00149         EXPECT_EQ(90, p->getHealth());
00150     }
00151
00152     // Tests Armour dealDamage()
00153     // ===== Positive Testing =====
00154     // Test Preconditions Bounds
00155     TEST(ArmourDealDamageTest, PositiveTesting) {
00156         Armour* a = new Armour(10);
00157         Personnel* p = new Personnel(new TerrainType(), 100, 10);
00158         a->setEntity(p);
00159         Personnel* x = new Personnel(new TerrainType(), 100, 10);
00160
00161         a->dealDamage(x);
00162         EXPECT_EQ(10, a->getValue());
00163         EXPECT_EQ(90, x->getHealth());
00164
00165         a->dealDamage(x);
00166         EXPECT_EQ(10, a->getValue());
00167         EXPECT_EQ(80, x->getHealth());
00168
00169         a->dealDamage(x);
00170         EXPECT_EQ(10, a->getValue());
00171         EXPECT_EQ(70, x->getHealth());
00172     }

```

```

00173
00174 // Tests Piercing AddOn Functionality
00175
00176 // Tests Piercing takeDamage()
00177 // ===== Positive Testing =====
00178 // Test Preconditions Bounds
00179 TEST(PiercingTakeDamageTest, PositiveTesting) {
00180     Piercing* pi = new Piercing(10);
00181     Personnel* p = new Personnel(new TerrainType(), 100, 10);
00182     pi->setEntity(p);
00183
00184     pi->takeDamage(10);
00185     EXPECT_EQ(10, pi->getValue());
00186     EXPECT_EQ(90, p->getHealth());
00187
00188     pi->takeDamage(10);
00189     EXPECT_EQ(10, pi->getValue());
00190     EXPECT_EQ(80, p->getHealth());
00191
00192     pi->takeDamage(10);
00193     EXPECT_EQ(10, pi->getValue());
00194     EXPECT_EQ(70, p->getHealth());
00195 }
00196
00197 // Tests Piercing dealDamage()
00198 // ===== Positive Testing =====
00199 // Test Preconditions Bounds
00200 TEST(PiercingDealDamageTest, PositiveTesting) {
00201     Piercing* pi = new Piercing(10);
00202     Personnel* p = new Personnel(new TerrainType(), 100, 10);
00203     pi->setEntity(p);
00204     Personnel* x = new Personnel(new TerrainType(), 100, 10);
00205
00206     pi->dealDamage(x);
00207     EXPECT_EQ(10, pi->getValue());
00208     EXPECT_EQ(80, x->getHealth());
00209
00210     pi->dealDamage(x);
00211     EXPECT_EQ(10, pi->getValue());
00212     EXPECT_EQ(60, x->getHealth());
00213
00214     pi->dealDamage(x);
00215     EXPECT_EQ(10, pi->getValue());
00216     EXPECT_EQ(40, x->getHealth());
00217 }
00218 }

```

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     Type* clone();
00030 };
00031
00032 #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
00023     void performStrat(KeyPoint* keyPoint, Alliance* alliance);
00024
00030     Strategy* clone();
00031 };
00032
00033 #endif

```

5.8 Alliance.cpp

```

00001 #include "Alliance.h"
00002 #include "Negotiator.h"
00003 #include "Entity.h"
00004 #include <time.h>
00005
00006 int Alliance::totalNum = 0;
00007
00008 Alliance::Alliance() {
00009     this->active = 1;
00010     this->aID = totalNum++;
00011     this->negotiator = NULL;
00012     srand(time(0));
00013 }
00014
00015 Alliance::Alliance(Alliance& alliance) {
00016     this->active = alliance.active;
00017     this->aID = alliance.aID;
00018
00019     for (int i = 0; i < alliance.members.size(); i++)
00020         this->addCountry(alliance.members[i]->clone());
00021
00022     for (int i = 0; i < alliance.production.size(); i++)
00023         this->addFactory(alliance.production[i]->clone());
00024
00025     for (int i = 0; i < alliance.reserveEntities.size(); i++)
00026         this->addReserveEntity(alliance.reserveEntities[i]->clone());
00027
00028     this->negotiator = NULL;
00029 }
00030
00031 Alliance::~Alliance() {
00032
00033     this->negotiator->removeAlliance(this);
00034
00035     if (this->negotiator->getNumAlliances() == 1)
00036         delete this->negotiator;
00037 }
00038
00039 void Alliance::setNegotiator(Negotiator* negotiator) {
00040     this->negotiator = negotiator;
00041 }
00042
00043 void Alliance::addCountry(Country* nation) {

```

```

00044     members.push_back(nation);
00045 }
00046
00047 vector<Entity*> Alliance::getReserveEntities(int number) {
00048     vector<Entity*> out;
00049     for (int i = 0; i < number && i < reserveEntities.size(); i++) {
00050         out.push_back(reserveEntities[i]);
00051         reserveEntities.erase(reserveEntities.begin() + i);
00052     }
00053
00054     return out;
00055 }
00056
00057 void Alliance::addReserveEntity(Entity* entity) {
00058     reserveEntities.push_back(entity);
00059 }
00060
00061 bool Alliance::considerPeace() {
00062     return (rand() % 2 == 0);
00063 }
00064
00065 void Alliance::addFactory(Factory* factory) {
00066     production.push_back(factory);
00067 }
00068
00069 void Alliance::surrender() {
00070     this->active = 2; //Number 2 means that Alliance has surrendered
00071
00072     this->negotiator->removeAlliance(this);
00073 }
00074
00075 int Alliance::getID() {
00076     return this->aID;
00077 }
00078
00079 bool Alliance::offerPeace() {
00080
00081     if (this->negotiator->sendPeace(this)) //Send the peace deal to all the alliances fighting against
00082     {
00083         this->active = 3; //Number 3 means that Alliance chose to peacefully pull out of war
00084         return true;
00085     }
00086
00087     return false;
00088 }
00089
00090 int Alliance::getActive() {
00091     return active;
00092 }
00093
00094 Alliance* Alliance::clone() {
00095     return new Alliance(*this);
00096 }

```

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);

```



```

00036
00040     ~Alliance();
00041
00054     void setNegotiator(Negotiator* newNegotiator);
00055
00068     void addCountry(Country* nation);
00069
00083     vector<Entity*> getReserveEntities(int number);
00084
00097     void addReserveEntity(Entity* entity);
00098
00110     bool considerPeace();
00111
00124     void addFactory(Factory* factory);
00125
00135     void surrender();
00136
00145     int getID();
00146
00155     bool offerPeace();
00156
00165     Alliance* clone();
00166
00178     void setActiveStatus(bool active);
00179
00180     int getActive();
00181 };
00182
00183 #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
00013 public:
00017     AquaticType();
00018
00027     string getTypeDesc();
00028
00029     Type* clone();
00030 };
00031
00032 #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 Area {
00007
00008 private:
00009     std::string areaName;
00010
00011 public:
00012     Area(std::string areaName);
00013
00014     virtual ~Area();
00015
00016     virtual bool isKeyPoint() = 0;
00017
00018     virtual void simulateBattle(Alliance* alliance) = 0;
00019
00020     std::string getAreaName() const;
00021
00022     virtual Area* clone() = 0;
00023 };
00024
00025 #endif

```

5.14 Armour.cpp

```

00001 #include "Armour.h"
00002
00003 Armour::Armour(int value) : AddOn(value) {}
00004
00005 void Armour::takeDamage(int damage) {
00006     if (value > 0) {
00007         value -= damage;
00008     } else {
00009         entity->takeDamage(damage);
00010     }
00011 }
00012
00013 void Armour::dealDamage(Entity* entity) {
00014     this->entity->dealDamage(entity);
00015 }
00016
00017 AddOn* Armour::clone() {
00018     Armour* armour = new Armour(value);
00019     armour->setEntity(entity->clone());
00020     return armour;
00021 }

```

5.15 Armour.h

```

00001 #ifndef ARMOUR_H
00002 #define ARMOUR_H
00003 #include "AddOn.h"
00004 #include "Entity.h"
00005
00006 class Armour : public AddOn {
00007
00008 public:
00009     Armour(int value);
00010
00011     void takeDamage(int damage);
00012
00013     void dealDamage(Entity* entity);
00014 }

```

```

00058     AddOn* clone();
00059 };
00060
00061 #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* k);
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() {}
00006
00007
00008 Country* Country::clone() {
00009
00010     Country* countryClone = new Country();
00011     countryClone->setID(this->id);
00012     countryClone->setName(this->name);
00013
00014     return countryClone;
00015 }
00016
00017 void Country::setID(int id){
00018     this->id = id;
00019 }
00020
00021 void Country::setName(string name){
00022     this->name = name;
00023 }
00024
00025 string Country::getName() const{

```

```
00026     return this->name;
00027 }
00028
00029 int Country::getID() const{
00030     return this->id;
00031 }
00032
00033
```

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();
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 #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 public:
00010     Defensive();
00011
00012     void performStrat(KeyPoint* keyPoint, Alliance* alliance);
00013
00014     Strategy* clone();
00015 };
00016 #endif
```

5.22 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 }

```

5.23 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:
00025     Entity();
00026
00032     Entity(Type* type, int health, int damage);
00033
00042     Type* getType();
00043
00056     void setType(Type* type);
00057
00066     Alliance* getAlliance();
00067
00080     void setAlliance(Alliance* alliance);
00081
00090     int getHealth();
00091
00104     void setHealth(int health);

```

```

00105
00114     int getDamage();
00115
00128     void setDamage(int damage);
00129
00130     virtual void takeDamage(int damage) = 0;
00131
00132     virtual void dealDamage(Entity* entity) = 0;
00133
00134     virtual Entity* clone() = 0;
00135 };
00136
00137 #endif

```

5.24 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 }

```

5.25 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     void setType(Type* type);
00060
00061     AddOn* getAddOn();
00071
00072     void setAddOns(AddOn* addOn);
00085
00086     virtual Factory* clone() = 0;
00087 };
00088
00089 #endif

```

5.26 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 }

```

5.27 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

```

5.28 KeyPoint.cpp

```

00001 #include "KeyPoint.h"
00002 #include "Weather.h"
00003 #include <time.h>
00004 #include <cstdlib>
00005
00006 using namespace std;
00007
00008 KeyPoint::KeyPoint(string areaName): Area(areaName) {}
00009
00010 KeyPoint::KeyPoint(KeyPoint& keyPoint): Area(keyPoint.getAreaName()) {
00011     for (int i = 0; i < keyPoint.entities.size(); i++)
00012         this->addEntity(keyPoint.entities[i]->clone());
00013
00014     weather = keyPoint.weather->clone();
00015 }
00016
00017 KeyPoint::~KeyPoint() {
00018     for (int i = 0; i < entities.size(); i++)
00019         delete entities[i];
00020
00021     for (int i = 0; i < generals.size(); i++)
00022         delete generals[i];

```

```

00023
00024     delete weather;
00025 }
00026
00027 bool KeyPoint::isKeyPoint() {
00028     return true;
00029 }
00030
00031 void KeyPoint::simulateBattle(Alliance* alliance) {
00032     for (int i = 0; i < entities.size(); i++) {
00033         if (entities[i]->getAlliance() == alliance) {
00034             int random;
00035             do {
00036                 random = rand() % entities.size();
00037             } while (entities[random]->getAlliance() == alliance);
00038
00039             if (rand() % (int)(weather->getMultiplier() * 100) <= (int)(weather->getMultiplier() *
100))
00040                 entities[i]->dealDamage(entities[random]);
00041         }
00042     }
00043 }
00044
00045 void KeyPoint::clearBattlefield() {
00046     for (vector<Entity*>::iterator it = entities.begin(); it != entities.end(); ++it) {
00047         if ((*it)->getHealth() <= 0) {
00048             for (int i = 0; i < generals.size(); i++) {
00049                 if (generals[i]->getAlliance() == (*it)->getAlliance()) {
00050                     generals[i]->initiateStrategy(this);
00051                     delete *it;
00052                     entities.erase(it);
00053                 }
00054             }
00055         }
00056     }
00057 }
00058
00059 void KeyPoint::moveEntitiesInto(Alliance* alliance, int numTroops) {
00060     vector<Entity*> troops = alliance->getReserveEntities(numTroops);
00061     for (int i = 0; i < troops.size(); i++)
00062         entities.push_back(troops[i]);
00063 }
00064
00065 void KeyPoint::moveEntitiesOutOf(Alliance* alliance, int numTroops) {
00066     vector<Entity*>::iterator it = entities.begin();
00067     for (int i = 0; i < numTroops && it != entities.end(); i++) {
00068         for (; it != entities.end(); ++it) {
00069             if ((*it)->getAlliance() == alliance) {
00070                 alliance->addReserveEntity(*it);
00071                 entities.erase(it);
00072             }
00073         }
00074     }
00075 }
00076
00077 void KeyPoint::addEntity(Entity* entity) {
00078     entities.push_back(entity);
00079 }
00080
00081 void KeyPoint::addGeneral(General* general) {
00082     generals.push_back(general);
00083 }
00084
00085 void KeyPoint::removeGeneral(General* general) {
00086     for (vector<General*>::iterator it = generals.begin(); it != generals.end(); ++it) {
00087         if (*it == general) {
00088             delete *it;
00089             generals.erase(it);
00090             return;
00091         }
00092     }
00093 }
00094
00095 Area* KeyPoint::clone() {
00096     return new KeyPoint(*this);
00097 }
00098
00099 void KeyPoint::setWeather(Weather* weather) {
00100     delete this->weather;
00101     this->weather = weather;
00102 }
00103
00104 void KeyPoint::changeWeather() {
00105     srand(time(0));
00106     int randomNum = 1 + (rand() % 10);

```



```

00109     std::string currWeather = this->weather->getWeather();
00110
00111     if (currWeather == "Sunny" && randomNum > 6) // 60% chance of not changing weather from Sunny and
        staying
00112         this->weather->handleChange(this);
00113     else if (currWeather == "Cloudy" && randomNum > 3) // 30% chance of not changing weather from
        Cloudy and staying
00114         this->weather->handleChange(this);
00115     else if (currWeather == "Rainy" && randomNum > 1) // 10% chance of not changing weather from Rainy
        and staying
00116         this->weather->handleChange(this);
00117
00118
00119 }
00120
00121 std::string KeyPoint::getWeather() const {
00122     return this->weather->getWeather();
00123 }

```

5.29 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     std::string AreaType;
00024
00025 public:
00031     KeyPoint(std::string areaName);
00032
00038     KeyPoint(KeyPoint& keyPoint);
00039
00040     ~KeyPoint();
00041
00050     bool isKeyPoint();
00051
00064     void simulateBattle(Alliance* alliance);
00065
00074     void clearBattlefield();
00075
00090     void moveEntitiesInto(Alliance* alliance, int numTroops);
00091
00106     void moveEntitiesOutOf(Alliance* alliance, int numTroops);
00107
00120     void addEntity(Entity* entity);
00121
00122     void addGeneral(General* general);
00123
00124     void removeGeneral(General* general);
00125
00134     Area* clone();
00135
00140     void changeWeather();
00141
00154     void setWeather(Weather* weather);
00155
00161     std::string getWeather() const;
00162
00163 };
00164
00165 #endif

```

5.30 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         }
00018     }
00019
00020     return true; // All the alliances being fought against agreed to the peace deal
00021 }
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     }
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 }

```

5.31 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
00014     ~Negotiator();
00015
00016     bool sendPeace(Alliance* offerAlliance);
00017
00018     void removeAlliance(Alliance* oldAlliance);
00019
00020     void addAlliance(Alliance* newAlliance);
00021
00022     int getNumAlliances();
00023 };
00024 #endif

```

5.32 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 Negotiator offerPeace()
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     // Tests Negotiator surrender()
00030     // ===== Positive Testing =====
00031     // Test Preconditions Bounds
00032     TEST(NegotiatorSurrender, PositiveTesting) {
00033         Alliance* a = new Alliance();
00034         Alliance* b = new Alliance();
00035         Alliance* c = new Alliance();
00036         Alliance* d = new Alliance();
00037         Alliance* e = new Alliance();
00038         Negotiator* n = new Negotiator();
00039         n->addAlliance(a);
00040         n->addAlliance(b);
00041         n->addAlliance(c);
00042         n->addAlliance(d);
00043         n->addAlliance(e);
00044         a->setNegotiator(n);
00045         b->setNegotiator(n);
00046         c->setNegotiator(n);
00047         d->setNegotiator(n);
00048         e->setNegotiator(n);
00049
00050         a->surrender();
00051         EXPECT_EQ(2, a->getActive());
00052
00053         b->surrender();
00054         EXPECT_EQ(2, a->getActive());
00055
00056         c->surrender();
00057         EXPECT_EQ(2, a->getActive());
00058
00059         d->surrender();
00060         EXPECT_EQ(2, a->getActive());
00061     }
00062 }
00063 }

```

5.33 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 }

```

5.34 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
00020     void performStrat(KeyPoint* keyPoint, Alliance* alliance);
00021
00027     Strategy* clone();
00028 };
00029
00030 #endif

```

5.35 Personnel.cpp

```

00001 #include "Personnel.h"
00002
00003 Personnel::Personnel(Type* type, int health, int damage): Entity(type, health, damage) {}
00004
00005 void Personnel::takeDamage(int damage) {
00006     setHealth(getHealth() - damage);
00007 }
00008
00009 void Personnel::dealDamage(Entity* entity) {
00010     entity->takeDamage(getDamage());
00011 }
00012
00013 Entity* Personnel::clone() {
00014     return new Personnel(this->getType()->clone(), this->getHealth(), this->getDamage());
00015 }

```

5.36 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
00035     void takeDamage(int damage);
00036
00049     void dealDamage(Entity* entity);
00050
00056     Entity* clone();
00057 };
00058
00059 #endif

```

5.37 PersonnelFactory.cpp

```

00001 #include "PersonnelFactory.h"
00002 #include "Personnel.h"
00003
00004 PersonnelFactory::PersonnelFactory(Type* type, AddOn* addOn): Factory(type, addOn) {}
00005
00006 Entity* PersonnelFactory::createEntity(Alliance* alliance) {
00007     Personnel* p = new Personnel(getType()->clone());
00008     if (getAddOn() != NULL) {
00009         AddOn* personnelAddOn = getAddOn()->clone();
00010         personnelAddOn->setEntity(p);
00011         return personnelAddOn;
00012     } else {
00013         return p;
00014     }
00015 }
00016
00017 Factory* PersonnelFactory::clone() {
00018     return new PersonnelFactory(getType()->clone(), getAddOn()->clone());
00019 }

```

5.38 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

```

5.39 Piercing.cpp

```

00001 #include "Piercing.h"
00002
00003 Piercing::Piercing(int value) : AddOn(value) {}
00004
00005 void Piercing::takeDamage(int damage) {
00006     entity->takeDamage(damage);
00007 }
00008
00009 void Piercing::dealDamage(Entity* entity) {
00010     int sumValue = this->entity->getDamage() + value;
00011     entity->takeDamage(sumValue);
00012 }
00013
00014 AddOn* Piercing::clone() {
00015     Piercing* piercing = new Piercing(value);
00016     piercing->setEntity(entity->clone());
00017     return piercing;
00018 }

```

5.40 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
00034     void takeDamage(int damage);
00035
00048     void dealDamage(Entity* entity);
00049
00058     AddOn* clone();
00059 };
00060
00061 #endif

```

5.41 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* k) {
00013     Sunny* newWeather = new Sunny();
00014     k->setWeather(newWeather);
00015 }
00016
00017 Weather* Rainy::clone() {
00018     return new Rainy();
00019 }

```

5.42 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:
00013     Rainy();
00014
00023     std::string getWeather();
00024
00037     void handleChange(KeyPoint* k);
00038
00044     Weather* clone();
00045 };
00046
00047 #endif

```

5.43 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 }

```

5.44 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

```

5.45 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

```

5.46 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
00028     virtual void performStrat(KeyPoint* keyPoint, Alliance* alliance) = 0;
00029
00030     virtual Strategy* clone() = 0;
00031 };
00032
00033 #endif

```

5.47 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 }

```

5.48 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
00010 public:
00011     Sunny();
00012
00013     virtual std::string getWeather();
00014
00015     virtual void handleChange(KeyPoint* k);
00016
00017     Weather* clone();
00018 };
00019 #endif

```

5.49 Support.cpp

```

00001 #include "Support.h"
00002
00003 Support::Support(Type* type, int health, int damage): Entity(type, health, damage) {}
00004
00005 void Support::dealDamage(Entity* entity) {
00006     entity->takeDamage(getDamage());
00007 }
00008
00009 void Support::takeDamage(int damage) {
00010     this->setHealth(this->getHealth() - damage);
00011 }
00012
00013 Entity* Support::clone() {
00014     return new Support(this->getType()->clone(), this->getHealth(), this->getDamage());
00015 }

```

5.50 Support.h

```

00001 #ifndef SUPPORT_H
00002 #define SUPPORT_H
00003
00004 #include "Entity.h"
00005
00006 class Support : public Entity {
00007
00008 public:
00009     Support(Type* type, int health = 1000, int damage = 30);
00010 };

```



```

00035     void takeDamage(int damage);
00036
00049     void dealDamage(Entity* entity);
00050
00056     Entity* clone();
00057 };
00058
00059 #endif

```

5.51 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     if (getAddOn() != NULL) {
00009         AddOn* personnelAddOn = getAddOn()->clone();
00010         personnelAddOn->setEntity(s);
00011         return personnelAddOn;
00012     } else {
00013         return s;
00014     }
00015 }
00016
00017 Factory* SupportFactory::clone() {
00018     return new SupportFactory(getType()->clone(), getAddOn()->clone());
00019 }

```

5.52 SupportFactory.h

```

00001 #ifndef SUPPORTFACTORY_H
00002 #define SUPPORTFACTORY_H
00003
00004 #include "Factory.h"
00005
00011 class SupportFactory : Factory {
00012
00013 public:
00020     SupportFactory(Type* type, AddOn* addOn);
00021
00034     Entity* createEntity(Alliance* alliance);
00035
00044     Factory* clone();
00045 };
00046
00047 #endif

```

5.53 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 }

```

5.54 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
00029     Type* clone();
00030 };
00031
00032 #endif
```

5.55 testmain.cpp

```
00001 #include "NegotiatorTest.h"
00002 #include "AddOnTest.h"
00003 #include <gtest/gtest.h>
00004
00005 int main(int argc, char **argv) {
00006     testing::InitGoogleTest(&argc, argv);
00007     return RUN_ALL_TESTS();
00008 }
```

5.56 Type.cpp

```
00001 #include "Type.h"
00002
00003 Type::Type() {}
```

5.57 Type.h

```
00001 #ifndef TYPE_H
00002 #define TYPE_H
00003
00004 #include <string>
00005
00006 using namespace std;
00007
00013 class Type {
00014
00015 public:
00019     Type();
00020
00021     virtual string getTypeDesc() = 0;
00022
00023     virtual Type* clone() = 0;
00024 };
00025
00026
00027 #endif
```

5.58 Vehicle.cpp

```
00001 #include "Vehicle.h"
00002
00003 Vehicle::Vehicle(Type* type, int health, int damage): Entity(type, health, damage) {}
00004
00005 void Vehicle::takeDamage(int damage) {
00006     setHealth(getHealth() - damage);
00007 }
00008
00009 void Vehicle::dealDamage(Entity* entity) {
00010     entity->takeDamage(getDamage());
00011 }
00012
00013 Entity* Vehicle::clone() {
00014     return new Vehicle(this->getType()->clone(), this->getHealth(), this->getDamage());
00015 }
```

5.59 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
00035     void takeDamage(int damage);
00036
00049     void dealDamage(Entity* entity);
00050
00056     Entity* clone();
00057 };
00058
00059 #endif

```

5.60 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     if (getAddOn() != NULL) {
00009         AddOn* personnelAddOn = getAddOn()->clone();
00010         personnelAddOn->setEntity(v);
00011         return personnelAddOn;
00012     } else {
00013         return v;
00014     }
00015 }
00016
00017 Factory* VehicleFactory::clone() {
00018     return new VehicleFactory(getType()->clone(), getAddOn()->clone());
00019 }

```

5.61 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

```

5.62 WarEngine.cpp

```

00001 #include "WarEngine.h"
00002
00003 WarEngine::WarEngine() {
00004     this->state = new WarEngineState();
00005 }
00006
00007 WarEngineMemento* WarEngine::saveState() {
00008     return new WarEngineMemento(state->clone());
00009 }
00010
00011 void WarEngine::loadState(WarEngineState* save) {
00012     delete this->state;

```

```

00013     this->state = save;
00014 }
00015
00016 WarEngine& WarEngine::getInstance() {
00017     static WarEngine uniqueInstance_;
00018     return uniqueInstance_;
00019 }
00020
00021 WarEngine::~WarEngine() {
00022     delete this->state;
00023 }
00024
00025 void WarEngine::simulate() {
00026
00027     vector<Alliance*> alliances = this->state->getAlliances();
00028
00029     for(int j = 0; j < alliances.size(); j++) {
00030         state->getArea()->simulateBattle(alliances[j]);
00031     }
00032 }
00033 }
00034
00035 void WarEngine::setWarTheatre(WarTheatre* battleGround) {
00036     state->setArea(battleGround);
00037 }

```

5.63 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 loadState(WarEngineState* save);
00066
00072     static WarEngine& getInstance();
00073
00079     void simulate();
00080
00094     void setWarTheatre(WarTheatre* battleGround);
00095 };
00096
00097 #endif

```

5.64 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 }

```

5.65 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
00010 class WarEngineMemento {
00011 friend class WarEngine;
00012 private:
00013     WarEngineState* state;
00014
00015     WarEngineMemento(WarEngineState* state);
00016
00017     void setState(WarEngineState* state);
00018
00019     WarEngineState* getState();
00020 };
00021 #endif

```

5.66 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     if(area == nullptr)
00013         throw "No Areas Stored.";
00014     return this->area;
00015 }
00016
00017 void WarEngineState::setAlliances(vector<Alliance*> alliances) {
00018     this->alliances = alliances;
00019 }
00020
00021 vector<Alliance*> WarEngineState::getAlliances() {
00022     if(alliances.size() == 0)
00023         std::throw_out_of_range("No Alliances stored.");
00024     return alliances;
00025 }
00026
00027 WarEngineState* WarEngineState::clone() {
00028     WarEngineState* clonedState = new WarEngineState();
00029     clonedState->setArea( this->area->clone() );
00030
00031     for(Alliance* alliance : this->alliances){
00032         Alliance* clonedAlliance = alliance->clone();
00033         clonedState->alliances.push_back(alliance);
00034     }
00035     return clonedState;
00036 }
00037
00038 WarEngineState::~WarEngineState(){
00039     for(Alliance* alliance : this->alliances){
00040         delete alliance;
00041     }
00042 }

```

```

00054     delete this->area;
00055
00056 }

```

5.67 WarEngineState.h

```

00001 #ifndef WARENGINESTATE_H
00002 #define WARENGINESTATE_H
00003 #include "Alliance.h"
00004 #include "Area.h"
00005 #include <vector>
00006
00007 using namespace std;
00008
00015 class WarEngineState {
00016
00017 private:
00018     Area* area;
00020     vector<Alliance*> alliances;
00021
00022 public:
00027     WarEngineState();
00028
00041     void setArea(Area* area);
00042
00051     Area* getArea();
00052
00065     void setAlliances(vector<Alliance*> alliances);
00066
00075     vector<Alliance*> getAlliances();
00076
00082     WarEngineState* clone();
00083
00087     ~WarEngineState();
00088 };
00089
00090 #endif

```

5.68 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

```

5.69 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     WarTheatre* clone();
00027 };
00028 #endif

```

5.70 Weather.cpp

```

00001 #include "Weather.h"
00002
00003 Weather::Weather() {}
00004
00005 Weather::~~Weather() {}
00006
00007 double Weather::getMultiplier() {
00008     return this->multiplier;
00009 }

```

5.71 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:
00015     Weather();
00016
00017     ~Weather();
00018
00019     double getMultiplier();
00020
00021     virtual void handleChange(KeyPoint* k) = 0;
00022
00023     virtual std::string getWeather() = 0;
00024
00025     virtual Weather* clone() = 0;
00026 };
00027 #endif

```


Index

- ~Alliance
 - Alliance, [16](#)
- ~Area
 - Area, [25](#)
- ~Factory
 - Factory, [44](#)
- ~KeyPoint
 - KeyPoint, [51](#)
- ~Negotiator
 - Negotiator, [58](#)
- ~Strategy
 - Strategy, [78](#)
- ~WarEngine
 - WarEngine, [96](#)
- ~WarEngineState
 - WarEngineState, [101](#)
- ~WarTheatre
 - WarTheatre, [105](#)
- ~Weather
 - Weather, [108](#)
- addAlliance
 - Negotiator, [59](#)
- addArea
 - WarTheatre, [105](#)
- addCountry
 - Alliance, [17](#)
- addEntity
 - KeyPoint, [52](#)
- addFactory
 - Alliance, [17](#)
- addGeneral
 - KeyPoint, [52](#)
- addNewSave
 - SaveArchive, [74](#)
- AddOn, [7](#)
 - AddOn, [8](#)
 - clone, [8](#)
 - dealDamage, [8](#)
 - entity, [11](#)
 - getEntity, [8](#)
 - getValue, [9](#)
 - setEntity, [9](#)
 - setValue, [10](#)
 - takeDamage, [10](#)
 - value, [11](#)
- addReserveEntity
 - Alliance, [18](#)
- AerialType, [11](#)
 - AerialType, [12](#)

- clone, [12](#)
- getTypeDesc, [12](#)
- Aggressive, [13](#)
 - Aggressive, [13](#)
 - clone, [13](#)
 - performStrat, [14](#)
- Alliance, [15](#)
 - ~Alliance, [16](#)
 - addCountry, [17](#)
 - addFactory, [17](#)
 - addReserveEntity, [18](#)
 - Alliance, [15](#), [16](#)
 - clone, [18](#)
 - considerPeace, [19](#)
 - getActive, [19](#)
 - getID, [19](#)
 - getReserveEntities, [20](#)
 - offerPeace, [21](#)
 - setActiveStatus, [21](#)
 - setNegotiator, [22](#)
 - surrender, [22](#)
- AquaticType, [23](#)
 - AquaticType, [23](#)
 - clone, [23](#)
 - getTypeDesc, [24](#)
- Area, [24](#)
 - ~Area, [25](#)
 - Area, [25](#)
 - clone, [25](#)
 - getAreaName, [26](#)
 - isKeyPoint, [26](#)
 - simulateBattle, [26](#)
- Armour, [27](#)
 - Armour, [27](#)
 - clone, [28](#)
 - dealDamage, [28](#)
 - takeDamage, [29](#)
- changeWeather
 - KeyPoint, [52](#)
- clearBattlefield
 - KeyPoint, [53](#)
- clearSaveList
 - SaveArchive, [75](#)
- clone
 - AddOn, [8](#)
 - AerialType, [12](#)
 - Aggressive, [13](#)
 - Alliance, [18](#)
 - AquaticType, [23](#)

- Area, 25
- Armour, 28
- Cloudy, 30
- Country, 33
- Defensive, 36
- Entity, 39
- Factory, 44
- KeyPoint, 53
- Passive, 62
- Personnel, 64
- PersonnelFactory, 67
- Piercing, 69
- Rainy, 72
- Strategy, 78
- Sunny, 80
- Support, 83
- SupportFactory, 86
- TerrainType, 88
- Vehicle, 91
- VehicleFactory, 94
- WarEngineState, 101
- WarTheatre, 106
- Weather, 108
- Cloudy, 30
 - clone, 30
 - Cloudy, 30
 - getWeather, 31
 - handleChange, 31
- considerPeace
 - Alliance, 19
- Country, 32
 - clone, 33
 - Country, 32
 - getID, 33
 - getName, 33
 - setID, 34
 - setName, 34
- createEntity
 - Factory, 45
 - PersonnelFactory, 67
 - SupportFactory, 86
 - VehicleFactory, 94
- dealDamage
 - AddOn, 8
 - Armour, 28
 - Entity, 39
 - Personnel, 64
 - Piercing, 69
 - Support, 83
 - Vehicle, 91
- Defensive, 35
 - clone, 36
 - Defensive, 36
 - performStrat, 36
- deleteSave
 - SaveArchive, 75
- Entity, 37
 - clone, 39
 - dealDamage, 39
 - Entity, 38
 - getAlliance, 39
 - getDamage, 39
 - getHealth, 40
 - getType, 40
 - setAlliance, 40
 - setDamage, 41
 - setHealth, 42
 - setType, 42
 - takeDamage, 43
- entity
 - AddOn, 11
- Factory, 43
 - ~Factory, 44
 - clone, 44
 - createEntity, 45
 - Factory, 44
 - getAddOn, 45
 - getType, 45
 - setAddOns, 45
 - setType, 46
- General, 47
 - General, 47
 - getAlliance, 48
 - initiateStrategy, 48
 - setStrategy, 49
- getActive
 - Alliance, 19
- getAddOn
 - Factory, 45
- getAlliance
 - Entity, 39
 - General, 48
- getAlliances
 - WarEngineState, 101
- getArea
 - WarEngineState, 102
- getAreaName
 - Area, 26
- getDamage
 - Entity, 39
- getEntity
 - AddOn, 8
- getHealth
 - Entity, 40
- getID
 - Alliance, 19
 - Country, 33
- getInstance
 - WarEngine, 97
- getLastSave
 - SaveArchive, 76
- getMultiplier
 - Weather, 109
- getName

- Country, 33
- getNumAlliances
 - Negotiator, 59
- getReserveEntities
 - Alliance, 20
- getSave
 - SaveArchive, 76
- getType
 - Entity, 40
 - Factory, 45
- getTypeDesc
 - AerialType, 12
 - AquaticType, 24
 - TerrainType, 88
 - Type, 89
- getValue
 - AddOn, 9
- getWeather
 - Cloudy, 31
 - KeyPoint, 54
 - Rainy, 72
 - Sunny, 80
 - Weather, 109
- handleChange
 - Cloudy, 31
 - Rainy, 72
 - Sunny, 81
 - Weather, 109
- initiateStrategy
 - General, 48
- isKeyPoint
 - Area, 26
 - KeyPoint, 54
 - WarTheatre, 106
- KeyPoint, 50
 - ~KeyPoint, 51
 - addEntity, 52
 - addGeneral, 52
 - changeWeather, 52
 - clearBattlefield, 53
 - clone, 53
 - getWeather, 54
 - isKeyPoint, 54
 - KeyPoint, 51
 - moveEntitiesInto, 54
 - moveEntitiesOutOf, 55
 - removeGeneral, 56
 - setWeather, 56
 - simulateBattle, 57
- loadState
 - WarEngine, 97
- moveEntitiesInto
 - KeyPoint, 54
- moveEntitiesOutOf
 - KeyPoint, 55
- multiplier
 - Weather, 109
- Negotiator, 58
 - ~Negotiator, 58
 - addAlliance, 59
 - getNumAlliances, 59
 - Negotiator, 58
 - removeAlliance, 59
 - sendPeace, 60
- offerPeace
 - Alliance, 21
- operator=
 - WarEngine, 97
- Passive, 61
 - clone, 62
 - Passive, 61
 - performStrat, 62
- performStrat
 - Aggressive, 14
 - Defensive, 36
 - Passive, 62
 - Strategy, 79
- Personnel, 63
 - clone, 64
 - dealDamage, 64
 - Personnel, 63
 - takeDamage, 65
- PersonnelFactory, 66
 - clone, 67
 - createEntity, 67
 - PersonnelFactory, 66
- Piercing, 68
 - clone, 69
 - dealDamage, 69
 - Piercing, 69
 - takeDamage, 70
- Rainy, 71
 - clone, 72
 - getWeather, 72
 - handleChange, 72
 - Rainy, 72
- removeAlliance
 - Negotiator, 59
- removeGeneral
 - KeyPoint, 56
- SaveArchive, 73
 - addNewSave, 74
 - clearSaveList, 75
 - deleteSave, 75
 - getLastSave, 76
 - getSave, 76
 - SaveArchive, 74
- saveState

- WarEngine, 98
- sendPeace
 - Negotiator, 60
- setActiveStatus
 - Alliance, 21
- setAddOns
 - Factory, 45
- setAlliance
 - Entity, 40
- setAlliances
 - WarEngineState, 102
- setArea
 - WarEngineState, 103
- setDamage
 - Entity, 41
- setEntity
 - AddOn, 9
- setHealth
 - Entity, 42
- setID
 - Country, 34
- setName
 - Country, 34
- setNegotiator
 - Alliance, 22
- setStrategy
 - General, 49
- setType
 - Entity, 42
 - Factory, 46
- setValue
 - AddOn, 10
- setWarTheatre
 - WarEngine, 98
- setWeather
 - KeyPoint, 56
- simulate
 - WarEngine, 99
- simulateBattle
 - Area, 26
 - KeyPoint, 57
 - WarTheatre, 106
- Strategy, 77
 - ~Strategy, 78
 - clone, 78
 - performStrat, 79
 - Strategy, 78
 - strategy, 79
- strategy
 - Strategy, 79
- Sunny, 79
 - clone, 80
 - getWeather, 80
 - handleChange, 81
 - Sunny, 80
- Support, 82
 - clone, 83
 - dealDamage, 83
 - Support, 82
 - takeDamage, 84
- SupportFactory, 85
 - clone, 86
 - createEntity, 86
 - SupportFactory, 85
- surrender
 - Alliance, 22
- takeDamage
 - AddOn, 10
 - Armour, 29
 - Entity, 43
 - Personnel, 65
 - Piercing, 70
 - Support, 84
 - Vehicle, 92
- TerrainType, 87
 - clone, 88
 - getTypeDesc, 88
 - TerrainType, 88
- Type, 89
 - getTypeDesc, 89
 - Type, 89
- value
 - AddOn, 11
- Vehicle, 90
 - clone, 91
 - dealDamage, 91
 - takeDamage, 92
 - Vehicle, 90
- VehicleFactory, 93
 - clone, 94
 - createEntity, 94
 - VehicleFactory, 93
- WarEngine, 95
 - ~WarEngine, 96
 - getInstance, 97
 - loadState, 97
 - operator=, 97
 - saveState, 98
 - setWarTheatre, 98
 - simulate, 99
 - WarEngine, 96
 - WarEngineMemento, 100
- WarEngineMemento, 99
 - WarEngine, 100
- WarEngineState, 100
 - ~WarEngineState, 101
 - clone, 101
 - getAlliances, 101
 - getArea, 102
 - setAlliances, 102
 - setArea, 103
 - WarEngineState, 101
- WarTheatre, 104
 - ~WarTheatre, 105

- addArea, [105](#)
- clone, [106](#)
- isKeyPoint, [106](#)
- simulateBattle, [106](#)
- WarTheatre, [104](#)

Weather, [107](#)

- ~Weather, [108](#)
- clone, [108](#)
- getMultiplier, [109](#)
- getWeather, [109](#)
- handleChange, [109](#)
- multiplier, [109](#)
- Weather, [108](#)