COS214 Spice Girls

0.1

Generated by Doxygen 1.9.6

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

Chapter 4

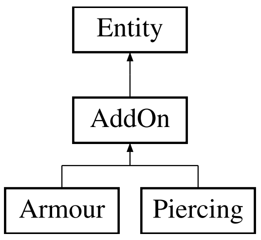
Class Documentation

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 addtional functionality to Entity objects.

Definition at line 10 of file AddOn.h.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 AddOn()

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

Instantiates an AddOn.

Parameters

value	must be an int

Definition at line 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.

```
{
    return this->entity;
}
```

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.

```
{
    return value;
}
```

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

entity	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

value	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:

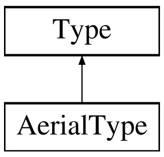
- AddOn.h
- AddOn.cpp

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.

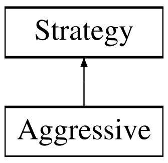
```
00005                                  {
00006     return "Aerial";
00007 }
```

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.

```
                              {
    return new Aggressive();
}
```

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

keyPoint	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.

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

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.
- int numRemainingEntities ()
- 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 runFactories ()
- 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 12 of file Alliance.cpp.

```
                    {
    this->active = 1;
    this->aID = totalNum++;
    this->negotiator = NULL;
    srand(time(0));
}
```

4.4.2.2 Alliance() [2/2]

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

Instantiates a copy of an Alliance.

Parameters

alliance	must be an alliance instance

Definition at line 19 of file Alliance.cpp.

```
                                    {
    this->active = alliance.active;
    this->aID = alliance.aID;

    for (int i = 0; i < alliance.members.size(); i++)
        this->addCountry(alliance.members[i]->clone());

    for (int i = 0; i < alliance.production.size(); i++)
        this->addFactory(alliance.production[i]->clone());

    for (int i = 0; i < alliance.reserveEntities.size(); i++)
        this->addReserveEntity(alliance.reserveEntities[i]->clone());

    this->negotiator = NULL;
}
```

4.4.2.3 ∼Alliance()

```
Alliance::∼Alliance ( )
```

Destructor for the Alliance object.

Definition at line 35 of file Alliance.cpp.

```
                     {

    for (int i = 0; i < members.size(); i++)
        //delete members[i];

    if (this->negotiator != NULL) {
        this->negotiator->removeAlliance(this);

        if (this->negotiator->getNumAlliances() == 1)
            delete this->negotiator;
    }
}
```

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 52 of file Alliance.cpp.

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

4.4.3.2 addFactory()

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

Adds a factory into the production vector which holds factories.

Preconditions:

- f must be an Factory∗

Postconditions:

- Factory is added to the production vector

Parameters

factory	must be a Factory∗

Returns

void

Definition at line 78 of file Alliance.cpp.
```
{
    production.push_back(factory);
}
```

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

entity	must be an Entity∗

Returns

void

Definition at line 66 of file Alliance.cpp.
```
{
    reserveEntities.push_back(entity);
}
```

4.4.3.4 clone()

```
Alliance ∗ Alliance::clone ( )
```

Instantiates and returns a clone of the current Alliance.

Postconditions:

- Returns the clone of the current Alliance

Returns

Alliance∗ The alliance clone

Definition at line 114 of file Alliance.cpp.
```
                            {
    return new Alliance(*this);
}
```

4.4.3.5 considerPeace()

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

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

Preconditions:

- id must be an integer

Postconditions:

- Result of consideration returned in the form of a bool

Returns

bool

Definition at line 74 of file Alliance.cpp.
```
                               {
    return (rand() % 2 == 0);
}
```

4.4.3.6 getActive()

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

Definition at line 110 of file Alliance.cpp.
```
                          {
    return active;
}
```

4.4.3.7 getID()

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

Returns Alliance's aID.

Postconditions:

- Returns the aID

Returns

int The ID of the Alliance object

Definition at line 95 of file Alliance.cpp.

```
                         {
    return this->aID;
}
```

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 reseverves return amount available

Parameters

number	must be an int

Returns

vector<Entity*>*

Definition at line 56 of file Alliance.cpp.

```
                                                        {
    vector<Entity*> out;
    for (int i = 0; i < number && i < reserveEntities.size(); i++) {
        out.push_back(reserveEntities[i]);
        reserveEntities.erase(reserveEntities.begin() + i);
    }

    return out;
}
```

4.4.3.9 numRemainingEntities()

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

Definition at line 70 of file Alliance.cpp.

```
                                    {
    return reserveEntities.size();
}
```

4.4.3.10 offerPeace()

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

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

Postconditions:

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

Returns

bool

Definition at line 99 of file Alliance.cpp.

```
                          {

    if (this->negotiator->sendPeace(this)) //Send the peace deal to all the alliances fighting against
    {
        this->active = 3; //Number 3 means that Alliance chose to peacefully pull out of war
        return true;
    }

    return false;
}
```

4.4.3.11 runFactories()

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

Definition at line 82 of file Alliance.cpp.

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

4.4.3.12 setActiveStatus()

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

Sets variable active to the passed in parameter.

PreCondtions:

- active must be an a bool

PostConditions:

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

Parameters

ID	a bool parameter

4.4.3.13 setNegotiator()

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

Sets the entity negotiator.

Preconditions:

- n must be an Negotiator∗

Postconditions:

- Sets the negotiator of the Alliance object

Parameters

n	must be a Negotiator∗

Returns

void

Definition at line 48 of file Alliance.cpp.

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

4.4.3.14 surrender()

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

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

Postconditions:

- Sets the active variable to false
- Removes this alliance from the Negotiator vector

Returns

void

Definition at line 89 of file Alliance.cpp.

```
                            {
    this->active = 2; //Number 2 means that Alliance has surrendered

    this->negotiator->removeAlliance(this);
}
```

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

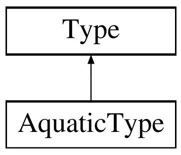
- Alliance.h
- Alliance.cpp

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.

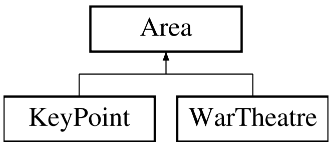
```
00007                                    {
00008     return "Aquatic";
00009 }
```

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
- virtual void addGeneral (General ∗general)=0

4.6.1 Detailed Description

Definition at line 8 of file Area.h.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 Area()

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

Instantiates the area.

Definition at line 5 of file Area.cpp.

```
                                    {
    this->areaName = areaName;
}
```

4.6.2.2 ∼Area()

```
Area::∼Area ( )  [virtual]
```

Destroys the area object.

Definition at line 9 of file Area.cpp.

```
00009 {}
```

4.6.3 Member Function Documentation

4.6.3.1 addGeneral()

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

Implemented in WarTheatre.

4.6.3.2 clone()

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

Implemented in KeyPoint, and WarTheatre.

4.6.3.3 getAreaName()

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

Get the Area Type object.

Returns

std::string reaturns the type

Definition at line 11 of file Area.cpp.

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

4.6.3.4 isKeyPoint()

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

Implemented in KeyPoint, and WarTheatre.

4.6.3.5 simulateBattle()

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

Implemented in KeyPoint, and WarTheatre.

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

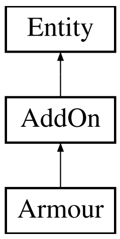
- Area.h
- Area.cpp

4.7 Armour Class Reference

Armour class.

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

Inheritance diagram for Armour:



Public Member Functions

- Armour (int value)

  Instantiates an Armour.
- void takeDamage (int damage)

  Decreases the entities' armour value (or health when their armour has depleted)
- void dealDamage (Entity ∗entity)

  Adds to the damage Entity objects inflict.
- AddOn ∗ clone ()

  Instantiates and returns a clone of the current Armour.

Additional Inherited Members

4.7.1 Detailed Description

Armour class.

Used to add protective armour to Entity objects.

Definition at line 11 of file Armour.h.

4.7.2 Constructor & Destructor Documentation

4.7.2.1 Armour()

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

Instantiates an Armour.

Parameters

value	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     if (getEntity() != NULL)
00020         armour->setEntity(entity->clone());
00021     return armour;
00022 }
```

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

entity	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

damage	must be an int

Returns

void

Implements AddOn.

Definition at line 5 of file Armour.cpp.

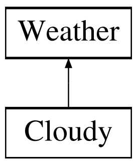
```
                              {
    if (value > 0) {
        value -= damage;
    } else {
        entity->takeDamage(damage);
    }
}
```

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 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 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.

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

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.

```
                             {
    return new Cloudy();
}
```

4.8.3.2 getWeather()

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

Returns string which tels 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.

```
                                      {
    return "Cloudy";
}
```

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

k	must be a KeyPoint∗

Returns

void

Implements Weather.

Definition at line 12 of file Cloudy.cpp.

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

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

- Cloudy.h
- Cloudy.cpp

4.9 Country Class Reference

Public Member Functions

- Country (std::string name)

  Instantiates the Country.
- Country ∗ clone ()

  Instantiates and returns a clone of the current Country.
- std::string getName () const

  Get the name of the country.
- int getID () const

  Get the id of the country.

4.9.1 Detailed Description

Definition at line 5 of file Country.h.

4.9.2 Constructor & Destructor Documentation

4.9.2.1 Country()

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

Instantiates the Country.

Parameters

name	must be a string

Definition at line 5 of file Country.cpp.

```
                              {
    this->name = name;
    this->id = rand() % 1000;
}
```

4.9.3 Member Function Documentation

4.9.3.1 clone()

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

Instantiates and returns a clone of the current Country.

Postconditions:

- Returns the clone of the current Country

Returns

Country∗ The country clone

Definition at line 11 of file Country.cpp.

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

4.9.3.2 getID()

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

Get the id of the country.

PostConditions:

- return the id the id of the country

Returns

int

Definition at line 19 of file Country.cpp.

```
{
    return this->id;
}
```

4.9.3.3 getName()

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

Get the name of the country.

PostConditions:

- Return the name of the country

Returns

string

Definition at line 15 of file Country.cpp.

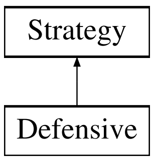
```
{
    return this->name;
}
```

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.

```
                           {
}
```

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.

```
                               {
    return new Defensive();
}
```

4.10.3.2 performStrat()

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

This function will perform an Defensive strategy.

Author

Antwi-Antwi

Parameters

keyPoint	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.

```
                                                        {

    int randomNumber = (rand() % 5) + 1;
    keyPoint->moveEntitiesInto(alliance, randomNumber);
}
```

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

- Defensive.h
- Defensive.cpp

4.11 EasySetup Class Reference

Public Member Functions

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

4.11.1 Detailed Description

Definition at line 11 of file EasySetup.h.

4.11.2 Constructor & Destructor Documentation

4.11.2.1 EasySetup()

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

Definition at line 21 of file EasySetup.cpp.

```
                        {
    saveArchive = new SaveArchive();
}
```

4.11.3 Member Function Documentation

4.11.3.1 loadPrevSave()

```
void EasySetup::loadPrevSave ( )
```

Definition at line 284 of file EasySetup.cpp.

```
                              {

    try{
        WarEngineMemento* saveFile = saveArchive->getLastSave();

        WarEngine::getInstance().loadSave(saveFile);
    }
    catch(const std::exception& error){

        std::cerr « error.what() « "\n";

    }
}
```

4.11.3.2 loadSpecificSave()

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

Definition at line 298 of file EasySetup.cpp.

```
                                             {

    try{

        WarEngineMemento* saveFile = saveArchive->getSave(name);

        WarEngine::getInstance().loadSave(saveFile);
    }
    catch(const std::out_of_range& range_error){

        std::cerr « range_error.what() « "\n";

    }
}
```

4.11.3.3 runSimulation()

```
void EasySetup::runSimulation ( )
```

Definition at line 267 of file EasySetup.cpp.

```
                               {

    WarEngine::getInstance().simulate();
}
```

4.11.3.4 saveSimulationSetup()

```
void EasySetup::saveSimulationSetup ( )
```

Definition at line 272 of file EasySetup.cpp.

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

4.11.3.5 setupSimulation()

```
void EasySetup::setupSimulation ( )
```

Definition at line 25 of file EasySetup.cpp.

```
00025                               {
00026     while (true)
00027     {
00028         cout « "Load simulation (L) or New Simulation (N): ";
00029         string selectedOption;
00030         cin » selectedOption;
00031         cin.ignore();
00032 
00033         if(toupper(selectedOption[0]) == 'L')
00034         {
00035             string saveName;
00036             cout « "Please enter the name of the save to be re-simulated" « endl;
00037             getline(cin, saveName); // getting the name of the save-archive
00038             try {
00039                 this->loadSpecificSave(saveName); // loading the save-archive
00040                 return; // will return if the above the function does not throw an exception
00041             } catch(const std::exception& exception) {
00042                 cout « "Error:  " « exception.what() « endl;
00043 
00044                 if (strcmp(exception.what(), "Save archive is empty") == 0) {
00045                     cout « "Please create new simulation" « endl;
00046                     goto setup;
00047 
00048                 } else if (strcmp(exception.what(), "No save with given name exists") == 0) {
00049 
00050                     cout « "Please enter the correct name of save-archive and try again or create new
      simulation" « endl;
00051                 }
00052             }
00053 
00054         } else if(toupper(selectedOption[0]) == 'N') {
00055             // setting up a new simulation
00056             goto setup;
00057         } else {
00058             cout « "Incorrect input:  Please enter (L) or (N)" « endl;
00059         }
00060     }
00061 
00062     setup:
00063         // Creating alliances and generals
00064         int numAlliesAndGenerals;
00065         cout « "Enter number of alliances:  ";
00066         cin » numAlliesAndGenerals;
00067 
00068         Alliance** alliances = new Alliance*[numAlliesAndGenerals];
00069         General** generals = new General*[numAlliesAndGenerals];
00070 
00071         int numCountries,
00072             numFactories;
00073         string  countryName,
00074                 factoryType,
00075                 selectedFactory,
00076                 selectedAddOn;
00077         Country* country;
```

```
        Type* type;
        AddOn* addOn;
        Factory* factory;

        Negotiator* negotiator = new Negotiator();

        for (int i = 0; i < numAlliesAndGenerals; i++) {
            alliances[i] = new Alliance();
            negotiator->addAlliance(alliances[i]);
            alliances[i]->setNegotiator(negotiator);
            WarEngine::getInstance().addAlliance(alliances[i]);

            cout « "Enter number of countries for Alliance " « alliances[i]->getID() « ":  ";
            cin » numCountries;
            cin.ignore();

            for (int k = 0; k < numCountries; k++) {
                cout « "Enter name of county " « k+1 « ":  ";
                getline(cin, countryName);
                country = new Country(countryName);
                alliances[i]->addCountry(country);
            }

            cout « "Enter number of factories for Alliance " « alliances[i]->getID() « ":  ";
            cin » numFactories;

            for (int k = 0; k < numFactories; k++) {
                retryType:
                cout « "Factory " « k+1 « " is of type Aquatic(Q), Aerial(E), or Terrain(T) : ";
                cin » factoryType;
                cin.ignore();

                if (toupper(factoryType[0]) == ’Q’) {
                    type = new AerialType;
                } else if (toupper(factoryType[0]) == ’E’) {
                    type = new AerialType;
                } else if (toupper(factoryType[0]) == ’T’) {
                    type = new TerrainType;
                } else {
                    cout « "Invalid type input!  Try again" « endl;
                    goto retryType;
                }

                retryAddOn:
                cout « "Select AddOn for factory " « k+1 « " Armour(A), Piercing(P) or None(N) : ";
                getline(cin, selectedAddOn);
                if (toupper(selectedAddOn[0]) == ’A’) {
                    int value;
                    cout « "Enter armour value:  ";
                    cin » value;
                    cin.ignore();
                    addOn = new Armour(value);
                } else if (toupper(selectedAddOn[0]) == ’P’) {
                    int value;
                    cout « "Enter piercing value:  ";
                    cin » value;
                    cin.ignore();
                    addOn = new Piercing(value);
                } else if (toupper(selectedAddOn[0] == ’N’)) {
                    addOn = NULL;
                } else {
                    cout « "Invalid AddOn input!  Try again" « endl;
                    goto retryAddOn;
                }

                retryFactory:
                cout « "Which factory is factory " « k+1 « " Vehicle(V), Personnel(P), or Support(S) :
 ";
                getline(cin, selectedFactory);
                if (toupper(selectedFactory[0]) == ’V’) {
                    factory = new VehicleFactory(type, addOn);
                } else if (toupper(selectedFactory[0]) == ’P’) {
                    factory = new PersonnelFactory(type, addOn);
                } else if (toupper(selectedFactory[0]) == ’S’) {
                    factory = new SupportFactory(type, addOn);
                } else {
                    cout « "Invalid factory input!  Try again" « endl;
                    goto retryFactory;
                }

                alliances[i]->addFactory(factory);
            }

            string selectedStrat;
            Strategy* strat;

            retryStrat:
```

```
            cout « "What is this Alliances generals strategy Passive(P), Defensive(D), or Aggressive(A) : ";
            getline(cin, selectedStrat);
            if (toupper(selectedStrat[0]) == 'P') {
                strat = new Passive();
            } else if (toupper(selectedStrat[0]) == 'D') {
                strat = new Defensive();
            } else if (toupper(selectedStrat[0]) == 'A') {
                strat = new Aggressive();
            } else {
                cout « "Invalid strategy input!  Try again" « endl;
                goto retryStrat;
            }

            generals[i] = new General(alliances[i], strat);
        }

        int factoryRun;
        cout « "How many production runs do you wish to perform:  ";
        cin » factoryRun;
        cin.ignore();
        for (int i = 0; i < numAlliesAndGenerals; i++) {
            for (int j = 0; j < factoryRun; j++) {
                alliances[i]->runFactories();
            }
        }

        // Creating main WarTheatre
        WarTheatre* mainBattleGround;
        cout « "Creating the main battle ground" « endl;
        string battleGroundName;
        cout « "Set main battle ground's name:  ";
        getline(cin, battleGroundName);
        mainBattleGround = new WarTheatre(battleGroundName);

        int sizeOfGrounds;
        cout « "Enter number of battle grounds in " « battleGroundName « " battle ground:  ";
        cin » sizeOfGrounds;
        cin.ignore();
        WarTheatre** battleGrounds = new WarTheatre*[sizeOfGrounds];

        // Creating sub WarTheatres
        for (int i = 0; i < sizeOfGrounds; i++) {
            battleGroundName.clear();
            cout « "Set battle ground " « i+1 « "'s name:  ";
            getline(cin, battleGroundName);
            battleGrounds[i] = new WarTheatre(battleGroundName);
        }

        vector<int> numKeyPoints;
        int numKeyPoint = 0;

        for (int i = 0; i < sizeOfGrounds; i++) {
            cout « "Enter number of key points in " « battleGrounds[i]->getAreaName() « " battle ground:  ";
            cin » numKeyPoint;
            cin.ignore();
            numKeyPoints.push_back(numKeyPoint);
            numKeyPoint = 0;
        }

        KeyPoint* keyPoint;
        string keyPointName;
        int numEntitiesInKeyPt;

        // Creating KeyPoints for the sub WarTheatres
        for (int i = 0; i < sizeOfGrounds; i++) {
            numKeyPoint = numKeyPoints[i];
            cout « "For " « battleGrounds[i]->getAreaName() « "'s key points" « endl;

            for (int k = 0; k < numKeyPoint; k++) {
                cout « "Set key point " « i+1 « "'s name:  ";
                getline(cin, keyPointName);
                keyPoint = new KeyPoint(keyPointName);

                for (int a = 0; a < numAlliesAndGenerals; a++) {
                    tryAgain:
                    cout « "There are " « alliances[a]->numRemainingEntities() « " for Alliance " « a+1 « endl;
                    cout « "How many would you like to place in " « keyPointName « " keypoint?  ";
                    cin » numEntitiesInKeyPt;
                    cin.ignore();

                    if (alliances[a]->numRemainingEntities() > 0 && alliances[a]->numRemainingEntities() < numEntitiesInKeyPt) {
                        cout « "You selected more than the available amount.  Try again " « endl;
                        goto tryAgain;
                    } else if (alliances[a]->numRemainingEntities() <= 0) {
                        continue;
                    } else {
                        keyPoint->moveEntitiesInto(alliances[a], numEntitiesInKeyPt);
                    }
                }

                battleGrounds[i]->addArea(keyPoint);
            }

            mainBattleGround->addArea(battleGrounds[i]);
        }

        for (int i = 0; i < numAlliesAndGenerals; i++) {
            mainBattleGround->addGeneral(generals[i]);
        }

        WarEngine::getInstance().setWarTheatre(mainBattleGround);
}
```

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

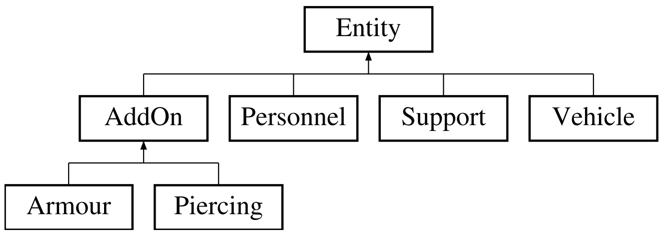
- EasySetup.h
- EasySetup.cpp

4.12 Entity Class Reference

Entity class.

`#include <Entity.h>`

Inheritance diagram for Entity:



Public Member Functions

- Entity ()

    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

4.12.1 Detailed Description

Entity class.

Used to simulate war entity objects.

Definition at line 13 of file Entity.h.

4.12.2 Constructor & Destructor Documentation

4.12.2.1 Entity() [1/2]

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

Instantiates the entity.

Definition at line 5 of file Entity.cpp.

```
                {
    health = 0;
    damage = 0;
    type = NULL;
}
```

4.12.2.2 Entity() [2/2]

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

Instantiates the entity.

Parameters

type	must be a Type∗

Definition at line 11 of file Entity.cpp.

```
                                              {
    this->health = health;
    this->damage = damage;
    this->type = type;
}
```

4.12.3 Member Function Documentation

4.12.3.1 clone()

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

Implemented in Armour, Personnel, Piercing, Support, and Vehicle.

4.12.3.2 dealDamage()

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

Implemented in Armour, Personnel, Piercing, Support, and Vehicle.

4.12.3.3 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.

```
                             {
    return this->alliance;
}
```

4.12.3.4 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.

```
                            {
    return this->damage;
}
```

4.12.3.5 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.

```
                            {
    return this->health;
}
```

4.12.3.6 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.

```
                            {
    return this->type;
}
```

4.12.3.7 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

alliance	must be a Alliance∗

Returns

void

Definition at line 29 of file Entity.cpp.

```
                                        {
    this->alliance = alliance;
}
```

4.12.3.8 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

damage	must be an int

Returns

void

Definition at line 45 of file Entity.cpp.
```
00045                                   {
00046     this->damage = damage;
00047 }
```

4.12.3.9 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

health	must be an int

Returns

void

Definition at line 37 of file Entity.cpp.
```
00037                                   {
00038     this->health = health;
00039 }
```

4.12.3.10 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

type	must be a Type∗

Returns

void

Definition at line 21 of file Entity.cpp.

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

4.12.3.11 takeDamage()

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

Implemented in Armour, Personnel, Piercing, Support, and Vehicle.

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

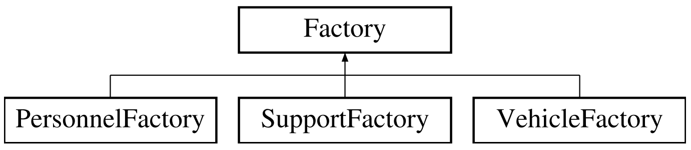
- Entity.h
- Entity.cpp

4.13 Factory Class Reference

Factory class.

`#include <Factory.h>`

Inheritance diagram for Factory:



Public Member Functions

- Factory (Type ∗type, AddOn ∗addOn)

  Instantiates the factory.
- ∼Factory ()

  Destroys the factory object.
- virtual Entity ∗ createEntity (Alliance ∗alliance)=0
- Type ∗ getType ()

  Returns factories type state.
- void setType (Type ∗type)

  Sets the factories type state.
- AddOn ∗ getAddOn ()

  Returns factories add ons.
- void setAddOns (AddOn ∗addOn)

  Sets the factories add ons.
- virtual Factory ∗ clone ()=0

4.13.1 Detailed Description

Factory class.

Used to instantiate Entity objects.

Definition at line 12 of file Factory.h.

4.13.2 Constructor & Destructor Documentation

4.13.2.1 Factory()

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

Instantiates the factory.

Parameters

type	must be a Type∗
addOn	must be a AddOn∗

Definition at line 3 of file Factory.cpp.

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

4.13.2.2 ∼Factory()

```
Factory::∼Factory ( )
```

Destroys the factory object.

Postconditions:

- All dynamic memory should be deallocated from the factory object

Definition at line 8 of file Factory.cpp.

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

4.13.3 Member Function Documentation

4.13.3.1 clone()

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

Implemented in PersonnelFactory, SupportFactory, and VehicleFactory.

4.13.3.2 createEntity()

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

Implemented in PersonnelFactory, SupportFactory, and VehicleFactory.

4.13.3.3 getAddOn()

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

Returns factories add ons.

Postconditions:

- Returns the add ons of the factory

Returns

AddOn∗ The decorators for the factory object

Definition at line 21 of file Factory.cpp.

```
                          {
    return this->addOn;
}
```

4.13.3.4 getType()

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

Returns factories type state.

Postconditions:

- Returns the type

Returns

Type∗ The type state of the factory object

Definition at line 13 of file Factory.cpp.

```
                        {
    return this->type;
}
```

4.13.3.5 setAddOns()

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

Sets the factories add ons.

Preconditions:

- addOns must be an AddOn∗

Postconditions:

- Sets the add ons of the factory object

Parameters

addOn	must be a AddOn∗

Returns

void

Definition at line 25 of file Factory.cpp.

```
                                      {
    this->addOn = addOn;
}
```

4.13.3.6 setType()

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

Sets the factories type state.

Preconditions:

- type must be an Type∗

Postconditions:

- Sets the type state of the factory object

Parameters

type	must be a Type∗

Returns

void

Definition at line 17 of file Factory.cpp.

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

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

- Factory.h
- Factory.cpp

4.14 General Class Reference

Public Member Functions

- General (Alliance ∗alliance, Strategy ∗stratety)

  Construct a new General object.
- void initiateStrategy (KeyPoint ∗keyPoint)

  The function intiates the strategy.
- bool setStrategy (Strategy ∗strategy)

  Set the Strategy object.
- Alliance ∗ getAlliance ()

  Returns the Alliance object.

4.14.1 Detailed Description

Definition at line 8 of file General.h.

4.14.2 Constructor & Destructor Documentation

4.14.2.1 General()

```
General::General (
            Alliance ∗ alliance,
            Strategy ∗ stratety )
```

Construct a new General object.

Parameters

alliance	must be an Alliance∗
stratety	must be a Strategy∗

Definition at line 3 of file General.cpp.

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

4.14.3 Member Function Documentation

4.14.3.1 getAlliance()

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

Returns the Alliance object.

PostConditions:

- Returns the alliance of the general

Returns

Alliance∗ The alliance that the general is associated

Definition at line 22 of file General.cpp.

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

4.14.3.2 initiateStrategy()

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

The function intiates the strategy.

Precondition:

- keyPoint muse be a KeyPoint∗

Parameters

keyPoint	must be a KeyPoint∗

Returns

void

Definition at line 9 of file General.cpp.

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

4.14.3.3 setStrategy()

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

Set the Strategy object.

PreConditons:

- strategy must be of type Strategy∗

PostConditions:

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

Parameters

strategy	

Returns

true if the setting the Strategy object was successful

false if the setting the Strategy object was unsuccessful

Definition at line 17 of file General.cpp.

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

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

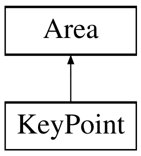
- General.h
- General.cpp

4.15 KeyPoint Class Reference

Keypoint class.

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

Inheritance diagram for KeyPoint:



Public Member Functions

- KeyPoint (std::string areaName)

  Instantiates the key point.
- KeyPoint (KeyPoint &keyPoint)

  Instantiates a copy of a KeyPoint.
- bool isKeyPoint ()

  Returns area type.
- void simulateBattle (Alliance ∗alliance)

  Simulate Battle with troops from the alliance passed in.
- void clearBattlefield (Alliance ∗alliance)

  Clears the battlefield of all deceased troops.
- void moveEntitiesInto (Alliance ∗alliance, int numTroops)

  Moves a specific alliances troops into this keypoint.
- void moveEntitiesOutOf (Alliance ∗alliance, int numTroops)

  Moves a specific alliances troops out of the keypoint.
- void addEntity (Entity ∗entity)

  Adds an enitity 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.

4.15.1 Detailed Description

Keypoint class.

Used to emulate strategic positions.

Definition at line 17 of file KeyPoint.h.

4.15.2 Constructor & Destructor Documentation

4.15.2.1 KeyPoint() [1/2]

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

Instantiates the key point.

Parameters

areaName	must be an string

4.15.2.2 KeyPoint() [2/2]

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

Instantiates a copy of a KeyPoint.

Parameters

keyPoint	must be an KeyPoint instance

Definition at line 15 of file KeyPoint.cpp.

```
                                  : Area(keyPoint.getAreaName()) {
    for (int i = 0; i < keyPoint.entities.size(); i++)
        this->addEntity(keyPoint.entities[i]->clone());

    weather = keyPoint.weather->clone();
}
```

4.15.2.3 ∼KeyPoint()

```
KeyPoint::∼KeyPoint ( )
```

Definition at line 22 of file KeyPoint.cpp.

```
                         {
    for (int i = 0; i < entities.size(); i++)
        delete entities[i];

    for (int i = 0; i < generals.size(); i++)
        delete generals[i];

    delete weather;
}
```

4.15.3 Member Function Documentation

4.15.3.1 addEntity()

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

Adds an enitity to the key point object.

Preconditions:

- entity must be an Entity∗

Postconditions:

- Add entity to key point

Parameters

entity	must be an Entity∗

Returns

void

Definition at line 113 of file KeyPoint.cpp.

```
                                    {
    entities.push_back(entity);
}
```

4.15.3.2 addGeneral()

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

Implements Area.

Definition at line 117 of file KeyPoint.cpp.

```
                                       {
    generals.push_back(general);
}
```

4.15.3.3 changeWeather()

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

Switches the Weather object to the next state.

Definition at line 140 of file KeyPoint.cpp.
```
                             {

    srand(time(0));

    int randomNum = 1 + (rand() % 10);
    std::string currWeather = this->weather->getWeather();

    if (currWeather == "Sunny" && randomNum > 6) // 60% chance of not changing weather from Sunny and
staying
        this->weather->handleChange(this);
    else if (currWeather == "Cloudy" && randomNum > 3) // 30% chance of not changing weather from
Cloudy and staying
        this->weather->handleChange(this);
    else if (currWeather == "Rainy" && randomNum > 1) // 10% chance of not changing weather from Rainy
and staying
        this->weather->handleChange(this);


}
```

4.15.3.4 clearBattlefield()

```
void KeyPoint::clearBattlefield (
            Alliance * alliance )
```

Clears the battlefield of all deceased troops.

Postconditions:

- Notify command centers of each troop who is killed

Parameters

alliance	must be an Alliance∗

Returns

void

Definition at line 61 of file KeyPoint.cpp.
```
                                                 {
    int destroyed = 0;
    double numUnits = 0;
    for (vector<Entity*>::iterator it = entities.begin();  it != entities.end(); ++it) {
        if ((*it)->getHealth() <= 0) {
            destroyed++;
            for (int i = 0; i < generals.size(); i++) {
                if (generals[i]->getAlliance() == (*it)->getAlliance()) {
                    generals[i]->initiateStrategy(this);
                    delete *it;
                    entities.erase(it);
                }
            }
        } else if ((*it)->getAlliance() == alliance) {
            numUnits++;
        }
    }

    // saving stats
    string stats = getAreaName() + ":\n";
    stats += "Key Point Satus:  ";
    if (numUnits / entities.size() >= 0.6) {
        stats += "Winning\n";
    } else if (numUnits / entities.size() >= 0.35) {
        stats += "Contested\n";
    } else {
        stats += "Losing\n";
    }
    stats += "Number of Entities Destroyed by Alliance:  " + to_string(destroyed);

    RoundStats::keyPointInformation.push_back(stats);
    RoundStats::numEntitiesDestroyed += destroyed;
}
```

4.15.3.5 clone()

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

Instantiates and returns a clone of the current Keypoint.

Postconditions:

- Returns the clone of the current Keypoint

Returns

Area∗ The Keypoint clone

Implements Area.

Definition at line 131 of file KeyPoint.cpp.

```
                         {
    return new KeyPoint(*this);
}
```

4.15.3.6 getWeather()

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

The weather at the current state is returned.

Returns

string The weather state

Definition at line 157 of file KeyPoint.cpp.

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

4.15.3.7 isKeyPoint()

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

Returns area type.

Postconditions:

- Returns true

Returns

bool The area type

Implements Area.

Definition at line 32 of file KeyPoint.cpp.

```
                                    {
    return true;
}
```

4.15.3.8 moveEntitiesInto()

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

Moves a specific alliances troops into this keypoint.

Preconditions:

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

Postconditions:

- Move troops to into this keypoint

Parameters

alliance	must be an Alliance∗
numTroops	must be an int

Returns

void

Definition at line 95 of file KeyPoint.cpp.

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

4.15.3.9 moveEntitiesOutOf()

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

Moves a specific alliances troops out of the keypoint.

Preconditions:

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

Postconditions:

- Move troops to reserve

Parameters

alliance	must be an Alliance∗
numTroops	must be an int

Returns

void

Definition at line 101 of file KeyPoint.cpp.

```
                                                                         {
    vector<Entity*>::iterator it = entities.begin();
    for (int i = 0; i < numTroops && it != entities.end(); i++) {
        for (; it != entities.end(); ++it) {
            if ((*it)->getAlliance() == alliance) {
                alliance->addReserveEntity(*it);
                entities.erase(it);
            }
        }
    }
}
```

4.15.3.10 removeGeneral()

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

Definition at line 121 of file KeyPoint.cpp.

```
                                               {
    for (vector<General*>::iterator it = generals.begin();  it != generals.end(); ++it) {
        if (*it == general) {
            delete *it;
            generals.erase(it);
            return;
        }
    }
}
```

4.15.3.11 setWeather()

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

Set the Weather object.

Preconditions:

- weather must be a Weather∗

Postconditions:

- must set the keyPoints weather state

Parameters

weather	must be a Weather∗

Returns

void

Definition at line 135 of file KeyPoint.cpp.

```
                                            {
    delete this->weather;
    this->weather = weather;
}
```

4.15.3.12 simulateBattle()

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

Simulate Battle with troops from the alliance passed in.

Preconditions:

- alliance must be an Alliance∗

Postconditions:

- Perform attacks on other alliance troops

Parameters

alliance	must be an Alliance∗

Returns

void

Implements Area.

Definition at line 36 of file KeyPoint.cpp.

```
                                                        {
    int numUnits = 0;
    for (int i = 0; i < entities.size(); i++) {
        if (entities[i]->getAlliance() == alliance) {
            numUnits++;
        }
    }

    if (numUnits != entities.size()) {
        for (int i = 0; i < entities.size(); i++) {
            if (entities[i]->getAlliance() == alliance) {
                int random;
                do {
                    random = rand() % entities.size();
                } while (entities[random]->getAlliance() == alliance);

                if (rand() % (int)(weather->getMultiplier() * 100) <= (int)(weather->getMultiplier() *
100))
                    entities[i]->dealDamage(entities[random]);
            }
        }
    }

    clearBattlefield(alliance);
}
```

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

- KeyPoint.h
- KeyPoint.cpp

4.16 Negotiator Class Reference

Public Member Functions

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

4.16.1 Detailed Description

Definition at line 6 of file Negotiator.h.

4.16.2 Constructor & Destructor Documentation

4.16.2.1 Negotiator()

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

Instantiates the Negotiator.

Definition at line 4 of file Negotiator.cpp.

```
00004 {}
```

4.16.2.2 ∼Negotiator()

```
Negotiator::∼Negotiator ( )
```

Destructor for the Negotiator object.

Definition at line 6 of file Negotiator.cpp.

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

4.16.3 Member Function Documentation

4.16.3.1 addAlliance()

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

Adds an alliance to the alliance vector.

Preconditions:

- newAlliance must be an Alliance pointer

Postconditions:

- Alliance is added to the vector

Returns

void

Definition at line 34 of file Negotiator.cpp.

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

4.16.3.2 getNumAlliances()

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

gets the number of Alliances in the negotiator

Postconditions:

- Returns the number of alliances

Returns

int Number of alliances

Definition at line 41 of file Negotiator.cpp.

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

4.16.3.3 removeAlliance()

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

Removes an alliance from the alliance vector.

Preconditions:

- oldAlliance must be an Alliance pointer

Postconditions:

- Alliance is removed from vector

Returns

void

Definition at line 24 of file Negotiator.cpp.

```
                                                       {

    for (int xx = 0; xx < alliances.size(); xx++)
    {
        if (alliances[xx]->getID() == oldAlliance->getID())
            alliances.erase( alliances.begin() + xx ); // Removes the specific alliances from this
       negotiator
    }

}
```

4.16.3.4 sendPeace()

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

Tries to offer peace to all the alliances in vector.

Preconditions:

- offerAlliance must be an Alliance pointer

Postconditions:

- Iterates through alliance vector and calls considerPeace for the enemies

Parameters

id	must be an int

Returns

bool

Definition at line 10 of file Negotiator.cpp.

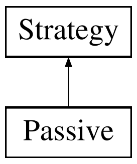
```
                                                                    {

    for (int yy = 0; yy < alliances.size(); yy++)
    {
        if (alliances[yy] != offerAlliance) {
            if (alliances[yy]->considerPeace() == false)
                return false; // There is at least one enemy alliances that does not want the peace
      deal
        }

    }

    return true; // All the alliances being fought against agreed to the peace deal
}
```

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

- Negotiator.h
- Negotiator.cpp

4.17 Passive Class Reference

Inheritance diagram for Passive:



Public Member Functions

- void performStrat (KeyPoint ∗keyPoint, Alliance ∗alliance)

  This function will perform a Passive strategy.
- Strategy ∗ clone ()

  Returns the clone of the Passive Strategy object.

Additional Inherited Members

4.17.1 Detailed Description

Definition at line 6 of file Passive.h.

4.17.2 Constructor & Destructor Documentation

4.17.2.1 Passive()

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

Definition at line 5 of file Passive.cpp.

```
00005 {}
```

4.17.3 Member Function Documentation

4.17.3.1 clone()

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

Returns the clone of the Passive Strategy object.

Returns

Strategy∗ The clone of the Passive Strategy object

Implements Strategy.

Definition at line 13 of file Passive.cpp.

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

4.17.3.2 performStrat()

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

This function will perform a Passive strategy.

Author

Antwi-Antwi

Parameters

keyPoint	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.

```
                                                            {

    int randomNumber = (rand() % 10) + 5;
    keyPoint->moveEntitiesOutOf(alliance, randomNumber);
}
```

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

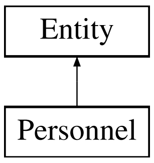
- Passive.h
- Passive.cpp

4.18 Personnel Class Reference

Personnel class.

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

Inheritance diagram for Personnel:



Public Member Functions

- Personnel (Type ∗type, int health=100, int damage=10)

  Instantiates the Personnel.
- void takeDamage (int damage)

  Removes health from the Personnel object.
- void dealDamage (Entity ∗entity)

  Inflicts damage onto another entity.
- Entity ∗ clone ()

  Returns the clone of the Personnel object.

4.18.1 Detailed Description

Personnel class.

Used to add addtional functionality to Entity objects.

Definition at line 11 of file Personnel.h.

4.18.2 Constructor & Destructor Documentation

4.18.2.1 Personnel()

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

Instantiates the Personnel.

Parameters

health	must be an int
damage	must be an int
type	must be a Type∗

Definition at line 5 of file Personnel.cpp.

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

4.18.3 Member Function Documentation

4.18.3.1 clone()

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

Returns the clone of the Personnel object.

Returns

Entity∗ Clone of personnel object

Implements Entity.

Definition at line 16 of file Personnel.cpp.

```
00016                            {
00017     Personnel* p;
00018     if (this->getType() == NULL) {
00019         p = new Personnel(NULL, this->getHealth(), this->getDamage());
00020     } else {
00021         p = new Personnel(this->getType()->clone(), this->getHealth(), this->getDamage());
00022     }
00023 
00024     p->setAlliance(this->getAlliance());
00025 
00026     return p;
00027 }
```

4.18.3.2 dealDamage()

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

Inflicts damage onto another entity.

Preconditions:

- entity must be an Entity∗

Postconditions:

- Reduces the health of the entity

Parameters

entity	must be an Entity∗

Returns

void

Implements Entity.

Definition at line 11 of file Personnel.cpp.

```
                                     {
    RoundStats::damageDone += getDamage();
    entity->takeDamage(getDamage());
}
```

4.18.3.3 takeDamage()

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

Removes health from the Personnel object.

Preconditions:

- damage must be an int

Postconditions:

- Reduces the health of the Personnel object

Parameters

damage	must be an int

Returns

void

Implements Entity.

Definition at line 7 of file Personnel.cpp.

```
00007                                    {
00008     setHealth(getHealth() - damage);
00009 }
```

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

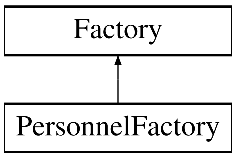
- Personnel.h
- Personnel.cpp

4.19 PersonnelFactory Class Reference

PersonnelFactory class.

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

Inheritance diagram for PersonnelFactory:



Public Member Functions

- PersonnelFactory (Type ∗type, AddOn ∗addOn)

  Instantiates the Personnel factory.
- Entity ∗ createEntity (Alliance ∗alliance)

  Instantiates and returns a Personnel for the given alliance.
- Factory ∗ clone ()

  Instantiates and returns a clone of the current Personnel factory.

4.19.1 Detailed Description

PersonnelFactory class.

Used to instantiate Personnel objects.

Definition at line 11 of file PersonnelFactory.h.

4.19.2 Constructor & Destructor Documentation

4.19.2.1 PersonnelFactory()

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

Instantiates the Personnel factory.

Parameters

type	must be a Type∗
addOn	must be a AddOn∗

Definition at line 5 of file PersonnelFactory.cpp.

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

4.19.3 Member Function Documentation

4.19.3.1 clone()

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

Instantiates and returns a clone of the current Personnel factory.

Postconditions:

- Returns the clone of the current Personnel factory

Returns

Factory∗ The Personnel factory clone

Implements Factory.

Definition at line 19 of file PersonnelFactory.cpp.

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

4.19.3.2 createEntity()

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

Instantiates and returns a Personnel for the given alliance.

Preconditions:

- alliance must be an Alliance∗

Postconditions:

- Returns the instantiated Personnel object with specific state

Parameters

alliance	must be a Alliance∗

Returns

Entity∗ The instatiated personnel

Implements Factory.

Definition at line 7 of file PersonnelFactory.cpp.

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

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

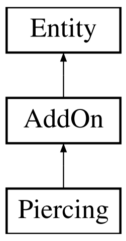
- PersonnelFactory.h
- PersonnelFactory.cpp

4.20 Piercing Class Reference

Piercing class.

`#include <Piercing.h>`

Inheritance diagram for Piercing:


$$\begin{tikzcd} \text{Entity} \\ \text{AddOn} \arrow[u] \\ \text{Piercing} \arrow[u] \end{tikzcd}$$

Public Member Functions

- Piercing (int value)

  Instantiates an Piercing.
- void takeDamage (int damage)

  Decreases the entities' armour value (or health when their armour has depleted)
- void dealDamage (Entity ∗entity)

  Adds to the damage Entity objects inflict.
- AddOn ∗ clone ()

  Instantiates and returns a clone of the current Piercing.

Additional Inherited Members

4.20.1 Detailed Description

Piercing class.

Used to add to the damage Entity objects inflict.

Definition at line 11 of file Piercing.h.

4.20.2 Constructor & Destructor Documentation

4.20.2.1 Piercing()

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

Instantiates an Piercing.

Parameters

value	must be an int

Definition at line 4 of file Piercing.cpp.

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

4.20.3 Member Function Documentation

4.20.3.1 clone()

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

Instantiates and returns a clone of the current Piercing.

Postconditions:

- Returns the clone of the current Piercing

Returns

Piercing∗ The Piercing clone

Implements AddOn.

Definition at line 16 of file Piercing.cpp.
```
                                {
    Piercing* piercing = new Piercing(value);
    if (getEntity() != NULL)
        piercing->setEntity(entity->clone());
    return piercing;
}
```

4.20.3.2 dealDamage()

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

Adds to the damage Entity objects inflict.

Preconditions:

- entity must be an Entity∗

Postconditions:

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

Parameters

entity	must be an Entity∗

Returns

void

Implements AddOn.

Definition at line 10 of file Piercing.cpp.
```
                                          {
    int sumValue = this->entity->getDamage() + value;
    entity->takeDamage(sumValue);
    RoundStats::damageDone += sumValue;
}
```

4.20.3.3 takeDamage()

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

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

Preconditions:

- damage must be an int

Postconditions:

- Does nothing

Parameters

damage	must be an int

Returns

void

Implements AddOn.

Definition at line 6 of file Piercing.cpp.

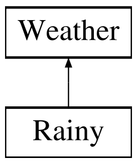
```
00006                                 {
00007     entity->takeDamage(damage);
00008 }
```

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

- Piercing.h
- Piercing.cpp

4.21 Rainy Class Reference

Inheritance diagram for Rainy:



Public Member Functions

- Rainy ()

  Instantiates the Runny object of the state pattern.
- std::string getWeather ()

  Returns string which tels us the weather.
- void handleChange (KeyPoint ∗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

4.21.1 Detailed Description

Definition at line 6 of file Rainy.h.

4.21.2 Constructor & Destructor Documentation

4.21.2.1 Rainy()

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

Instantiates the Runny object of the state pattern.

Definition at line 4 of file Rainy.cpp.

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

4.21.3 Member Function Documentation

4.21.3.1 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.

```
                         {
    return new Rainy();
}
```

4.21.3.2 getWeather()

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

Returns string which tels 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.

```
                          {
    return "Rainy";
}
```

4.21.3.3 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

k	must be a KeyPoint∗

Returns

void

Implements Weather.

Definition at line 12 of file Rainy.cpp.

```
                                     {
    Sunny* newWeather = new Sunny();
    k->setWeather(newWeather);
}
```

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

- Rainy.h
- Rainy.cpp

4.22 RoundStats Class Reference

Static Public Member Functions

- static void clearStats ()
- static string toString ()

Static Public Attributes

- static int numEntitiesCreated = 0
- static int numEntitiesDestroyed = 0
- static int damageDone = 0
- static vector< string > keyPointInformation
- static vector< string > entityMovementInformation

4.22.1 Detailed Description

Definition at line 9 of file RoundStats.h.

4.22.2 Member Function Documentation

4.22.2.1 clearStats()

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

Definition at line 9 of file RoundStats.cpp.

```
00009                           {
00010     numEntitiesCreated = 0;
00011     numEntitiesDestroyed = 0;
00012     keyPointInformation.clear();
00013     entityMovementInformation.clear();
00014 }
```

4.22.2.2 toString()

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

Definition at line 16 of file RoundStats.cpp.

```
                                 {
    string out = "Number of Entities Created:  " + to_string(numEntitiesCreated) + "\n";
    out += "Number of Entities Destroyed by Alliance:  " + to_string(numEntitiesDestroyed) + "\n";
    out += "Damage Given by Alliance:  " + to_string(damageDone) + "\n";

    out += "\nKey Point Round Information:\n";
    for (int i = 0; i < keyPointInformation.size(); i++)
        out += keyPointInformation[i] + "\n";

    out += "\nMovement Round Information:\n";
    for (int i = 0; i < entityMovementInformation.size(); i++)
        out += entityMovementInformation[i] + "\n";

    return out;
}
```

4.22.3 Member Data Documentation

4.22.3.1 damageDone

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

Definition at line 13 of file RoundStats.h.

4.22.3.2 entityMovementInformation

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

Definition at line 15 of file RoundStats.h.

4.22.3.3 keyPointInformation

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

Definition at line 14 of file RoundStats.h.

4.22.3.4 numEntitiesCreated

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

Definition at line 11 of file RoundStats.h.

4.22.3.5 numEntitiesDestroyed

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

Definition at line 12 of file RoundStats.h.

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

- RoundStats.h
- RoundStats.cpp

4.23 SaveArchive Class Reference

Stores a list of mementos containing simulation state.

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

Public Member Functions

- SaveArchive ()

  Instantiates the SaveArchive class.
- void addNewSave (std::string newSaveName, WarEngineMemento ∗newSave)

  Adds a new save to the list of stored mementos.
- WarEngineMemento ∗ getLastSave ()

  Returns the last saved memento.
- WarEngineMemento ∗ getSave (std::string name)

  Returns the last saved memento. Preconditions:
- void clearSaveList ()

  Erases all saved mementos from the list of saves. Postconditions:
- void deleteSave (std::string name)

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

4.23.1 Detailed Description

Stores a list of mementos containing simulation state.

Definition at line 11 of file SaveArchive.h.

4.23.2 Constructor & Destructor Documentation

4.23.2.1 SaveArchive()

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

Instantiates the SaveArchive class.

Definition at line 3 of file SaveArchive.cpp.

```
00003 {}
```

4.23.3 Member Function Documentation

4.23.3.1 addNewSave()

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

Adds a new save to the list of stored mementos.

Preconditions:

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

Postconditions:

- Adds a new memento to list of saves

Parameters

newSave	must be a WarEngineMemento∗
newSaveName	must be a string

Returns

void

Definition at line 5 of file SaveArchive.cpp.

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

4.23.3.2 clearSaveList()

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

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

- Clears all elements in the saveList vector

Returns

void

Definition at line 35 of file SaveArchive.cpp.

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

4.23.3.3 deleteSave()

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

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

- name must be a string in date/time format

Postconditions:

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

Parameters

name	a string

Returns

void

Exceptions

std::out_of_range	save archive is empty

Definition at line 39 of file SaveArchive.cpp.

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

4.23.3.4 getLastSave()

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

Returns the last saved memento.

Postconditions:

- Returns the last element in the saveList vector

Returns

WarEngineMemento∗

Exceptions

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

Definition at line 9 of file SaveArchive.cpp.

```
                                         {

    if(saveList.size() == 0){
        throw "Save archive is empty.";
    }

    WarEngineMemento* lastSave = saveList.begin()->second;

    saveList.erase( saveList.begin() );

    return lastSave;
}
```

4.23.3.5 getSave()

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

Returns the last saved memento. Preconditions:

- name must be a string

Postconditions:

- Returns the last element in the saveList vector

Parameters

name	a string

Returns

WarEngineMemento∗

Exceptions

std::out_of_range	save archive is empty

Definition at line 22 of file SaveArchive.cpp.

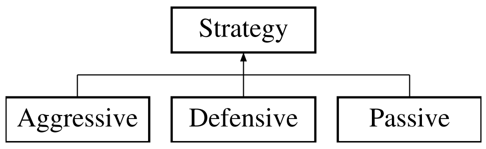
```
00022                                                     {
00023     if(saveList.size() == 0){
00024         std::__throw_out_of_range("Save archive is empty");
00025     }
00026 
00027     auto iter = saveList.find(name);
00028 
00029     if(iter == saveList.end())
00030         std::__throw_invalid_argument("No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

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

- SaveArchive.h
- SaveArchive.cpp

4.24 Strategy Class Reference

Inheritance diagram for Strategy:



Public Member Functions

- Strategy ()

  Construct a new Strategy object.
- ∼Strategy ()

  Destroy the Strategy object.
- virtual void performStrat (KeyPoint ∗keyPoint, Alliance ∗alliance)=0
- virtual Strategy ∗ clone ()=0

Protected Attributes

- std::string strategy

4.24.1 Detailed Description

Definition at line 10 of file Strategy.h.

4.24.2 Constructor & Destructor Documentation

4.24.2.1 Strategy()

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

Construct a new Strategy object.

Definition at line 7 of file Strategy.cpp.

```
00007 {}
```

4.24.2.2 ∼Strategy()

```
Strategy::∼Strategy ( )
```

Destroy the Strategy object.

Definition at line 9 of file Strategy.cpp.

```
00009 {}
```

4.24.3 Member Function Documentation

4.24.3.1 clone()

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

Implemented in Aggressive, Defensive, and Passive.

4.24.3.2 performStrat()

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

Implemented in Aggressive, Defensive, and Passive.

4.24.4 Member Data Documentation

4.24.4.1 strategy

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

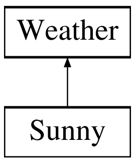
Definition at line 13 of file Strategy.h.

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

- Strategy.h
- Strategy.cpp

4.25 Sunny Class Reference

Inheritance diagram for Sunny:



Public Member Functions

- Sunny ()

  Instantiates the Sunny object of the state pattern.
- virtual std::string getWeather ()

  Returns string which tells us the weather.
- virtual void handleChange (KeyPoint ∗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

4.25.1 Detailed Description

Definition at line 8 of file Sunny.h.

4.25.2 Constructor & Destructor Documentation

4.25.2.1 Sunny()

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

Instantiates the Sunny object of the state pattern.

Definition at line 4 of file Sunny.cpp.

```
               {
    this->multiplier = 1.0;
}
```

4.25.3 Member Function Documentation

4.25.3.1 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.

```
                            {
    return new Sunny();
}
```

4.25.3.2 getWeather()

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

Returns string which tells us the weather.

Postconditions:

- Returns the wether of ths current state

Returns

std::string which is the current state

Implements Weather.

Definition at line 8 of file Sunny.cpp.

```
                                  {
    return "Sunny";
}
```

4.25.3.3 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

k	must be a KeyPoint*

Returns

void

Implements Weather.

Definition at line 12 of file Sunny.cpp.

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

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

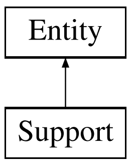
- Sunny.h
- Sunny.cpp

4.26 Support Class Reference

Support class.

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

Inheritance diagram for Support:



Public Member Functions

- Support (Type ∗type, int health=1000, int damage=30)

  Instantiates the support.
- void takeDamage (int damage)

  Removes health from the support object.
- void dealDamage (Entity ∗entity)

  Inflicts damage onto another entity.
- Entity ∗ clone ()

  Returns the clone of the Support object.

4.26.1 Detailed Description

Support class.

Used to add addtional functionality to Entity objects.

Definition at line 11 of file Support.h.

4.26.2 Constructor & Destructor Documentation

4.26.2.1 Support()

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

Instantiates the support.

Parameters

health	must be an int
damage	must be an int
type	must be a Type∗

Definition at line 4 of file Support.cpp.

```
00004 :  Entity(type, health, damage) {}
```

4.26.3 Member Function Documentation

4.26.3.1 clone()

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

Returns the clone of the Support object.

Returns

Entity∗ The clone of the support object

Implements Entity.

Definition at line 15 of file Support.cpp.

```
                    {
    Support* s;
    if (this->getType() == NULL) {
        s = new Support(NULL, this->getHealth(), this->getDamage());
    } else {
        s = new Support(this->getType()->clone(), this->getHealth(), this->getDamage());
    }

    s->setAlliance(this->getAlliance());

    return s;
}
```

4.26.3.2 dealDamage()

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

Inflicts damage onto another entity.

Preconditions:

- entity must be an Entity∗

Postconditions:

- Reduces the health of the entity

Parameters

entity	must be an Entity∗

Returns

void

Implements Entity.

Definition at line 6 of file Support.cpp.

```
                                    {
        RoundStats::damageDone += getDamage();
        entity->takeDamage(getDamage());
    }
```

4.26.3.3 takeDamage()

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

Removes health from the support object.

Preconditions:

- damage must be an int

Postconditions:

- Reduces the health of the support object

Parameters

damage	must be an int

Returns

void

Implements Entity.

Definition at line 11 of file Support.cpp.

```
00011                               {
00012     this->setHealth(this->getHealth() - damage);
00013 }
```

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

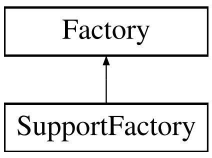
- Support.h
- Support.cpp

4.27 SupportFactory Class Reference

SupportFactory class.

`#include <SupportFactory.h>`

Inheritance diagram for SupportFactory:



Public Member Functions

- SupportFactory (Type ∗type, AddOn ∗addOn)

  Instantiates the support factory.
- Entity ∗ createEntity (Alliance ∗alliance)

  Instantiates and returns a support for the given alliance.
- Factory ∗ clone ()

  Instantiates and returns a clone of the current support factory.

4.27.1 Detailed Description

SupportFactory class.

Used to instantiate Support objects.

Definition at line 11 of file SupportFactory.h.

4.27.2 Constructor & Destructor Documentation

4.27.2.1 SupportFactory()

```
SupportFactory::SupportFactory (
            Type ∗ type,
            AddOn ∗ addOn )
```

Instantiates the support factory.

Parameters

type	must be a Type∗
addOn	must be a AddOn∗

Definition at line 4 of file SupportFactory.cpp.

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

4.27.3 Member Function Documentation

4.27.3.1 clone()

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

Instantiates and returns a clone of the current support factory.

Postconditions:

- Returns the clone of the current support factory

Returns

Factory∗ The support factory clone

Implements Factory.

Definition at line 18 of file SupportFactory.cpp.

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

4.27.3.2 createEntity()

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

Instantiates and returns a support for the given alliance.

Preconditions:

- alliance must be an Alliance∗

Postconditions:

- Returns the instantiated support object with specific state

Parameters

alliance	must be a Alliance∗

Returns

Entity∗ The instatiated support

Implements Factory.

Definition at line 6 of file SupportFactory.cpp.

```
                                                  {
    Support* s = new Support(getType()->clone());
    s->setAlliance(alliance);
    if (getAddOn() != NULL) {
        AddOn* personnelAddOn = getAddOn()->clone();
        personnelAddOn->setEntity(s);
        return personnelAddOn;
    } else {
        return s;
    }
}
```

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

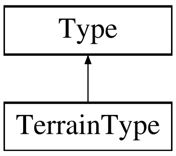
- SupportFactory.h
- SupportFactory.cpp

4.28 TerrainType Class Reference

TerrainType class.

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

Inheritance diagram for TerrainType:



Public Member Functions

- TerrainType ()

    Instantiates the terrain type.
- string getTypeDesc ()

    Returns terrain type description.
- Type ∗ clone ()

4.28.1 Detailed Description

TerrainType class.

Used to define Entity objects as terrain type.

Definition at line 11 of file TerrainType.h.

4.28.2 Constructor & Destructor Documentation

4.28.2.1 TerrainType()

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

Instantiates the terrain type.

Definition at line 3 of file TerrainType.cpp.
```
00003 {}
```

4.28.3 Member Function Documentation

4.28.3.1 clone()

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

Implements Type.

Definition at line 9 of file TerrainType.cpp.

```
                       {
    return new TerrainType();
}
```

4.28.3.2 getTypeDesc()

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

Returns terrain type description.

Postconditions:

- Returns the terrain type

Returns

string The terrain type string

Implements Type.

Definition at line 5 of file TerrainType.cpp.

```
                             {
    return "Terrain";
}
```

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

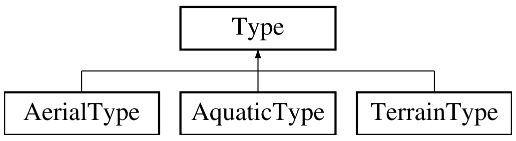
- TerrainType.h
- TerrainType.cpp

4.29 Type Class Reference

Type class.

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

Inheritance diagram for Type:



Public Member Functions

- Type ()

  Instantiates the type.
- virtual string getTypeDesc ()=0
- virtual Type ∗ **clone** ()=0

4.29.1 Detailed Description

Type class.

Used to define Entity objects type.

Definition at line 13 of file Type.h.

4.29.2 Constructor & Destructor Documentation

4.29.2.1 Type()

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

Instantiates the type.

Definition at line 3 of file Type.cpp.

```
00003 {}
```

4.29.3 Member Function Documentation

4.29.3.1 getTypeDesc()

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

Implemented in AerialType, AquaticType, and TerrainType.

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

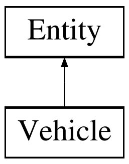
- Type.h
- Type.cpp

4.30 Vehicle Class Reference

Vehicle class.

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

Inheritance diagram for Vehicle:



Public Member Functions

- Vehicle (Type ∗type, int health=500, int damage=10)

  Instantiates the vehicle.
- void takeDamage (int damage)

  Removes health from the vehicle object.
- void dealDamage (Entity ∗entity)

  Inflicts damage onto another entity.
- Entity ∗ clone ()

  Returns the clone of the Vehicle object.

4.30.1 Detailed Description

Vehicle class.

Used to add addtional functionality to Entity objects.

Definition at line 11 of file Vehicle.h.

4.30.2 Constructor & Destructor Documentation

4.30.2.1 Vehicle()

```
Vehicle::Vehicle (
              Type * type,
              int health = 500,
              int damage = 10 )
```

Instantiates the vehicle.

Parameters

health	must be an int
damage	must be an int
type	must be a Type∗

Definition at line 4 of file Vehicle.cpp.

```
00004 :  Entity(type, health, damage) {}
```

4.30.3 Member Function Documentation

4.30.3.1 clone()

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

Returns the clone of the Vehicle object.

Returns

Entity∗ The clone of the vehicle object

Implements Entity.

Definition at line 15 of file Vehicle.cpp.

```
00015                        {
00016     Vehicle* v;
00017     if (this->getType() == NULL) {
00018         v = new Vehicle(NULL, this->getHealth(), this->getDamage());
00019     } else {
00020         v = new Vehicle(this->getType()->clone(), this->getHealth(), this->getDamage());
00021     }
00022 
00023     v->setAlliance(this->getAlliance());
00024 
00025     return v;
00026 }
```

4.30.3.2 dealDamage()

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

Inflicts damage onto another entity.

Preconditions:

- entity must be an Entity∗

Postconditions:

- Reduces the health of the entity

Parameters

entity	must be an Entity∗

Returns

void

Implements Entity.

Definition at line 10 of file Vehicle.cpp.
```
00010                                   {
00011     RoundStats::damageDone += getDamage();
00012     entity->takeDamage(getDamage());
00013 }
```

4.30.3.3 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

damage	must be an int

Returns

void

Implements Entity.

Definition at line 6 of file Vehicle.cpp.
```
00006                               {
00007     setHealth(getHealth() - damage);
00008 }
```

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

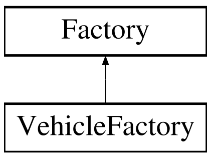
- Vehicle.h
- Vehicle.cpp

4.31 VehicleFactory Class Reference

VehicleFactory class.

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

Inheritance diagram for VehicleFactory:



Public Member Functions

- VehicleFactory (Type ∗type, AddOn ∗addOn)

  Instantiates the vehicle factory.
- Entity ∗ createEntity (Alliance ∗alliance)

  Instantiates and returns a vehicle for the given alliance.
- Factory ∗ clone ()

  Instantiates and returns a clone of the current vehicle factory.

4.31.1 Detailed Description

VehicleFactory class.

Used to instantiate Vehicle objects.

Definition at line 10 of file VehicleFactory.h.

4.31.2 Constructor & Destructor Documentation

4.31.2.1 VehicleFactory()

```
VehicleFactory::VehicleFactory (
            Type ∗ type,
            AddOn ∗ addOn )
```

Instantiates the vehicle factory.

Parameters

type	must be a Type∗
addOn	must be a AddOn∗

Definition at line 4 of file VehicleFactory.cpp.

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

4.31.3 Member Function Documentation

4.31.3.1 clone()

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

Instantiates and returns a clone of the current vehicle factory.

Postconditions:

- Returns the clone of the current vehicle factory

Returns

Factory∗ The vehicle factory clone

Implements Factory.

Definition at line 18 of file VehicleFactory.cpp.

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

4.31.3.2 createEntity()

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

Instantiates and returns a vehicle for the given alliance.

Preconditions:

- alliance must be an Alliance∗

Postconditions:

- Returns the instantiated vehicle object with specific state

Parameters

alliance	must be a Alliance∗

Returns

Vehicle∗ The instatiated vehicle

Implements Factory.

Definition at line 6 of file VehicleFactory.cpp.

```
                                                            {
    Vehicle* v = new Vehicle(getType()->clone());
    v->setAlliance(alliance);
    if (getAddOn() != NULL) {
        AddOn* personnelAddOn = getAddOn()->clone();
        personnelAddOn->setEntity(v);
        return personnelAddOn;
    } else {
        return v;
    }
}
```

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

- VehicleFactory.h
- VehicleFactory.cpp

4.32 WarEngine Class Reference

```
#include <WarEngine.h>
```

Public Member Functions

- WarEngineMemento ∗ saveState ()

  Captures current state of simulation via member variables and creates WarEngineMemento instance storing all relevant members in WarEngineState.
- void loadSave (WarEngineMemento ∗save)

  Takes in an instance of saved WarEngine states and sets current instance's member variables to memento state.
- void setWarTheatre (WarTheatre ∗battleGround)

  Sets the state's area to passed in battleGround parameter.
- void addAlliance (Alliance ∗alliance)
- void simulate ()

  Simulates the war.

Static Public Member Functions

- static WarEngine & getInstance ()

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

Protected Member Functions

- WarEngine ()

  Constructor for class. Is responsible for ensuring only a single instance of class exists.
- WarEngine (const WarEngine &)

  Parameterized constructor for class.
- WarEngine & operator= (const WarEngine &)

  Overloaded operator = for class.
- ∼WarEngine ()

  Destrcutor for class responsible for freeing all allocated memory.

4.32.1 Detailed Description

Class that contains all information regarding current simulation. Only one instance of class is allowed.

Definition at line 12 of file WarEngine.h.

4.32.2 Constructor & Destructor Documentation

4.32.2.1 WarEngine() [1/2]

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

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

Definition at line 5 of file WarEngine.cpp.

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

4.32.2.2 WarEngine() [2/2]

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

Parameterized constructor for class.

Parameters

warEngine&	An anonymous warEngine reference.

Postconditions:

- parameter must be of type WarEngine&

Definition at line 33 of file WarEngine.h.

```
{};
```

4.32.2.3 ∼WarEngine()

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

Destrcutor for class responsible for freeing all allocated memory.

Definition at line 23 of file WarEngine.cpp.

```
                         {
    delete this->state;
}
```

4.32.3 Member Function Documentation

4.32.3.1 addAlliance()

```
void WarEngine::addAlliance (
            Alliance * alliance )
```

Definition at line 57 of file WarEngine.cpp.

```
                                        {
    state->alliances.push_back(alliance);
}
```

4.32.3.2 getInstance()

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

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

Returns

WarEngine&

Definition at line 18 of file WarEngine.cpp.

```
                              {
    static WarEngine uniqueInstance_;
    return uniqueInstance_;
}
```

4.32.3.3 loadSave()

```
void WarEngine::loadSave (
            WarEngineMemento * save )
```

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

Preconditions:

- Save must be of type WarEngineMemento*

Postconditions:

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

Parameters

save	must be a WarEngineMemento∗

Returns

void

Definition at line 13 of file WarEngine.cpp.
```
{
    delete this->state;
    this->state = save->getState();
}
```

4.32.3.4 operator=()

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

Overloaded operator = for class.

Definition at line 38 of file WarEngine.h.
```
{ return *this; };
```

4.32.3.5 saveState()

```
WarEngineMemento * WarEngine::saveState ( )
```

Captures current state of simulation via member variables and creates WarEngineMemento instance storing all relevant members in WarEngineState.

Returns

WarEngineMemento∗

Definition at line 9 of file WarEngine.cpp.
```
{
    return new WarEngineMemento(state->clone());
}
```

4.32.3.6 setWarTheatre()

```
void WarEngine::setWarTheatre (
            WarTheatre * battleGround )
```

Sets the state's area to passed in battleGround parameter.

Parameters

battleGround	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 53 of file WarEngine.cpp.

```
                                                  {
    state->setArea(battleGround);
}
```

4.32.3.7 simulate()

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

Simulates the war.

PostConditions:

- Will simulate the war by running the engine

Returns

void

Definition at line 28 of file WarEngine.cpp.

```
                          {
    vector<Alliance*> alliances = this->state->getAlliances();

    for(int i = 0; i < alliances.size(); i++) {
        if (alliances[i]->getActive() == 1) {
            RoundStats::clearStats();
            state->getArea()->simulateBattle(alliances[i]);
            cout « "========================================================================" « endl;
            cout « "Alliance " « alliances[i]->getID() « ":" « endl;

            if (alliances[i]->getID() == 2) {
                cout « "Status:  Surrendered" « endl;
            } else if (alliances[i]->getID() == 3) {
                cout « "Status:  Found Peace" « endl;
            } else {
                cout « "Status:  Active" « endl;
            }

            cout « RoundStats::toString() « endl;
            cout « "========================================================================" « endl;
        }
    }

}
```

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

- WarEngine.h
- WarEngine.cpp

4.33 WarEngineMemento Class Reference

```
#include <WarEngineMemento.h>
```

Friends

- class WarEngine

4.33.1 Detailed Description

Class that encapsulates and externalises WarEngine State.

Definition at line 15 of file WarEngineMemento.h.

4.33.2 Friends And Related Function Documentation

4.33.2.1 WarEngine

```
friend class WarEngine [friend]
```

Definition at line 17 of file WarEngineMemento.h.

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

- WarEngineMemento.h
- WarEngineMemento.cpp

4.34 WarEngineState Class Reference

Class for storing current state of entire simulation.

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

Friends

- class WarEngine

4.34.1 Detailed Description

Class for storing current state of entire simulation.

Class contains member variables areas which stores a vector of all war theatres and keypoints as well as a vector of all alliances in current simulation.

Definition at line 17 of file WarEngineState.h.

4.34.2 Friends And Related Function Documentation

4.34.2.1 WarEngine

```
friend class WarEngine  [friend]
```

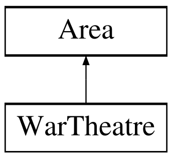
Definition at line 19 of file WarEngineState.h.

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

- WarEngineState.h
- WarEngineState.cpp

4.35 WarTheatre Class Reference

Inheritance diagram for WarTheatre:



Public Member Functions

- WarTheatre (std::string areaName)

  Instantiates the war theatre.
- ∼WarTheatre ()

  Destroys the war theatre object.
- bool isKeyPoint ()

  Returns area type.
- void simulateBattle (Alliance ∗alliance)

  Simulate Battle with troops from the alliance passed in.
- void addArea (Area ∗area)

  Adds an area to the war theatre object.
- void addGeneral (General ∗general)

  Adds a general to all the points held by the WarTheatre.
- WarTheatre ∗ clone ()

  Instantiates and returns a clone of the current war theatre.

4.35.1 Detailed Description

Definition at line 10 of file WarTheatre.h.

4.35.2 Constructor & Destructor Documentation

4.35.2.1 WarTheatre()

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

Instantiates the war theatre.

Definition at line 5 of file WarTheatre.cpp.

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

4.35.2.2 ∼WarTheatre()

```
WarTheatre::∼WarTheatre ( )
```

Destroys the war theatre object.

Postconditions:

- All dynamic memory should be deallocated from the war theatre object

Definition at line 7 of file WarTheatre.cpp.

```
00007                            {
00008     for (int i = 0; i < areas.size(); i++)
00009         delete areas[i];
00010 }
```

4.35.3 Member Function Documentation

4.35.3.1 addArea()

```
void WarTheatre::addArea (
             Area * area )
```

Adds an area to the war theatre object.

Preconditions:

- area must be an Area∗

Postconditions:

- Add area to war theatre object

Parameters

area	must be an Area∗

Returns

void

Definition at line 21 of file WarTheatre.cpp.

```
                                    {
    areas.push_back(area);
}
```

4.35.3.2 addGeneral()

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

Adds a general to all the points held by the WarTheatre.

Precoditions:

- general must be a General∗

Postconditions:

- Add general to all points

Parameters

general	must be a General∗

Implements Area.

Definition at line 34 of file WarTheatre.cpp.

```
                                             {
    for (int i = 0; i < areas.size(); i++)
        areas[i]->addGeneral(general);
}
```

4.35.3.3 clone()

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

Instantiates and returns a clone of the current war theatre.

Postconditions:

- Returns the clone of the current war theatre

Returns

WarTheatre∗ The war theatre clone

Implements Area.

Definition at line 25 of file WarTheatre.cpp.

```
                          {
    WarTheatre* w = new WarTheatre(getAreaName());

    for (int i = 0; i < areas.size(); i++)
        w->addArea(areas[i]->clone());

    return w;
}
```

4.35.3.4 isKeyPoint()

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

Returns area type.

Postconditions:

- Returns false

Returns

bool The area type

Implements Area.

Definition at line 12 of file WarTheatre.cpp.

```
                         {
    return false;
}
```

4.35.3.5 simulateBattle()

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

Simulate Battle with troops from the alliance passed in.

Preconditions:

- alliance must be an Alliance∗

Postconditions:

- Call attacks function of areas

Parameters

alliance	must be an Alliance∗

Returns

void

Implements Area.

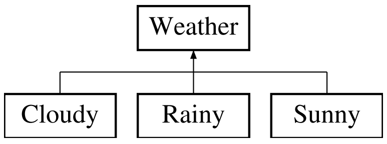
Definition at line 16 of file WarTheatre.cpp.
```
                                                    {
    for (int i = 0; i < areas.size(); i++)
        areas[i]->simulateBattle(alliance);
}
```

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

- WarTheatre.h
- WarTheatre.cpp

4.36 Weather Class Reference

Inheritance diagram for Weather:



Public Member Functions

- Weather ()

  Instantiates the Weather object.
- ∼Weather ()

  Destructor for the Weather object.
- double getMultiplier ()

  Returns double which shows the weather multiplier.
- virtual void handleChange (KeyPoint ∗k)=0
- virtual std::string getWeather ()=0
- virtual Weather ∗ clone ()=0

Protected Attributes

- double multiplier

4.36.1 Detailed Description

Definition at line 9 of file Weather.h.

4.36.2 Constructor & Destructor Documentation

4.36.2.1 Weather()

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

Instantiates the Weather object.

Definition at line 3 of file Weather.cpp.

```
00003 {}
```

4.36.2.2 ∼Weather()

```
Weather::∼Weather ( )
```

Destructor for the Weather object.

Definition at line 5 of file Weather.cpp.

```
00005 {}
```

4.36.3 Member Function Documentation

4.36.3.1 clone()

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

Implemented in Cloudy, Rainy, and Sunny.

4.36.3.2 getMultiplier()

```
double Weather::getMultiplier ( )
```

Returns double which shows the weather multiplier.

Postconditions:

- Returns the double multiplier

Returns

double which is the multiplier

Definition at line 7 of file Weather.cpp.

```
{
    return this->multiplier;
}
```

4.36.3.3 getWeather()

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

Implemented in Cloudy, Rainy, and Sunny.

4.36.3.4 handleChange()

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

Implemented in Cloudy, Rainy, and Sunny.

4.36.4 Member Data Documentation

4.36.4.1 multiplier

```
double Weather::multiplier  [protected]
```

Definition at line 12 of file Weather.h.

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

- Weather.h
- Weather.cpp

Chapter 5

File Documentation

5.1 AddOn.cpp

```cpp
#include "AddOn.h"

AddOn::AddOn(int value):  Entity() {
    this->value = value;
    entity = NULL;
}

void AddOn::setValue(int value) {
    this->value = value;
}

int AddOn::getValue() {
    return value;
}

void AddOn::setEntity(Entity* entity) {
    this->entity = entity;
}

Entity* AddOn::getEntity() {
    return this->entity;
}
```

5.2 AddOn.h

```cpp
#ifndef ADDON_H
#define ADDON_H
#include "Entity.h"

class AddOn :  public Entity {

protected:
    int value;
    Entity* entity;

public:
    AddOn(int value);

    void setValue(int value);

    int getValue();

    void setEntity(Entity* entity);

    Entity* getEntity();

    virtual void takeDamage(int damage) = 0;

    virtual void dealDamage(Entity* entity) = 0;

    virtual AddOn* clone() = 0;
};

#endif
```

5.3 AddOnTest.h

```
#include <stdexcept>
#include "AddOn.h"
#include "Piercing.h"
#include "Armour.h"
#include "Personnel.h"
#include "TerrainType.h"
#include "gtest/gtest.h"

namespace {

    // Tests General AddOn Functionality

    // Tests AddOn setValue()
    // ============ Precondition Testing ============
    // Test Precondition Negative
    TEST(AddOnSetValueTest, TestPreconditionNegative) {
        Armour* a = new Armour(10);
        try {
            a->setValue(-5);
            FAIL();
        } catch (std::invalid_argument& err) {
            EXPECT_EQ(err.what(), std::string("value must be greater than zero"));
        } catch (...)  {
            FAIL();
        }
    }

    // Test Precondition Positive
    TEST(AddOnSetValueTest, TestPreconditionPositive) {
        Armour* a = new Armour(5);
        a->setValue(5);
        EXPECT_EQ(5, a->getValue());
    }

    // Test Precondition Bounds
    TEST(AddOnSetValueTest, TestPreconditionBounds) {
        Armour* a = new Armour(5);
        try {
            a->setValue(0);
            FAIL();
        } catch (std::invalid_argument& err) {
            EXPECT_EQ(err.what(), std::string("value must be greater than zero"));
        } catch (...)  {
            FAIL();
        }
    }

    // ============ Positive Testing ============
    // Test Preconditions Bounds
    TEST(AddOnSetValueTest, PositiveTesting) {
        Armour* a = new Armour(5);

        a->setValue(5);
        EXPECT_EQ(5, a->getValue());

        a->setValue(10);
        EXPECT_EQ(10, a->getValue());

        a->setValue(20);
        EXPECT_EQ(20, a->getValue());

        a->setValue(55);
        EXPECT_EQ(55, a->getValue());

        a->setValue(3);
        EXPECT_EQ(3, a->getValue());

        a->setValue(100);
        EXPECT_EQ(100, a->getValue());
    }

    // Tests AddOn setEntity()
    // ============ Positive Testing ============
    // Test Preconditions Bounds
    TEST(AddOnSetEntityTest, PositiveTesting) {
        Armour* a = new Armour(5);

        Personnel* p = new Personnel(new TerrainType(), 100, 10);
        a->setEntity(p);
        EXPECT_EQ(p, a->getEntity());

        Personnel* m = new Personnel(new TerrainType(), 100, 10);
        a->setEntity(m);
        EXPECT_EQ(m, a->getEntity());

        Personnel* n = new Personnel(new TerrainType(), 100, 10);
        a->setEntity(n);
        EXPECT_EQ(n, a->getEntity());
    }

    // Tests Armour AddOn Functionality

    // Tests Armour takeDamage()
    // ============ Precondition Testing ============
    // Test Precondition Negative
    TEST(ArmourTakeDamageTest, TestPreconditionNegative) {
        Armour* a = new Armour(10);
        try {
            a->takeDamage(-5);
            FAIL();
        } catch (std::invalid_argument& err) {
            EXPECT_EQ(err.what(), std::string("damage must be greater than zero"));
        } catch (...)  {
            FAIL();
        }
    }

    // Test Precondition Positive
    TEST(ArmourTakeDamageTest, TestPreconditionPositive) {
        Armour* a = new Armour(10);
        Personnel* p = new Personnel(new TerrainType(), 100, 10);

        a->setEntity(p);
        a->takeDamage(10);
        EXPECT_EQ(0, a->getValue());
        EXPECT_EQ(100, p->getHealth());
    }

    // Test Precondition Bounds
    TEST(ArmourTakeDamageTest, TestPreconditionBounds) {
        Armour* a = new Armour(10);
        try {
            a->takeDamage(0);
            FAIL();
        } catch (std::invalid_argument& err) {
            EXPECT_EQ(err.what(), std::string("damage must be greater than zero"));
        } catch (...)  {
            FAIL();
        }
    }

    // ============ Positive Testing ============
    // Test Preconditions Bounds
    TEST(ArmourTakeDamageTest, PositiveTesting) {
        Armour* a = new Armour(20);
        Personnel* p = new Personnel(new TerrainType(), 100, 10);
        a->setEntity(p);

        a->takeDamage(10);
        EXPECT_EQ(10, a->getValue());
        EXPECT_EQ(100, p->getHealth());

        a->takeDamage(10);
        EXPECT_EQ(0, a->getValue());
        EXPECT_EQ(100, p->getHealth());

        a->takeDamage(10);
        EXPECT_EQ(0, a->getValue());
        EXPECT_EQ(90, p->getHealth());
    }

    // Tests Armour dealDamage()
    // ============ Positive Testing ============
    // Test Preconditions Bounds
    TEST(ArmourDealDamageTest, PositiveTesting) {
        Armour* a = new Armour(10);
        Personnel* p = new Personnel(new TerrainType(), 100, 10);
        a->setEntity(p);
        Personnel* x = new Personnel(new TerrainType(), 100, 10);

        a->dealDamage(x);
        EXPECT_EQ(10, a->getValue());
        EXPECT_EQ(90, x->getHealth());

        a->dealDamage(x);
        EXPECT_EQ(10, a->getValue());
        EXPECT_EQ(80, x->getHealth());

        a->dealDamage(x);
        EXPECT_EQ(10, a->getValue());
        EXPECT_EQ(70, x->getHealth());
    }

    // Tests Piercing AddOn Functionality

    // Tests Piercing takeDamage()
    // ============ Positive Testing ============
    // Test Preconditions Bounds
    TEST(PiercingTakeDamageTest, PositiveTesting) {
        Piercing* pi = new Piercing(10);
        Personnel* p = new Personnel(new TerrainType(), 100, 10);
        pi->setEntity(p);

        pi->takeDamage(10);
        EXPECT_EQ(10, pi->getValue());
        EXPECT_EQ(90, p->getHealth());

        pi->takeDamage(10);
        EXPECT_EQ(10, pi->getValue());
        EXPECT_EQ(80, p->getHealth());

        pi->takeDamage(10);
        EXPECT_EQ(10, pi->getValue());
        EXPECT_EQ(70, p->getHealth());
    }

    // Tests Piercing dealDamage()
    // ============ Positive Testing ============
    // Test Preconditions Bounds
    TEST(PiercingDealDamageTest, PositiveTesting) {
        Piercing* pi = new Piercing(10);
        Personnel* p = new Personnel(new TerrainType(), 100, 10);
        pi->setEntity(p);
        Personnel* x = new Personnel(new TerrainType(), 100, 10);

        pi->dealDamage(x);
        EXPECT_EQ(10, pi->getValue());
        EXPECT_EQ(80, x->getHealth());

        pi->dealDamage(x);
        EXPECT_EQ(10, pi->getValue());
        EXPECT_EQ(60, x->getHealth());

        pi->dealDamage(x);
        EXPECT_EQ(10, pi->getValue());
        EXPECT_EQ(40, x->getHealth());
    }
}
```

5.4 AerialType.cpp

```
#include "AerialType.h"

AerialType::AerialType() {}

string AerialType::getTypeDesc() {
    return "Aerial";
}

Type* AerialType::clone() {
    return new AerialType();
}
```

5.5 AerialType.h

```
#ifndef AERIALTYPE_H
#define AERIALTYPE_H

#include "Type.h"

class AerialType :  public Type {

public:
    AerialType();

    string getTypeDesc();

    Type* clone();
};

#endif
```

5.6 Aggressive.cpp

```
#include "Aggressive.h"
#include "KeyPoint.h"

Aggressive::Aggressive() {}

void Aggressive::performStrat(KeyPoint* keyPoint, Alliance* alliance) {
    int randomNumber = (rand() % 10) + 5;
    keyPoint->moveEntitiesInto(alliance, randomNumber);
}

Strategy* Aggressive::clone() {
    return new Aggressive();
}
```

5.7 Aggressive.h

```
#ifndef AGGRESSIVE_H
#define AGGRESSIVE_H
#include "Strategy.h"

class Aggressive :  public Strategy {

public:
    Aggressive();

    void performStrat(KeyPoint* keyPoint, Alliance* alliance);

    Strategy* clone();
};

#endif
```

5.8 Alliance.cpp

```
#include "Alliance.h"
#include "Negotiator.h"
#include "Entity.h"
#include "RoundStats.h"
#include <time.h>
#include <iostream>

using namespace std;

int Alliance::totalNum = 0;

Alliance::Alliance() {
    this->active = 1;
    this->aID = totalNum++;
    this->negotiator = NULL;
    srand(time(0));
}

Alliance::Alliance(Alliance& alliance) {
    this->active = alliance.active;
    this->aID = alliance.aID;

    for (int i = 0; i < alliance.members.size(); i++)
        this->addCountry(alliance.members[i]->clone());

    for (int i = 0; i < alliance.production.size(); i++)
        this->addFactory(alliance.production[i]->clone());

    for (int i = 0; i < alliance.reserveEntities.size(); i++)
        this->addReserveEntity(alliance.reserveEntities[i]->clone());

    this->negotiator = NULL;
}

Alliance::~Alliance() {

    for (int i = 0; i < members.size(); i++)
        //delete members[i];

    if (this->negotiator != NULL) {
        this->negotiator->removeAlliance(this);

        if (this->negotiator->getNumAlliances() == 1)
```

```
            delete this->negotiator;
    }
}

void Alliance::setNegotiator(Negotiator* negotiator) {
    this->negotiator = negotiator;
}

void Alliance::addCountry(Country* nation) {
    members.push_back(nation);
}

vector<Entity*> Alliance::getReserveEntities(int number) {
    vector<Entity*> out;
    for (int i = 0; i < number && i < reserveEntities.size(); i++) {
        out.push_back(reserveEntities[i]);
        reserveEntities.erase(reserveEntities.begin() + i);
    }

    return out;
}

void Alliance::addReserveEntity(Entity* entity) {
    reserveEntities.push_back(entity);
}

int Alliance::numRemainingEntities() {
    return reserveEntities.size();
}

bool Alliance::considerPeace() {
    return (rand() % 2 == 0);
}

void Alliance::addFactory(Factory* factory) {
    production.push_back(factory);
}

void Alliance::runFactories() {
    for (int i = 0; i < production.size(); i++) {
        RoundStats::numEntitiesCreated++;
        reserveEntities.push_back(production[i]->createEntity(this));
    }
}

void Alliance::surrender() {
    this->active = 2; //Number 2 means that Alliance has surrendered

    this->negotiator->removeAlliance(this);
}

int Alliance::getID() {
    return this->aID;
}

bool Alliance::offerPeace() {

    if (this->negotiator->sendPeace(this)) //Send the peace deal to all the alliances fighting against
    {
        this->active = 3; //Number 3 means that Alliance chose to peacefully pull out of war
        return true;
    }

    return false;
}

int Alliance::getActive() {
    return active;
}

Alliance* Alliance::clone() {
    return new Alliance(*this);
}
```

5.9 Alliance.h

```
#ifndef ALLIANCE_H
#define ALLIANCE_H
#include "Country.h"
#include "Factory.h"
#include "Country.h"
#include <vector>

```

```
class Negotiator;
class Entity;

using namespace std;

class Alliance {

private:
    static int totalNum;
    int aID;
    vector<Factory*> production;
    Negotiator* negotiator;
    vector<Country*> members;
    int active;
    vector<Entity*> reserveEntities;

public:
    Alliance();

    Alliance(Alliance& alliance);

    ~Alliance();

    void setNegotiator(Negotiator* newNegotiator);

    void addCountry(Country* nation);

    vector<Entity*> getReserveEntities(int number);

    void addReserveEntity(Entity* entity);

    int numRemainingEntities();

    bool considerPeace();

    void addFactory(Factory* factory);

    void runFactories();

    void surrender();

    int getID();

    bool offerPeace();

    Alliance* clone();

    void setActiveStatus(bool active);

    int getActive();
};

#endif
```

5.10 AquaticType.cpp

```
#include "AquaticType.h"

using namespace std;

AquaticType::AquaticType() {}

string AquaticType::getTypeDesc() {
    return "Aquatic";
}

Type* AquaticType::clone() {
    return new AquaticType();
}
```

5.11 AquaticType.h

```
#ifndef AQUATICTYPE_H
#define AQUATICTYPE_H

#include "Type.h"

class AquaticType :  public Type {

```

```
public:
    AquaticType();

    string getTypeDesc();

    Type* clone();
};

#endif
```

5.12 Area.cpp

```
#include "Area.h"

using namespace std;

Area::Area(string areaName) {
    this->areaName = areaName;
}

Area::~Area() {}

std::string Area::getAreaName()const {
    return areaName;
}
```

5.13 Area.h

```
#ifndef AREA_H
#define AREA_H
#include <string>
#include "Alliance.h"

class General;

class Area {

private:
    std::string areaName;

public:
    Area(std::string areaName);

    virtual ~Area();

    virtual bool isKeyPoint() = 0;

    virtual void simulateBattle(Alliance* alliance) = 0;

    std::string getAreaName() const;

    virtual Area* clone() = 0;

    virtual void addGeneral(General* general) = 0;
};

#endif
```

5.14 Armour.cpp

```
#include "Armour.h"

Armour::Armour(int value) :  AddOn(value) {}

void Armour::takeDamage(int damage) {
    if (value > 0) {
        value -= damage;
    } else {
        entity->takeDamage(damage);
    }
}

void Armour::dealDamage(Entity* entity) {
    this->entity->dealDamage(entity);
}
```

```

AddOn* Armour::clone() {
    Armour* armour = new Armour(value);
    if (getEntity() != NULL)
        armour->setEntity(entity->clone());
    return armour;
}
```

5.15 Armour.h

```
#ifndef ARMOUR_H
#define ARMOUR_H
#include "AddOn.h"
#include "Entity.h"

class Armour :  public AddOn {


public:
    Armour(int value);

    void takeDamage(int damage);

    void dealDamage(Entity* entity);

    AddOn* clone();
};

#endif
```

5.16 Cloudy.cpp

```
#include "Cloudy.h"
#include "Rainy.h"

Cloudy::Cloudy():  Weather() {
    this->multiplier = 0.75;
}

std::string Cloudy::getWeather() {
    return "Cloudy";
}

void Cloudy::handleChange(KeyPoint* k) {
    Rainy* newWeather = new Rainy();
    k->setWeather(newWeather);
}

Weather* Cloudy::clone() {
    return new Cloudy();
}
```

5.17 Cloudy.h

```
#ifndef CLOUDY_H
#define CLOUDY_H
#include "Weather.h"
#include <string>

class Cloudy :  public Weather {

public:
    Cloudy();

    std::string getWeather();

    void handleChange(KeyPoint* k);

    Weather* clone();
};

#endif
```

5.18 Country.cpp

```
#include "Country.h"

using namespace std;

Country::Country(std::string name){
    this->name = name;
    this->id = rand() % 1000;
}


Country* Country::clone(){
    return new Country(this->name);
}

string Country::getName()const{
    return this->name;
}

int Country::getID()const{
    return this->id;
}


```

5.19 Country.h

```
#ifndef COUNTRY_H
#define COUNTRY_H
#include <string>

class Country {

private:
    std::string name;
    int id;

public:
    Country(std::string name);

    Country* clone();

    std::string getName() const;

    int getID() const;

};

#endif
```

5.20 Defensive.cpp

```
#include "Defensive.h"

Defensive::Defensive() {

}

void Defensive::performStrat(KeyPoint* keyPoint, Alliance* alliance) {

    int randomNumber = (rand() % 5) + 1;
    keyPoint->moveEntitiesInto(alliance, randomNumber);
}

Strategy* Defensive::clone() {
    return new Defensive();
}
```

5.21 Defensive.h

```
#ifndef DEFENSIVE_H
#define DEFENSIVE_H
#include "Strategy.h"
#include "KeyPoint.h"
```

```
#include "Alliance.h"
#include "Personnel.h"
class Defensive :  public Strategy {


public:
    Defensive();

    void performStrat(KeyPoint* keyPoint, Alliance* alliance);

    Strategy* clone();
};

#endif
```

5.22 EasySetup.cpp

```
#include "EasySetup.h"
#include <string.h>
#include "Alliance.h"
#include "Country.h"
#include "AquaticType.h"
#include "AerialType.h"
#include "TerrainType.h"
#include "Piercing.h"
#include "Armour.h"
#include "PersonnelFactory.h"
#include "VehicleFactory.h"
#include "SupportFactory.h"
#include "KeyPoint.h"
#include "WarTheatre.h"
#include "Passive.h"
#include "Aggressive.h"
#include "Defensive.h"
#include "WarEngine.h"
#include "Negotiator.h"

EasySetup::EasySetup() {
    saveArchive = new SaveArchive();
}

void EasySetup::setupSimulation() {
    while (true)
    {
        cout « "Load simulation (L) or New Simulation (N): ";
        string selectedOption;
        cin » selectedOption;
        cin.ignore();

        if(toupper(selectedOption[0]) == 'L')
        {
            string saveName;
            cout « "Please enter the name of the save to be re-simulated" « endl;
            getline(cin, saveName); // getting the name of the save-archive
            try {
                this->loadSpecificSave(saveName); // loading the save-archive
                return; // will return if the above the function does not throw an exception
            } catch(const std::exception& exception) {
                cout « "Error:  " « exception.what() « endl;

                if (strcmp(exception.what(), "Save archive is empty") == 0) {
                    cout « "Please create new simulation" « endl;
                    goto setup;

                } else if (strcmp(exception.what(), "No save with given name exists") == 0) {

                    cout « "Please enter the correct name of save-archive and try again or create new simulation" « endl;
                }
            }

        } else if(toupper(selectedOption[0]) == 'N') {
            // setting up a new simulation
            goto setup;
        } else {
            cout « "Incorrect input:  Please enter (L) or (N)" « endl;
        }
    }

    setup:
        // Creating alliances and generals
        int numAlliesAndGenerals;
        cout « "Enter number of alliances:  ";
        cin » numAlliesAndGenerals;

        Alliance** alliances = new Alliance*[numAlliesAndGenerals];
        General** generals = new General*[numAlliesAndGenerals];

        int numCountries,
            numFactories;
        string  countryName,
                factoryType,
                selectedFactory,
                selectedAddOn;
        Country* country;
        Type* type;
        AddOn* addOn;
        Factory* factory;

        Negotiator* negotiator = new Negotiator();

        for (int i = 0; i < numAlliesAndGenerals; i++) {
            alliances[i] = new Alliance();
            negotiator->addAlliance(alliances[i]);
            alliances[i]->setNegotiator(negotiator);
            WarEngine::getInstance().addAlliance(alliances[i]);

            cout « "Enter number of countries for Alliance " « alliances[i]->getID() « ":  ";
            cin » numCountries;
            cin.ignore();

            for (int k = 0; k < numCountries; k++) {
                cout « "Enter name of county " « k+1 « ":  ";
                getline(cin, countryName);
                country = new Country(countryName);
                alliances[i]->addCountry(country);
            }

            cout « "Enter number of factories for Alliance " « alliances[i]->getID() « ":  ";
            cin » numFactories;

            for (int k = 0; k < numFactories; k++) {
                retryType:
                cout « "Factory " « k+1 « " is of type Aquatic(Q), Aerial(E), or Terrain(T) : ";
                cin » factoryType;
                cin.ignore();

                if (toupper(factoryType[0]) == 'Q') {
                    type = new AerialType;
                } else if (toupper(factoryType[0]) == 'E') {
                    type = new AerialType;
                } else if (toupper(factoryType[0]) == 'T') {
                    type = new TerrainType;
                } else {
                    cout « "Invalid type input!  Try again" « endl;
                    goto retryType;
                }

                retryAddOn:
                cout « "Select AddOn for factory " « k+1 « " Armour(A), Piercing(P) or None(N) : ";
                getline(cin, selectedAddOn);
                if (toupper(selectedAddOn[0]) == 'A') {
                    int value;
                    cout « "Enter armour value:  ";
                    cin » value;
                    cin.ignore();
                    addOn = new Armour(value);
                } else if (toupper(selectedAddOn[0]) == 'P') {
                    int value;
                    cout « "Enter piercing value:  ";
                    cin » value;
                    cin.ignore();
                    addOn = new Piercing(value);
                } else if (toupper(selectedAddOn[0] == 'N')) {
                    addOn = NULL;
                } else {
                    cout « "Invalid AddOn input!  Try again" « endl;
                    goto retryAddOn;
                }

                retryFactory:
                cout « "Which factory is factory " « k+1 « " Vehicle(V), Personnel(P), or Support(S) :
";
                getline(cin, selectedFactory);
                if (toupper(selectedFactory[0]) == 'V') {
                    factory = new VehicleFactory(type, addOn);
                } else if (toupper(selectedFactory[0]) == 'P') {
                    factory = new PersonnelFactory(type, addOn);
                } else if (toupper(selectedFactory[0]) == 'S') {
                    factory = new SupportFactory(type, addOn);
                } else {
                    cout « "Invalid factory input!  Try again" « endl;
                    goto retryFactory;
                }

                alliances[i]->addFactory(factory);
            }

            string selectedStrat;
            Strategy* strat;

            retryStrat:
            cout « "What is this Alliances generals strategy Passive(P), Defensive(D), or Aggressive(A) : ";
            getline(cin, selectedStrat);
            if (toupper(selectedStrat[0]) == 'P') {
                strat = new Passive();
            } else if (toupper(selectedStrat[0]) == 'D') {
                strat = new Defensive();
            } else if (toupper(selectedStrat[0]) == 'A') {
                strat = new Aggressive();
            } else {
                cout « "Invalid strategy input!  Try again" « endl;
                goto retryStrat;
            }

            generals[i] = new General(alliances[i], strat);
        }

        int factoryRun;
        cout « "How many production runs do you wish to perform:  ";
        cin » factoryRun;
        cin.ignore();
        for (int i = 0; i < numAlliesAndGenerals; i++) {
            for (int j = 0; j < factoryRun; j++) {
                alliances[i]->runFactories();
            }
        }

        // Creating main WarTheatre
        WarTheatre* mainBattleGround;
        cout « "Creating the main battle ground" « endl;
        string battleGroundName;
        cout « "Set main battle ground's name:  ";
        getline(cin, battleGroundName);
        mainBattleGround = new WarTheatre(battleGroundName);

        int sizeOfGrounds;
        cout « "Enter number of battle grounds in " « battleGroundName « " battle ground:  ";
        cin » sizeOfGrounds;
        cin.ignore();
        WarTheatre** battleGrounds = new WarTheatre*[sizeOfGrounds];

        // Creating sub WarTheatres
        for (int i = 0; i < sizeOfGrounds; i++) {
            battleGroundName.clear();
            cout « "Set battle ground " « i+1 « "'s name:  ";
            getline(cin, battleGroundName);
            battleGrounds[i] = new WarTheatre(battleGroundName);
        }

        vector<int> numKeyPoints;
        int numKeyPoint = 0;

        for (int i = 0; i < sizeOfGrounds; i++) {
            cout « "Enter number of key points in " « battleGrounds[i]->getAreaName() « " battle ground:  ";
            cin » numKeyPoint;
            cin.ignore();
            numKeyPoints.push_back(numKeyPoint);
            numKeyPoint = 0;
        }

        KeyPoint* keyPoint;
        string keyPointName;
        int numEntitiesInKeyPt;

        // Creating KeyPoints for the sub WarTheatres
        for (int i = 0; i < sizeOfGrounds; i++) {
            numKeyPoint = numKeyPoints[i];
            cout « "For " « battleGrounds[i]->getAreaName() « "'s key points" « endl;

            for (int k = 0; k < numKeyPoint; k++) {
                cout « "Set key point " « i+1 « "'s name:  ";
                getline(cin, keyPointName);
                keyPoint = new KeyPoint(keyPointName);

                for (int a = 0; a < numAlliesAndGenerals; a++) {
                    tryAgain:
                    cout « "There are " « alliances[a]->numRemainingEntities() « " for Alliance " « a+1 « endl;
                    cout « "How many would you like to place in " « keyPointName « " keypoint?  ";
                    cin » numEntitiesInKeyPt;
                    cin.ignore();

                    if (alliances[a]->numRemainingEntities() > 0 && alliances[a]->numRemainingEntities() < numEntitiesInKeyPt) {
                        cout « "You selected more than the available amount.  Try again " « endl;
                        goto tryAgain;
                    } else if (alliances[a]->numRemainingEntities() <= 0) {
                        continue;
                    } else {
                        keyPoint->moveEntitiesInto(alliances[a], numEntitiesInKeyPt);
                    }
                }

                battleGrounds[i]->addArea(keyPoint);
            }

            mainBattleGround->addArea(battleGrounds[i]);
        }

        for (int i = 0; i < numAlliesAndGenerals; i++) {
            mainBattleGround->addGeneral(generals[i]);
        }

        WarEngine::getInstance().setWarTheatre(mainBattleGround);
}

void EasySetup::runSimulation() {

    WarEngine::getInstance().simulate();
}

void EasySetup::saveSimulationSetup() {

    // Getting the name of the save
    cout « "Please enter name of save:  ";
    string saveName;
    getline(cin, saveName);

    // saving the current state of the simulation
    saveArchive->addNewSave(saveName, WarEngine::getInstance().saveState());

}

void EasySetup::loadPrevSave() {

    try{
        WarEngineMemento* saveFile = saveArchive->getLastSave();

        WarEngine::getInstance().loadSave(saveFile);
    }
    catch(const std::exception& error){

        std::cerr « error.what() « "\n";

    }
}

void EasySetup::loadSpecificSave(string name) {

    try{

        WarEngineMemento* saveFile = saveArchive->getSave(name);

        WarEngine::getInstance().loadSave(saveFile);
    }
    catch(const std::out_of_range& range_error){

        std::cerr « range_error.what() « "\n";

    }
}
```

5.23 EasySetup.h

```
#ifndef EASYSETUP_H
#define EASYSETUP_H
#include <iostream>
```

```
#include <cctype>
#include <string>
#include <vector>
#include "SaveArchive.h"

using namespace std;

class EasySetup
{
    private:
        SaveArchive* saveArchive;

    public:
        EasySetup();
        void setupSimulation();
        void runSimulation();
        void loadPrevSave();
        void loadSpecificSave(std::string name);
        void saveSimulationSetup();
};

#endif
```

5.24 Entity.cpp

```
#include "Entity.h"
#include "Alliance.h"


Entity::Entity() {
    health = 0;
    damage = 0;
    type = NULL;
}

Entity::Entity(Type* type, int health, int damage) {
    this->health = health;
    this->damage = damage;
    this->type = type;
}

Type* Entity::getType() {
    return this->type;
}

void Entity::setType(Type* type) {
    this->type = type;
}

Alliance* Entity::getAlliance() {
    return this->alliance;
}

void Entity::setAlliance(Alliance* alliance) {
    this->alliance = alliance;
}

int Entity::getHealth() {
    return this->health;
}

void Entity::setHealth(int health) {
    this->health = health;
}

int Entity::getDamage() {
    return this->damage;
}

void Entity::setDamage(int damage) {
    this->damage = damage;
}
```

5.25 Entity.h

```
#ifndef ENTITY_H
#define ENTITY_H

#include "Type.h"
```

```

class Alliance;

class Entity {

private:
    Type* type;
    Alliance* alliance;
    int health;
    int damage;

public:
    Entity();

    Entity(Type* type, int health, int damage);

    Type* getType();

    void setType(Type* type);

    Alliance* getAlliance();

    void setAlliance(Alliance* alliance);

    int getHealth();

    void setHealth(int health);

    int getDamage();

    void setDamage(int damage);

    virtual void takeDamage(int damage) = 0;

    virtual void dealDamage(Entity* entity) = 0;

    virtual Entity* clone() = 0;
};

#endif
```

5.26 Factory.cpp

```
#include "Factory.h"

Factory::Factory(Type* type, AddOn* addOn) {
    this->type = type;
    this->addOn = addOn;
}

Factory::~Factory() {
    delete type;
    delete addOn;
}

Type* Factory::getType() {
    return this->type;
}

void Factory::setType(Type* type) {
    this->type = type;
}

AddOn* Factory::getAddOn() {
    return this->addOn;
}

void Factory::setAddOns(AddOn* addOn) {
    this->addOn = addOn;
}
```

5.27 Factory.h

```
#ifndef FACTORY_H
#define FACTORY_H

#include "Type.h"
#include "AddOn.h"

```

```
class Factory {

private:
    Type* type;
    AddOn* addOn;

public:
    Factory(Type* type, AddOn* addOn);

    ~Factory();

    virtual Entity* createEntity(Alliance* alliance) = 0;

    Type* getType();


    void setType(Type* type);


    AddOn* getAddOn();

    void setAddOns(AddOn* addOn);

    virtual Factory* clone() = 0;
};

#endif
```

5.28 General.cpp

```
#include "General.h"

General::General(Alliance* alliance, Strategy* strategy) {
    this->alliance = alliance;
    this->strategy = strategy;
    numDeaths = 0;
}

void General::initiateStrategy(KeyPoint* keyPoint) {
    numDeaths++;
    if (numDeaths >= 5) {
        strategy->performStrat(keyPoint, this->alliance);
        numDeaths = 0;
    }
}

bool General::setStrategy(Strategy* strategy){
    this->strategy = strategy;
    return true;
}

Alliance* General::getAlliance(){
    return this->alliance;
}
```

5.29 General.h

```
#ifndef GENERAL_H
#define GENERAL_H
#include "Alliance.h"
#include "Strategy.h"

class KeyPoint;

class General {

private:
    Alliance* alliance;
    Strategy* strategy;
    int numDeaths;

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

    void initiateStrategy(KeyPoint* keyPoint);

    bool setStrategy(Strategy* strategy);

    Alliance* getAlliance();
};

#endif
```

5.30 KeyPoint.cpp

```
#include "KeyPoint.h"
#include "Weather.h"
#include "RoundStats.h"
#include "Sunny.h"
#include <time.h>
#include <cstdlib>
#include <iostream>

using namespace std;

KeyPoint::KeyPoint(string areaName):  Area(areaName) {
    weather = new Sunny();
}

KeyPoint::KeyPoint(KeyPoint& keyPoint):  Area(keyPoint.getAreaName()) {
    for (int i = 0; i < keyPoint.entities.size(); i++)
        this->addEntity(keyPoint.entities[i]->clone());

    weather = keyPoint.weather->clone();
}

KeyPoint::~KeyPoint() {
    for (int i = 0; i < entities.size(); i++)
        delete entities[i];

    for (int i = 0; i < generals.size(); i++)
        delete generals[i];

    delete weather;
}

bool KeyPoint::isKeyPoint() {
    return true;
}

void KeyPoint::simulateBattle(Alliance* alliance) {
    int numUnits = 0;
    for (int i = 0; i < entities.size(); i++) {
        if (entities[i]->getAlliance() == alliance) {
            numUnits++;
        }
    }

    if (numUnits != entities.size()) {
        for (int i = 0; i < entities.size(); i++) {
            if (entities[i]->getAlliance() == alliance) {
                int random;
                do {
                    random = rand() % entities.size();
                } while (entities[random]->getAlliance() == alliance);

                if (rand() % (int)(weather->getMultiplier() * 100) <= (int)(weather->getMultiplier() *
100))
                    entities[i]->dealDamage(entities[random]);
            }
        }
    }

    clearBattlefield(alliance);
}

void KeyPoint::clearBattlefield(Alliance* alliance) {
    int destroyed = 0;
    double numUnits = 0;
    for (vector<Entity*>::iterator it = entities.begin();  it != entities.end(); ++it) {
        if ((*it)->getHealth() <= 0) {
            destroyed++;
            for (int i = 0; i < generals.size(); i++) {
                if (generals[i]->getAlliance() == (*it)->getAlliance()) {
                    generals[i]->initiateStrategy(this);
                    delete *it;
                    entities.erase(it);
                }
            }
        } else if ((*it)->getAlliance() == alliance) {
            numUnits++;
        }
```

```cpp
    }

    // saving stats
    string stats = getAreaName() + ":\n";
    stats += "Key Point Satus:  ";
    if (numUnits / entities.size() >= 0.6) {
        stats += "Winning\n";
    } else if (numUnits / entities.size() >= 0.35) {
        stats += "Contested\n";
    } else {
        stats += "Losing\n";
    }
    stats += "Number of Entities Destroyed by Alliance:  " + to_string(destroyed);

    RoundStats::keyPointInformation.push_back(stats);
    RoundStats::numEntitiesDestroyed += destroyed;
}

void KeyPoint::moveEntitiesInto(Alliance* alliance, int numTroops) {
    vector<Entity*> troops = alliance->getReserveEntities(numTroops);
    for (int i = 0; i < troops.size(); i++)
        entities.push_back(troops[i]);
}

void KeyPoint::moveEntitiesOutOf(Alliance* alliance, int numTroops) {
    vector<Entity*>::iterator it = entities.begin();
    for (int i = 0; i < numTroops && it != entities.end(); i++) {
        for (; it != entities.end(); ++it) {
            if ((*it)->getAlliance() == alliance) {
                alliance->addReserveEntity(*it);
                entities.erase(it);
            }
        }
    }
}

void KeyPoint::addEntity(Entity* entity) {
    entities.push_back(entity);
}

void KeyPoint::addGeneral(General* general) {
    generals.push_back(general);
}

void KeyPoint::removeGeneral(General* general) {
    for (vector<General*>::iterator it = generals.begin();  it != generals.end(); ++it) {
        if (*it == general) {
            delete *it;
            generals.erase(it);
            return;
        }
    }
}

Area* KeyPoint::clone() {
    return new KeyPoint(*this);
}

void KeyPoint::setWeather(Weather* weather) {
    delete this->weather;
    this->weather = weather;
}

void KeyPoint::changeWeather() {

    srand(time(0));

    int randomNum = 1 + (rand() % 10);
    std::string currWeather = this->weather->getWeather();

    if (currWeather == "Sunny" && randomNum > 6) // 60% chance of not changing weather from Sunny and staying
        this->weather->handleChange(this);
    else if (currWeather == "Cloudy" && randomNum > 3) // 30% chance of not changing weather from Cloudy and staying
        this->weather->handleChange(this);
    else if (currWeather == "Rainy" && randomNum > 1) // 10% chance of not changing weather from Rainy and staying
        this->weather->handleChange(this);


}

std::string KeyPoint::getWeather()const {
    return this->weather->getWeather();
}
```

5.31 KeyPoint.h

```
#ifndef KEYPOINT_H
#define KEYPOINT_H

#include "Alliance.h"
#include "Area.h"
#include "Entity.h"
#include "General.h"
#include <vector>

class Weather;

class KeyPoint :  public Area {

private:
    vector<Entity*> entities;
    vector<General*> generals;
    Weather* weather;
    std::string AreaType;

public:
    KeyPoint(std::string areaName);

    KeyPoint(KeyPoint& keyPoint);

    ~KeyPoint();

    bool isKeyPoint();

    void simulateBattle(Alliance* alliance);

    void clearBattlefield(Alliance* alliance);

    void moveEntitiesInto(Alliance* alliance, int numTroops);

    void moveEntitiesOutOf(Alliance* alliance, int numTroops);

    void addEntity(Entity* entity);

    void addGeneral(General* general);

    void removeGeneral(General* general);

    Area* clone();

    void changeWeather();

  void setWeather(Weather* weather);

    std::string getWeather() const;

};

#endif
```

5.32 Negotiator.cpp

```
#include "Negotiator.h"
#include<bits/stdc++.h>

Negotiator::Negotiator() {}

Negotiator::~Negotiator() {
    alliances.clear();
}

bool Negotiator::sendPeace(Alliance* offerAlliance) {

    for (int yy = 0; yy < alliances.size(); yy++)
    {
        if (alliances[yy] != offerAlliance) {
            if (alliances[yy]->considerPeace() == false)
                return false; // There is at least one enemy alliances that does not want the peace
deal
        }

    }

    return true; // All the alliances being fought against agreed to the peace deal
}

void Negotiator::removeAlliance(Alliance* oldAlliance) {

    for (int xx = 0; xx < alliances.size(); xx++)
    {
        if (alliances[xx]->getID() == oldAlliance->getID())
            alliances.erase( alliances.begin() + xx ); // Removes the specific alliances from this
negotiator
    }

}

void Negotiator::addAlliance(Alliance* newAlliance) {

    if (std::find(alliances.begin(), alliances.end(), newAlliance) != alliances.end())
        alliances.push_back(newAlliance);

}

int Negotiator::getNumAlliances() {
    return this->alliances.size();
}
```

5.33 Negotiator.h

```
#ifndef NEGOTIATOR_H
#define NEGOTIATOR_H
#include <vector>
#include "Alliance.h"

class Negotiator {

private:
    vector<Alliance*> alliances;

public:
    Negotiator();

    ~Negotiator();

    bool sendPeace(Alliance* offerAlliance);

    void removeAlliance(Alliance* oldAlliance);

    void addAlliance(Alliance* newAlliance);

    int getNumAlliances();
};

#endif
```

5.34 NegotiatorTest.h

```
#include <stdexcept>
#include "Negotiator.h"
#include "Alliance.h"
#include "gtest/gtest.h"

namespace {

    // Tests Negotiator Functionality

    // Tests Negotiator offerPeace()
    // ============ Positive Testing ============
    // Test Preconditions Bounds
    TEST(NegotiatorOfferPeace, PositiveTesting) {
        Alliance* a = new Alliance();
        Alliance* b = new Alliance();
        Negotiator* n = new Negotiator();
        n->addAlliance(a);
        n->addAlliance(b);
        a->setNegotiator(n);
        b->setNegotiator(n);

        if (a->offerPeace()) {
            EXPECT_EQ(3, a->getActive());
        } else {
            EXPECT_EQ(1, a->getActive());
        }
    }

    // Tests Negotiator surrender()
    // ============ Positive Testing ============
    // Test Preconditions Bounds
    TEST(NegotiatorSurrender, PositiveTesting) {
        Alliance* a = new Alliance();
        Alliance* b = new Alliance();
        Alliance* c = new Alliance();
        Alliance* d = new Alliance();
        Alliance* e = new Alliance();
        Negotiator* n = new Negotiator();
        n->addAlliance(a);
        n->addAlliance(b);
        n->addAlliance(c);
        n->addAlliance(d);
        n->addAlliance(e);
        a->setNegotiator(n);
        b->setNegotiator(n);
        c->setNegotiator(n);
        d->setNegotiator(n);
        e->setNegotiator(n);

        a->surrender();
        EXPECT_EQ(2, a->getActive());

        b->surrender();
        EXPECT_EQ(2, a->getActive());

        c->surrender();
        EXPECT_EQ(2, a->getActive());

        d->surrender();
        EXPECT_EQ(2, a->getActive());
    }

}
```

5.35 Passive.cpp

```
#include "Passive.h"

using namespace std;

Passive::Passive() {}

void Passive::performStrat(KeyPoint* keyPoint, Alliance* alliance) {

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

Strategy* Passive::clone() {
    return new Passive();
}
```

5.36 Passive.h

```
#ifndef PASSIVE_H
#define PASSIVE_H
#include "Strategy.h"
#include "KeyPoint.h"

class Passive :  public Strategy {

public:
    Passive();

    void performStrat(KeyPoint* keyPoint, Alliance* alliance);

    Strategy* clone();
};

#endif
```

5.37 Personnel.cpp

```
#include "Personnel.h"
```

```
#include "RoundStats.h"
#include <iostream>

Personnel::Personnel(Type* type, int health, int damage):  Entity(type, health, damage) {}

void Personnel::takeDamage(int damage) {
    setHealth(getHealth() - damage);
}

void Personnel::dealDamage(Entity* entity) {
    RoundStats::damageDone += getDamage();
    entity->takeDamage(getDamage());
}

Entity* Personnel::clone() {
    Personnel* p;
    if (this->getType() == NULL) {
        p = new Personnel(NULL, this->getHealth(), this->getDamage());
    } else {
        p = new Personnel(this->getType()->clone(), this->getHealth(), this->getDamage());
    }

    p->setAlliance(this->getAlliance());

    return p;
}
```

5.38 Personnel.h

```
#ifndef PERSONNEL_H
#define PERSONNEL_H

#include "Entity.h"

class Personnel :  public Entity {

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

    void takeDamage(int damage);

    void dealDamage(Entity* entity);

    Entity* clone();
};

#endif
```

5.39 PersonnelFactory.cpp

```
#include "PersonnelFactory.h"
#include "Personnel.h"
#include <iostream>

PersonnelFactory::PersonnelFactory(Type* type, AddOn* addOn):  Factory(type, addOn) {}

Entity* PersonnelFactory::createEntity(Alliance* alliance) {
    Personnel* p = new Personnel(getType()->clone());
    p->setAlliance(alliance);
    if (getAddOn() != NULL) {
        AddOn* personnelAddOn = getAddOn()->clone();
        personnelAddOn->setEntity(p);
        return personnelAddOn;
    } else {
        return p;
    }
}

Factory* PersonnelFactory::clone() {
    return new PersonnelFactory(getType()->clone(), getAddOn()->clone());
}
```

5.40 PersonnelFactory.h

```
#ifndef PERSONNELFACTORY_H
```

```
#define PERSONNELFACTORY_H

#include "Factory.h"

class PersonnelFactory :  public Factory {

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

    Entity* createEntity(Alliance* alliance);

    Factory* clone();
};

#endif
```

5.41 Piercing.cpp

```
#include "Piercing.h"
#include "RoundStats.h"

Piercing::Piercing(int value) :  AddOn(value) {}

void Piercing::takeDamage(int damage) {
    entity->takeDamage(damage);
}

void Piercing::dealDamage(Entity* entity) {
    int sumValue = this->entity->getDamage() + value;
    entity->takeDamage(sumValue);
    RoundStats::damageDone += sumValue;
}

AddOn* Piercing::clone() {
    Piercing* piercing = new Piercing(value);
    if (getEntity() != NULL)
        piercing->setEntity(entity->clone());
    return piercing;
}
```

5.42 Piercing.h

```
#ifndef PIERCING_H
#define PIERCING_H
#include "AddOn.h"
#include "Entity.h"

class Piercing :  public AddOn {


public:
    Piercing(int value);

    void takeDamage(int damage);

    void dealDamage(Entity* entity);

    AddOn* clone();
};

#endif
```

5.43 Rainy.cpp

```
#include "Rainy.h"
#include "Sunny.h"

Rainy::Rainy():  Weather() {
    this->multiplier = 0.5;
}

std::string Rainy::getWeather() {
    return "Rainy";
}

```

```
void Rainy::handleChange(KeyPoint* k) {
    Sunny* newWeather = new Sunny();
    k->setWeather(newWeather);
}

Weather* Rainy::clone() {
    return new Rainy();
}
```

5.44 Rainy.h

```
#ifndef RAINY_H
#define RAINY_H
#include "Weather.h"
#include "KeyPoint.h"

class Rainy :  public Weather {


public:
    Rainy();

    std::string getWeather();

    void handleChange(KeyPoint* k);

    Weather* clone();
};

#endif
```

5.45 RoundStats.cpp

```
#include "RoundStats.h"

int RoundStats::numEntitiesCreated = 0;
int RoundStats::numEntitiesDestroyed = 0;
int RoundStats::damageDone = 0;
vector<string> RoundStats::keyPointInformation;
vector<string> RoundStats::entityMovementInformation;

void RoundStats::clearStats() {
    numEntitiesCreated = 0;
    numEntitiesDestroyed = 0;
    keyPointInformation.clear();
    entityMovementInformation.clear();
}

string RoundStats::toString() {
    string out = "Number of Entities Created:  " + to_string(numEntitiesCreated) + "\n";
    out += "Number of Entities Destroyed by Alliance:  " + to_string(numEntitiesDestroyed) + "\n";
    out += "Damage Given by Alliance:  " + to_string(damageDone) + "\n";

    out += "\nKey Point Round Information:\n";
    for (int i = 0; i < keyPointInformation.size(); i++)
        out += keyPointInformation[i] + "\n";

    out += "\nMovement Round Information:\n";
    for (int i = 0; i < entityMovementInformation.size(); i++)
        out += entityMovementInformation[i] + "\n";

    return out;
}
```

5.46 RoundStats.h

```
#ifndef ROUNDSTATS_H
#define ROUNDSTATS_H

#include <vector>
#include <string>

using namespace std;

class RoundStats {
    public:
        static int numEntitiesCreated;
        static int numEntitiesDestroyed;
        static int damageDone;
        static vector<string> keyPointInformation;
        static vector<string> entityMovementInformation;

        static void clearStats();
        static string toString();
};

#endif
```

5.47 SaveArchive.cpp

```
#include "SaveArchive.h"

SaveArchive::SaveArchive() {}

void SaveArchive::addNewSave(std::string newSaveName, WarEngineMemento* newSave) {
    saveList.insert({newSaveName, newSave});
}

WarEngineMemento* SaveArchive::getLastSave() {

    if(saveList.size() == 0){
        throw "Save archive is empty.";
    }

    WarEngineMemento* lastSave = saveList.begin()->second;

    saveList.erase( saveList.begin() );

    return lastSave;
}

WarEngineMemento* SaveArchive::getSave(std::string name) {
    if(saveList.size() == 0){
        std::__throw_out_of_range("Save archive is empty");
    }

    auto iter = saveList.find(name);

    if(iter == saveList.end())
        std::__throw_invalid_argument("No save with given name exists");

    return iter->second;
}

void SaveArchive::clearSaveList() {
    saveList.clear();
}

void SaveArchive::deleteSave(std::string name) {
    if(saveList.size() == 0){
        std::__throw_out_of_range("Save archive is empty");
    }

    auto iter = saveList.find(name) ;

    if(iter == saveList.end())
        return;

    saveList.erase( iter );
}
```

5.48 SaveArchive.h

```
#ifndef SAVEARCHIVE_H
#define SAVEARCHIVE_H
#include <unordered_map>
#include <string>
#include "WarEngineMemento.h"

class SaveArchive {

private:
    std::unordered_map<std::string, WarEngineMemento*> saveList;

public:
    SaveArchive();

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

    WarEngineMemento* getLastSave();

    WarEngineMemento* getSave(std::string name);

    void clearSaveList();

    void deleteSave(std::string name);
};

#endif
```

5.49 Strategy.cpp

```
#include "Strategy.h"
#include "KeyPoint.h"
#include "Alliance.h"

using namespace std;

Strategy::Strategy() {}

Strategy::~Strategy(){}

```

5.50 Strategy.h

```
#ifndef STRATEGY_H
#define STRATEGY_H
#include <string>
#include <ctime>
#include <cstdlib>

class KeyPoint;
class Alliance;

class Strategy {

protected:
    std::string strategy;

public:
    Strategy();

    ~Strategy();

    virtual void performStrat(KeyPoint* keyPoint, Alliance* alliance) = 0;

    virtual Strategy* clone() = 0;
};

#endif
```

5.51 Sunny.cpp

```
#include "Sunny.h"
#include "Cloudy.h"

Sunny::Sunny() {
    this->multiplier = 1.0;
}

std::string Sunny::getWeather() {
    return "Sunny";
}

void Sunny::handleChange(KeyPoint* k) {
    Cloudy* newWeather = new Cloudy();
    k->setWeather(newWeather);
}

Weather* Sunny::clone() {
    return new Sunny();
}
```

5.52 Sunny.h

```
#ifndef SUNNY_H
#define SUNNY_H
#include "Weather.h"
#include "KeyPoint.h"

#include "Weather.h"

class Sunny :  public Weather {

public:
    Sunny();

    virtual std::string getWeather();

    virtual void handleChange(KeyPoint* k);

    Weather* clone();
};

#endif
```

5.53 Support.cpp

```
#include "Support.h"
#include "RoundStats.h"

Support::Support(Type* type, int health, int damage):  Entity(type, health, damage) {}

void Support::dealDamage(Entity* entity) {
    RoundStats::damageDone += getDamage();
    entity->takeDamage(getDamage());
}

void Support::takeDamage(int damage) {
    this->setHealth(this->getHealth() - damage);
}

Entity* Support::clone() {
    Support* s;
    if (this->getType() == NULL) {
        s = new Support(NULL, this->getHealth(), this->getDamage());
    } else {
        s = new Support(this->getType()->clone(), this->getHealth(), this->getDamage());
    }

    s->setAlliance(this->getAlliance());

    return s;
}
```

5.54 Support.h

```
#ifndef SUPPORT_H
#define SUPPORT_H

#include "Entity.h"

class Support :  public Entity {

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

    void takeDamage(int damage);

    void dealDamage(Entity* entity);

    Entity* clone();
};

#endif
```

5.55 SupportFactory.cpp

```
#include "SupportFactory.h"
```

```
#include "Support.h"

SupportFactory::SupportFactory(Type* type, AddOn* addOn):  Factory(type, addOn) {}

Entity* SupportFactory::createEntity(Alliance* alliance) {
    Support* s = new Support(getType()->clone());
    s->setAlliance(alliance);
    if (getAddOn() != NULL) {
        AddOn* personnelAddOn = getAddOn()->clone();
        personnelAddOn->setEntity(s);
        return personnelAddOn;
    } else {
        return s;
    }
}

Factory* SupportFactory::clone() {
    return new SupportFactory(getType()->clone(), getAddOn()->clone());
}
```

5.56 SupportFactory.h

```
#ifndef SUPPORTFACTORY_H
#define SUPPORTFACTORY_H

#include "Factory.h"

class SupportFactory :  public Factory {

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

    Entity* createEntity(Alliance* alliance);

    Factory* clone();
};

#endif
```

5.57 TerrainType.cpp

```
#include "TerrainType.h"

TerrainType::TerrainType() {}

string TerrainType::getTypeDesc() {
    return "Terrain";
}

Type* TerrainType::clone() {
    return new TerrainType();
}
```

5.58 TerrainType.h

```
#ifndef TERRAINTYPE_H
#define TERRAINTYPE_H

#include "Type.h"

class TerrainType :  public Type {

public:
    TerrainType();

    string getTypeDesc();

    Type* clone();
};

#endif
```

5.59 testmain.cpp

```
#include "NegotiatorTest.h"
#include "AddOnTest.h"
#include <gtest/gtest.h>
#include "EasySetup.h"
#include "WarEngine.h"
#include "KeyPoint.h"

void setupWarEngine() {
    Alliance* a1 = new Alliance();
    a1->addCountry(new Country("Germany"));

    Alliance* a2 = new Alliance();
    a2->addCountry(new Country("Finland"));

    WarEngine::getInstance().addAlliance(a1);
    WarEngine::getInstance().addAlliance(a2);

    KeyPoint* k1 = new KeyPoint("West");
    KeyPoint* k2 = new KeyPoint("North");
    KeyPoint* k3 = new KeyPoint("East");

    Personnel* p1 = new Personnel(NULL);
    p1->setAlliance(a1);
    k1->addEntity(p1->clone());
    k2->addEntity(p1->clone());
    k3->addEntity(p1->clone());

    Personnel* p2 = new Personnel(NULL);
    p2->setAlliance(a2);
    k1->addEntity(p2->clone());
    k2->addEntity(p2->clone());

    WarTheatre* w = new WarTheatre("Europe");
    w->addArea(k1);
    w->addArea(k2);
    w->addArea(k3);

    WarEngine::getInstance().setWarTheatre(w);
}

int main(int argc, char **argv) {
    // EasySetup* easySetup = new EasySetup();
    // easySetup->setupSimulation();

    setupWarEngine();
    WarEngine::getInstance().simulate();

    // testing::InitGoogleTest(&argc, argv);
    // return RUN_ALL_TESTS();
}
```

5.60 Type.cpp

```
#include "Type.h"

Type::Type() {}
```

5.61 Type.h

```
#ifndef TYPE_H
#define TYPE_H

#include <string>

using namespace std;

class Type {

public:
    Type();

    virtual string getTypeDesc() = 0;

    virtual Type* clone() = 0;

};

#endif
```

5.62 Vehicle.cpp

```
#include "Vehicle.h"
#include "RoundStats.h"

Vehicle::Vehicle(Type* type, int health, int damage): Entity(type, health, damage) {}

void Vehicle::takeDamage(int damage) {
    setHealth(getHealth() - damage);
}

void Vehicle::dealDamage(Entity* entity) {
    RoundStats::damageDone += getDamage();
    entity->takeDamage(getDamage());
}

Entity* Vehicle::clone() {
    Vehicle* v;
    if (this->getType() == NULL) {
        v = new Vehicle(NULL, this->getHealth(), this->getDamage());
    } else {
        v = new Vehicle(this->getType()->clone(), this->getHealth(), this->getDamage());
    }

    v->setAlliance(this->getAlliance());

    return v;
}
```

5.63 Vehicle.h

```
#ifndef VEHICLE_H
#define VEHICLE_H

#include "Entity.h"

class Vehicle : public Entity {

public:
    Vehicle(Type* type, int health = 500, int damage = 10);

    void takeDamage(int damage);

    void dealDamage(Entity* entity);

    Entity* clone();
};

#endif
```

5.64 VehicleFactory.cpp

```
#include "VehicleFactory.h"
#include "Vehicle.h"

VehicleFactory::VehicleFactory(Type* type, AddOn* addOn): Factory(type, addOn) {}

Entity* VehicleFactory::createEntity(Alliance* alliance) {
    Vehicle* v = new Vehicle(getType()->clone());
    v->setAlliance(alliance);
    if (getAddOn() != NULL) {
        AddOn* personnelAddOn = getAddOn()->clone();
        personnelAddOn->setEntity(v);
        return personnelAddOn;
    } else {
        return v;
    }
}

Factory* VehicleFactory::clone() {
    return new VehicleFactory(getType()->clone(), getAddOn()->clone());
}
```

5.65 VehicleFactory.h

```
#ifndef VEHICLEFACTORY_H
```

```
#define VEHICLEFACTORY_H
#include "Factory.h"

class VehicleFactory :  public Factory {

public:
    VehicleFactory(Type* type, AddOn* addOn);

    Entity* createEntity(Alliance* alliance);

    Factory* clone();
};

#endif
```

5.66 WarEngine.cpp

```
#include "WarEngine.h"
#include "RoundStats.h"
#include <iostream>

WarEngine::WarEngine(){
    this->state = new WarEngineState();
}

WarEngineMemento* WarEngine::saveState() {
    return new WarEngineMemento(state->clone());
}

void WarEngine::loadSave(WarEngineMemento* save) {
    delete this->state;
    this->state = save->getState();
}

WarEngine& WarEngine::getInstance(){
    static WarEngine uniqueInstance_;
    return uniqueInstance_;
}

WarEngine::~WarEngine(){
    delete this->state;
}


void WarEngine::simulate() {
    vector<Alliance*> alliances = this->state->getAlliances();

    for(int i = 0; i < alliances.size(); i++) {
        if (alliances[i]->getActive() == 1) {
            RoundStats::clearStats();
            state->getArea()->simulateBattle(alliances[i]);
            cout « "========================================================================" « endl;
            cout « "Alliance " « alliances[i]->getID() « ":" « endl;

            if (alliances[i]->getID() == 2) {
                cout « "Status:  Surrendered" « endl;
            } else if (alliances[i]->getID() == 3) {
                cout « "Status:  Found Peace" « endl;
            } else {
                cout « "Status:  Active" « endl;
            }

            cout « RoundStats::toString() « endl;
            cout « "========================================================================" « endl;
        }
    }

}

void WarEngine::setWarTheatre(WarTheatre* battleGround){
    state->setArea(battleGround);
}

void WarEngine::addAlliance(Alliance* alliance) {
    state->alliances.push_back(alliance);
}
```

5.67 WarEngine.h

```
#ifndef WARENGINE_H
```

```
#define WARENGINE_H

#include "WarEngineState.h"
#include "WarEngineMemento.h"
#include "WarTheatre.h"

class WarEngine {

private:
    WarEngineState* state;
    bool gameOver;

protected:
    WarEngine();

    WarEngine(const WarEngine&){};

    WarEngine& operator=(const WarEngine&){ return *this; };

    ~WarEngine();

public:
    WarEngineMemento* saveState();

    void loadSave(WarEngineMemento* save);

    static WarEngine& getInstance();

    void setWarTheatre(WarTheatre* battleGround);

    void addAlliance(Alliance* alliance);

    void simulate();

};

#endif
```

5.68 WarEngineMemento.cpp

```
#include "WarEngineMemento.h"

WarEngineMemento::WarEngineMemento(WarEngineState * state){
    this->state = state;
}

void WarEngineMemento::setState(WarEngineState* state){
    this->state = state;
}

WarEngineState* WarEngineMemento::getState(){
    return state;
}
```

5.69 WarEngineMemento.h

```
#ifndef WARENGINEMEMENTO_H
#define WARENGINEMEMENTO_H

#include "WarEngineState.h"
#include <string>
#include <vector>

class WarEngine;

class WarEngineMemento {

friend class WarEngine;

private:
    WarEngineState* state;

    WarEngineMemento(WarEngineState* state);

    void setState(WarEngineState* state);

    WarEngineState* getState();

};

#endif
```

5.70 WarEngineState.cpp

```
#include "WarEngineState.h"
#include "Negotiator.h"
#include "Defensive.h"

WarEngineState::WarEngineState() {
    area = nullptr;
}

void WarEngineState::setArea(Area* area) {
    this->area = area;
}

Area* WarEngineState::getArea() {

    if(area == nullptr)
        throw "No Areas Stored.";

    return this->area;
}

void WarEngineState::setAlliances(vector<Alliance*> alliances) {
    this->alliances = alliances;
}

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

    if(alliances.size() == 0)
        std::__throw_out_of_range("No Alliances stored.");

    return alliances;
}

WarEngineState* WarEngineState::clone(){

    WarEngineState* clonedState = new WarEngineState();
    Area* copiedArea = this->area->clone();

    Negotiator* negotiator = new Negotiator();
    for(Alliance* alliance :  this->alliances){

        Alliance* clonedAlliance = alliance->clone();

        clonedState->alliances.push_back(clonedAlliance);
        clonedAlliance->setNegotiator(negotiator);
        negotiator->addAlliance(clonedAlliance);
        copiedArea->addGeneral(new General(clonedAlliance, new Defensive()));
    }

    clonedState->setArea(copiedArea);

    return clonedState;
}

WarEngineState::~WarEngineState(){

    for(Alliance* alliance :  this->alliances){
        delete alliance;
    }

    //delete this->area;

}
```

5.71 WarEngineState.h

```
#ifndef WARENGINESTATE_H
#define WARENGINESTATE_H
#include "Alliance.h"
#include "Area.h"
#include <vector>

class WarEngine;

using namespace std;

class WarEngineState {

friend class WarEngine;

private:
    Area* area;
    vector<Alliance*> alliances;

    WarEngineState();

    void setArea(Area* area);

    Area* getArea();

    void setAlliances(vector<Alliance*> alliances);

    vector<Alliance*> getAlliances();

    WarEngineState* clone();

    ~WarEngineState();
};

#endif
```

5.72 WarTheatre.cpp

```
#include "WarTheatre.h"

using namespace std;

WarTheatre::WarTheatre(string areaName):  Area(areaName) {}

WarTheatre::~WarTheatre() {
    for (int i = 0; i < areas.size(); i++)
        delete areas[i];
}

bool WarTheatre::isKeyPoint() {
    return false;
}

void WarTheatre::simulateBattle(Alliance* alliance) {
    for (int i = 0; i < areas.size(); i++)
        areas[i]->simulateBattle(alliance);
}

void WarTheatre::addArea(Area* area) {
    areas.push_back(area);
}

WarTheatre* WarTheatre::clone() {
    WarTheatre* w = new WarTheatre(getAreaName());

    for (int i = 0; i < areas.size(); i++)
        w->addArea(areas[i]->clone());

    return w;
}

void WarTheatre::addGeneral(General* general) {
    for (int i = 0; i < areas.size(); i++)
        areas[i]->addGeneral(general);
}
```

5.73 WarTheatre.h

```
#ifndef WARTHEATRE_H
#define WARTHEATRE_H

#include "Area.h"
#include "Alliance.h"
#include <vector>

using namespace std;

class WarTheatre :  public Area {

private:
    vector<Area*> areas;

public:
    WarTheatre(std::string areaName);

    ~WarTheatre();

    bool isKeyPoint();

    void simulateBattle(Alliance* alliance);

    void addArea(Area* area);

    void addGeneral(General* general);

    WarTheatre* clone();
};

#endif
```

5.74 Weather.cpp

```
#include "Weather.h"

Weather::Weather() {}

Weather::~Weather() {}

double Weather::getMultiplier() {
    return this->multiplier;
}
```

5.75 Weather.h

```
#ifndef WEATHER_H
#define WEATHER_H
#include <string>
#include "Weather.h"
#include "KeyPoint.h"

class KeyPoint;

class Weather {

protected:
    double multiplier;

public:
    Weather();

    ~Weather();

    double getMultiplier();

    virtual void handleChange(KeyPoint* k) = 0;

    virtual std::string getWeather() = 0;

    virtual Weather* clone() = 0;
};

#endif
```

Index