COS214PROJECT

1st

Generated by Doxygen 1.9.5

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

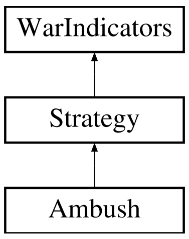
Class Documentation

4.1 Ambush Class Reference

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

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

Inheritance diagram for Ambush:



Public Member Functions

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

  constructor. initializes stealth and calls base class constructor to initialize myKingdom, enemyKingdom, my↩ Bannerman, enemyBannerman, strategy, min and minFavour.
- bool attack (Bannerman ∗myBannerman, Bannerman ∗enemyBannerman)

  makes two bannerman from enemy kingdoms fight each other until one loses
- ∼**Ambush** ()

  destructor.

Private Attributes

- int stealth

Additional Inherited Members

4.1.1 Detailed Description

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

Author

Morgan Bentley

Date

October 2022

4.1.2 Constructor & Destructor Documentation

4.1.2.1 Ambush()

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

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

Parameters

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

4.1.3 Member Function Documentation

4.1.3.1 attack()

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

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

Parameters

myBannerman	- attacking bannerman object.
enemyBannerman	- Bannerman object being attacked.

Returns

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

Implements Strategy.

4.1.4 Member Data Documentation

4.1.4.1 stealth

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

skill level of Bannerman using this strategy

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

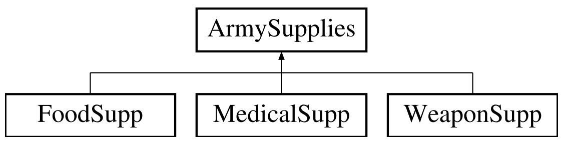
- Ambush.h
- Ambush.cpp

4.2 ArmySupplies Class Reference

Abstract Class containing the information on the supplies.

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

Inheritance diagram for ArmySupplies:



Public Member Functions

- virtual int getAmount ()=0

    getAmount() is used to return the amount of supplies.

4.2.1 Detailed Description

Abstract Class containing the information on the supplies.

Author

Ronin Brookes 19069686

4.2.2 Member Function Documentation

4.2.2.1 getAmount()

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

getAmount() is used to return the amount of supplies.

Returns

an integer representation of the amount of supplies.

Implemented in FoodSupp, MedicalSupp, and WeaponSupp.

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

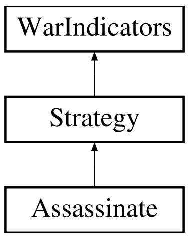
- ArmySupplies.h

4.3 Assassinate Class Reference

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

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

Inheritance diagram for Assassinate:



Public Member Functions

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

  constructor. initializes stealth and calls base class constructor to initialize myKingdom, enemyKingdom, my↩ Bannerman, enemyBannerman, strategy, min and minFavour.
- bool attack (Bannerman ∗myBannerman, Bannerman ∗enemyBannerman)

  makes assassin of attacking Bannerman try to kill enemyBannerman
- ∼**Assassinate** ()

  destructor.

Private Attributes

- int stealth
- bool alive

Additional Inherited Members

4.3.1 Detailed Description

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

Author

Morgan Bentley

Date

October 2022

4.3.2 Constructor & Destructor Documentation

4.3.2.1 Assassinate()

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

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

Parameters

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

4.3.3 Member Function Documentation

4.3.3.1 attack()

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

makes assassin of attacking Bannerman try to kill enemyBannerman

Parameters

myBannerman	- attacking Bannerman object.
enemyBannerman	- Bannerman object being attacked.

Returns

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

Implements Strategy.

4.3.4 Member Data Documentation

4.3.4.1 alive

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

life state of Bannerman's assassin, is false if assassin was killed

4.3.4.2 stealth

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

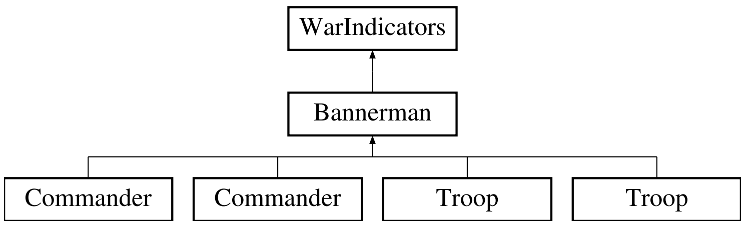
skill level of Assassin from Bannerman using this strategy

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

- Assassinate.h
- Assassinate.cpp

4.4 Bannerman Class Reference

Inheritance diagram for Bannerman:



Public Member Functions

- **Bannerman** ()

  Default constructor.
- virtual void increaseFavour ()=0

  Abstract. increases the favour of the bannerman.
- virtual void decreaseFavour ()=0

  Abstract. decreases the favour of the bannerman.
- virtual void attach (Raven ∗o)=0

  Abstract. Attaches a Raven Observer on the bannerman.
- virtual void detach (Raven ∗o)=0

  Abstract. detaches a Raven Observer from the bannerman.
- virtual void increaseHP (int boost)=0

  Abstract. increases the health points(HP) of the bannerman.
- virtual void changeStrategy (Strategy ∗strategy)=0

  Abstract. changes the attack strategy of the bannerman.
- virtual void attack (Bannerman ∗myBannerman, Bannerman ∗enemyBannerman)=0

  Abstract. uses strategy to attack another kingdom.
- virtual void increasePower (int boost)=0

  Abstract. increases the damage capability of the bannerman.
- virtual string getName ()=0

  Accessor that returns the name of the bannerman.
- virtual int getHP ()=0

  Abstract. Accessor that returns the name of the component.
- virtual int getDamage ()=0

Abstract. Accessor that returns the damage capability of the bannerman.
- virtual void receiveDamage (int boost)=0

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

  Abstract. decreases the damage capability of the bannerman.
- virtual void decreaseWeapons ()=0

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

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

  Abstract. Decreases the number of medical supplies that the bannerman has.
- virtual int getWeapons ()=0

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

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

  Abstract. Accessor that returns the number of medical supplies that the bannerman has.
- virtual void setWeapons (int numWeapons)=0

  Abstract. Sets the number of weapons of the bannerman.
- virtual void setFood (int numFood)=0

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

  Abstract. Sets the number of medical supplies that the bannerman has.
- virtual void setRaven (list< Raven ∗ > r)=0

  Abstract. Assigns a list of Raven Observers to the bannerman's ravenList.
- virtual void setMaster (MasterOfCoin ∗m)=0

  Abstract. Assigns a MasterofCoin mediator to ensure that the army has the supplies it needs.
- virtual int getFavour ()=0

  Abstract. returns the favour of the bannerman.
- ∼**Bannerman** ()

  Default destructor.
- virtual list< Bannerman ∗ > getTroops ()=0

  Abstract. gets the list of troops under the commander.

Public Attributes

- bool assassin

Protected Attributes

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

4.4.1 Member Function Documentation

4.4.1.1 attach()

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

Abstract. Attaches a Raven Observer on the bannerman.

Parameters

o	- the Raven Observer to attach.

Implemented in Commander, and Troop.

4.4.1.2 attack()

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

Abstract. uses strategy to attack another kingdom.

Parameters

myBannerman	- The attacking bannerman object.
enemyBannerman	- The bannerman object being attacked.

Implemented in Commander, and Troop.

4.4.1.3 changeStrategy()

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

Abstract. changes the attack strategy of the bannerman.

Parameters

strategy	- the new attack strategy.

Implemented in Commander, and Troop.

4.4.1.4 decreaseFavour()

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

Abstract. decreases the favour of the bannerman.

Implemented in Commander, and Troop.

4.4.1.5 decreaseFood()

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

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

Implemented in Commander, and Troop.

4.4.1.6 decreaseMedical()

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

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

Implemented in Commander, and Troop.

4.4.1.7 decreasePower()

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

Abstract. decreases the damage capability of the bannerman.

Parameters

x	- the number by which to decrease damage.

Implemented in Commander, and Troop.

4.4.1.8 decreaseWeapons()

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

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

Implemented in Commander, and Troop.

4.4.1.9 detach()

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

Abstract. detaches a Raven Observer from the bannerman.

Parameters

o	- the Raven Observer to detach.

Implemented in Commander, and Troop.

4.4.1.10 getDamage()

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

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

Returns

The damage of the bannerman.

Implemented in Commander, and Troop.

4.4.1.11 getFavour()

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

Abstract. returns the favour of the bannerman.

Implemented in Commander, and Troop.

4.4.1.12 getFood()

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

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

Returns

The numFood of the bannerman.

Implemented in Commander, and Troop.

4.4.1.13 getHP()

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

Abstract. Accessor that returns the name of the component.

Returns

The name of the bannerman.

Implemented in Commander, and Troop.

4.4.1.14 getMedical()

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

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

Returns

The numMedical of the bannerman.

Implemented in Commander, and Troop.

4.4.1.15 getName()

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

Accessor that returns the name of the bannerman.

Returns

The name of the bannerman.

Implemented in Commander, and Troop.

4.4.1.16 getTroops()

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

Abstract. gets the list of troops under the commander.

Returns

the list of troops variable

Implemented in Commander, and Troop.

4.4.1.17 getWeapons()

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

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

Returns

The numWeapons of the bannerman.

Implemented in Commander, and Troop.

4.4.1.18 increaseFavour()

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

Abstract. increases the favour of the bannerman.

Implemented in Commander, and Troop.

4.4.1.19 increaseHP()

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

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

Parameters

boost	- the number by which to increase HP.

Implemented in Commander, and Troop.

4.4.1.20 increasePower()

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

Abstract. increases the damage capability of the bannerman.

Parameters

boost	- the number by which to increase damage.

Implemented in Commander, and Troop.

4.4.1.21 receiveDamage()

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

Abstract. Increases the damage capability of the bannerman.

Parameters

boost	- the number by which to increase damage

Implemented in Commander, and Troop.

4.4.1.22 setFood()

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

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

Parameters

numFood	- The new numFood the component should have

Implemented in Commander, and Troop.

4.4.1.23 setMaster()

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

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

Parameters

m	- the new MasterOfCoin mediator.

Implemented in Commander, and Troop.

4.4.1.24 setMedical()

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

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

Parameters

numMedical	- The new numMedical the component should have

Implemented in Commander, and Troop.

4.4.1.25 setRaven()

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

Abstract. Assigns a list of Raven Observers to the bannerman's ravenList.

Parameters

r	- the list of Raven Observer to attach.

Implemented in Commander, and Troop.

4.4.1.26 setWeapons()

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

Abstract. Sets the number of weapons of the bannerman.

Parameters

numWeapons	- The new name the numWeapons should have

Implemented in Commander, and Troop.

4.4.2 Member Data Documentation

4.4.2.1 assassin

```
bool Bannerman::assassin
```

Indicates whether or not the bannerman is an assassin

4.4.2.2 damage

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

Damage capability the bannerman has

4.4.2.3 favour

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

Amount of favour the bannerman has

4.4.2.4 HP

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

The health points of the bannerman

4.4.2.5 m

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

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

4.4.2.6 name

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

Name of the bannerman

4.4.2.7 numFood

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

Amount of food the bannerman has

4.4.2.8 numMedical

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

Amount of medical supplies the bannerman has

4.4.2.9 numWeapons

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

Number of weapons the bannerman has

4.4.2.10 ravenList

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

A list of Raven observers that have been attached to the bannerman object

4.4.2.11 strategy

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

The attack strategy the bannerman uses

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

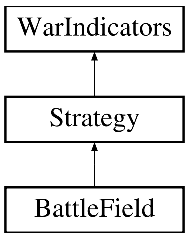
- Bannerman.h
- Bannerman.cpp

4.5 BattleField Class Reference

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

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

Inheritance diagram for BattleField:



Public Member Functions

- BattleField (Kingdom ∗myKingdom, Kingdom ∗enemyKingdom, Bannerman ∗myBannerman, Bannerman ∗enemyBannerman, string name, int min, int minFavour, Historian ∗h, HistoryBook ∗hb)

  constructor. initializes stealth and calls base class constructor to initialize myKingdom, enemyKingdom, my←Bannerman, enemyBannerman, strategy, min and minFavour.
- bool attack (Bannerman ∗myBannerman, Bannerman ∗enemyBannerman)

  makes two bannerman from enemy kingdoms fight each other until one loses
- ∼**BattleField** ()

  destructor.

Additional Inherited Members

4.5.1 Detailed Description

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

Author

Morgan Bentley

Date

October 2022

4.5.2 Constructor & Destructor Documentation

4.5.2.1 BattleField()

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

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

Parameters

myKingdom	- Kingdom pointer of attacking Bannerman's Kingdom.
enemyKingdom	- Kingdom pointer of enemyBannerman's Kingdom.
myBannerman	- attacking Bannerman.
enemyBannerman	- defending Bannerman.
name	- name of concrete strategy.
min	- minimum supplies for food,weapons and medicine

4.5.3 Member Function Documentation

4.5.3.1 attack()

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

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

Parameters

myBannerman	- attacking bannerman object.
enemyBannerman	- Bannerman object being attacked.

Returns

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

Implements Strategy.

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

- BattleField.h
- BattleField.cpp

4.6 Class Class Reference

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

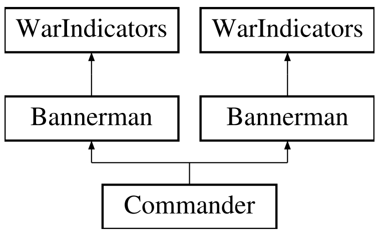
- Class.h

4.7 Commander Class Reference

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

`#include <Commander.h>`

Inheritance diagram for Commander:



Public Member Functions

- Commander (string name)

  Commander Constructor which takes in the name of a commander.
- Iterator ∗ **createIterator** ()

  creates a conIterator object, which is a means to traverse groundForces sequentially.
- void **removeBannerman** (Bannerman ∗x)

  removes a bannerman object from the groundforces list.
- void attack (Bannerman ∗myBannerman, Bannerman ∗enemyBannerman) override

  uses strategy to make all troops in groundForces to attack another kingdom.
- void **addBannerman** (Bannerman ∗b)

  adds a bannerman object to the groundforces list.
- list< Bannerman ∗ > getTroops ()
- int getHP () override

  Accessor. Returns the HP.
- int getDamage () override

  Accessor. Returns the damage.
- void attach (Raven ∗o) override

  Attaches a Raven observer to all the bannerman objects in the groundForces list.
- void detach (Raven ∗o) override

  Detaches a Raven observer from all the bannerman objects in the groundForces list.
- void receiveDamage (int x) override

  increases the damage variables of the groundForces by x
- void decreaseWeapons () override

  decreases the numWeapons variables of the bannerman objects in groundForces by 1.
- void decreaseFood () override

  decreases the numFood variables of the bannerman objects in groundForces by 1.
- void decreaseMedical () override

  decreases the numMedical variables of the bannerman objects in groundForces by 1.
- void increasePower (int boost) override

  increases the damage variables of the bannerman objects in groundForces.
- int getWeapons () override

  Accessor. Returns numWeapons.
- int getFood () override

  Accessor. Returns the total groundForces amount of food.
- int getMedical () override

  Accessor. Returns the total groundForces amount of medical supplies.
- void increaseHP (int boost) override

  increases the HP variables of the bannerman objects in groundForces.
- void changeStrategy (Strategy ∗strategy) override

  Changes the attack strategy variable of the bannerman objects in groundForces.
- void increaseFavour () override

  Increases the loyalty favour levels variable of the bannerman objects in groundForces by 1.
- void decreaseFavour () override

  Decreases the loyalty favour levels variable of the bannerman objects in groundForces by 1.
- int getFavour () override

  Returns the sum of the loyalty favour levels variable of the bannerman objects in groundForce.
- void setWeapons (int numWeapons) override

  Sets the number of weapon supplies of the bannerman objects in groundForces.
- void setFood (int numFood) override

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

- void setMedical (int numMedical) override

  Sets the number of medical supplies of the bannerman objects in groundForces.
- void decreasePower (int x) override

  Decreases the damage capability of all the bannerman objects in groundForces.
- ∼**Commander** ()

  Default destructor.
- void setRaven (list< Raven ∗ > r) override

  Assigns a Raven observer list to all the bannerman objects in the groundForces list.
- void setMaster (MasterOfCoin ∗m) override

  Assigns a MasterofCoin mediator all the bannerman objects in the groundForces list to ensure that the army has the supplies it needs.
- void setStrategy (Strategy ∗s)

  Sets the attack strategy variable of the bannerman objects in groundForces.
- string getName ()

  Returns the name of this commander object.

Private Attributes

- list< Bannerman ∗ > **groundForces**

  list of bannerman components.
- string **name**

  name of the bannerman as a whole, represented by the commander speaking for the bannerman

Additional Inherited Members

4.7.1 Detailed Description

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

Author

Thapelo Thoka

Date

October 2022

4.7.2 Constructor & Destructor Documentation

4.7.2.1 Commander()

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

Commander Constructor which takes in the name of a commander.

Parameters

name	- name of commander

4.7.3 Member Function Documentation

4.7.3.1 attach()

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

Attaches a Raven observer to all the bannerman objects in the groundForces list.

Parameters

o	- the Raven observer object

Implements Bannerman.

4.7.3.2 attack()

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

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

Parameters

myBannerman	- The attacking bannerman object.
enemyBannerman	- The bannerman object being attacked.

Implements Bannerman.

4.7.3.3 changeStrategy()

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

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

Parameters

strategy	- The new strategy bannerman objects in groundForces should have.

Implements Bannerman.

4.7.3.4 decreaseFavour()

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

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

Implements Bannerman.

4.7.3.5 decreaseFood()

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

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

Implements Bannerman.

4.7.3.6 decreaseMedical()

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

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

Implements Bannerman.

4.7.3.7 decreasePower()

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

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

Parameters

x	- the number by which to decrease damage.

Implements Bannerman.

4.7.3.8 decreaseWeapons()

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

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

Implements Bannerman.

4.7.3.9 detach()

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

Detaches a Raven observer from all the bannerman objects in the groundForces list.

Parameters

o	- the Raven observer object

Implements Bannerman.

4.7.3.10 getDamage()

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

Accessor. Returns the damage.

Returns

The total groundForces damage.

Implements Bannerman.

4.7.3.11 getFavour()

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

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

Returns

favour

Implements Bannerman.

4.7.3.12 getFood()

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

Accessor. Returns the total groundForces amount of food.

Returns

numFood.

Implements Bannerman.

4.7.3.13 getHP()

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

Accessor. Returns the HP.

Returns

The total groundForces HP.

Implements Bannerman.

4.7.3.14 getMedical()

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

Accessor. Returns the total groundForces amount of medical supplies.

Returns

numMedical.

Implements Bannerman.

4.7.3.15 getName()

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

Returns the name of this commander object.

Returns

name

Implements Bannerman.

4.7.3.16 getTroops()

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

Returns the groundForces bannerman list

Implements Bannerman.

4.7.3.17 getWeapons()

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

Accessor. Returns numWeapons.

Returns

The total groundForces number of Weapons.

Implements Bannerman.

4.7.3.18 increaseFavour()

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

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

Implements Bannerman.

4.7.3.19 increaseHP()

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

increases the HP variables of the bannerman objects in groundForces.

Parameters

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

Implements Bannerman.

4.7.3.20 increasePower()

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

increases the damage variables of the bannerman objects in groundForces.

Parameters

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

Implements Bannerman.

4.7.3.21 receiveDamage()

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

increases the damage variables of the groundForces by x

Parameters

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

Implements Bannerman.

4.7.3.22 setFood()

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

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

Parameters

numFood	- The new numFood bannerman objects in groundForces should have.

Implements Bannerman.

4.7.3.23 setMaster()

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

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

Parameters

m	- the new MasterOfCoin mediator.

Implements Bannerman.

4.7.3.24 setMedical()

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

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

Parameters

numMedical	- The new numMedical bannerman objects in groundForces should have.

Implements Bannerman.

4.7.3.25 setRaven()

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

Assigns a Raven observer list to all the bannerman objects in the groundForces list.

Parameters

r	- the Raven observer list to attach

Implements Bannerman.

4.7.3.26 setStrategy()

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

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

Parameters

strategy	- The new strategy bannerman objects in groundForces should have.

4.7.3.27 setWeapons()

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

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

Parameters

numWeapons	- The new numWeapons bannerman objects in groundForces should have.

Implements Bannerman.

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

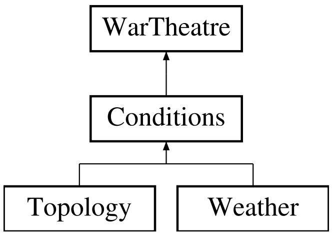
- Commander.h
- Commander.cpp

4.8 Conditions Class Reference

decorator class. This is the class/object decorates the concreteComponent. Inherits from WarTheatre class.

`#include <Conditions.h>`

Inheritance diagram for Conditions:



Public Member Functions

- void sendScout ()

  decorator function for thr pattern
- Conditions (WarTheatre ∗myVenue)

  constructor for the decorator class. merely used to assign variables and call the decorator function
- virtual ∼**Conditions** ()

  destructor to delete the WarTheatre pointer

Private Attributes

- WarTheatre ∗ myVenue

4.8.1 Detailed Description

decorator class. This is the class/object decorates the concreteComponent. Inherits from WarTheatre class.

Author

Keabetswe Mothapo

Date

October 2022

4.8.2 Constructor & Destructor Documentation

4.8.2.1 Conditions()

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

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

Parameters

myVenue	- pointer to the object that is to be decoraed

4.8.3 Member Function Documentation

4.8.3.1 sendScout()

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

decorator function for thr pattern

Implements WarTheatre.

Reimplemented in Topology, and Weather.

4.8.4 Member Data Documentation

4.8.4.1 myVenue

`WarTheatre* Conditions::myVenue [private]`

Pointer to the decorated component

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

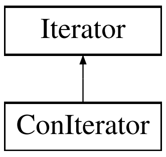
- Conditions.h
- Conditions.cpp

4.9 ConIterator Class Reference

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

`#include <ConIterator.h>`

Inheritance diagram for ConIterator:



Public Member Functions

- ConIterator (list< Bannerman $*$ > X)
- Bannerman $*$ Current ()

  Returns a pointer to the current bannerman object in groundForces being accessed, , primitive operations can be performed on the returned bannerman object.
- Bannerman $*$ next ()

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

  Determines whether or not the current bannerman object which (∗it) currently points to is the last element in the list or not.
- bool isActive ()

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

Private Attributes

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

4.9.1 Detailed Description

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

Author

Thapelo Thoka

Date

October 2022

4.9.2 Constructor & Destructor Documentation

4.9.2.1 ConIterator()

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

ConIterator Constructor which takes in a list of bannerman objects.

Parameters

X	- the groundForces list which will be accessed sequentially.

4.9.3 Member Function Documentation

4.9.3.1 Current()

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

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

Returns

Bannerman*

Implements Iterator.

4.9.3.2 hasNext()

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

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

Returns

false

true

Test

Implements Iterator.

4.9.3.3 isActive()

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

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

Returns

true

false

Test

Implements Iterator.

4.9.3.4 next()

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

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

Returns

Bannerman∗

Test

Implements Iterator.

4.9.4 Member Data Documentation

4.9.4.1 armyList

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

The list of bannerman objects to traverse

4.9.4.2 it

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

An iterator which is a pointer to the current bannerman object in Commander's groundForces being accessed, primative operations can be performed using this pointer.

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

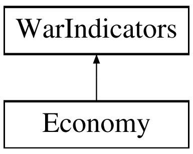
- ConIterator.h
- ConIterator.cpp

4.10 Economy Class Reference

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

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

Inheritance diagram for Economy:



Public Member Functions

- Economy (State *state, int currency)

    Constructor. initializes State pointer and currency.
- void **SetState** ()

    tells State to invoke its getDemotionState method.
- State * getState ()

    returns State pointer.
- void **decreaseCurrency** ()

    tells State to invoke its decreaseCurrency method and then it invokes notify method of MasterofCoin passing itself as the argument.
- int getCurrency ()

    returns currency.
- void removeCurrency (int i)

    reduces currency by the passed in amount.
- virtual ~**Economy** ()

    destructor. deallocates State.

Private Attributes

- State ∗ state
- int currency

Additional Inherited Members

4.10.1 Detailed Description

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

Author

Morgan Bentley

Date

October 2022

4.10.2 Constructor & Destructor Documentation

4.10.2.1 Economy()

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

Constructor. initializes State pointer and currency.

Parameters

state	- State of Economy.
currency	- Monetary reserves of the Kingdom.

4.10.3 Member Function Documentation

4.10.3.1 getCurrency()

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

returns currency.

Returns

currency of Economy.

4.10.3.2 getState()

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

returns State pointer.

Returns

State of Economy.

4.10.3.3 removeCurrency()

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

reduces currency by the passed in amount.

Parameters

i	- Expense of manufacturing required Supplies.

4.10.4 Member Data Documentation

4.10.4.1 currency

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

Monetary reserves of the Kingdom

4.10.4.2 state

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

state of economy

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

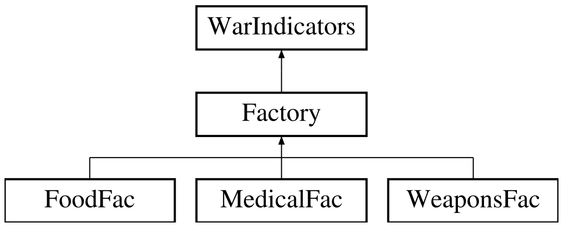
- Economy.h
- Economy.cpp

4.11 Factory Class Reference

Factory class to serve as a base class for all factory concrete classes used to create supplies.

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

Inheritance diagram for Factory:



Public Member Functions

- void **operation** ()

    operation() is the template Method that calls make(), indirectly manufacturing the supplies.
- ArmySupplies ∗ getSupply ()

    getsupply() is the get Method used to get the supply variable.
- virtual ∼**Factory** ()

    ∼Factory() is the destructer for the Factory class. It is never implemented as its derived classes delete the supply pointer.

Protected Member Functions

- virtual ArmySupplies ∗ make ()=0

    make() is the factory Method called in operation() to make the new supplies.

Private Attributes

- ArmySupplies ∗ supply

Additional Inherited Members

4.11.1 Detailed Description

Factory class to serve as a base class for all factory concrete classes used to create supplies.

Author

Ronin Brookes 19069686

4.11.2 Member Function Documentation

4.11.2.1 getSupply()

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

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

Returns

ArmySupplies pointer to the newly created supplies.

4.11.2.2 make()

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

make() is the factory Method called in operation() to make the new supplies.

Returns

ArmySupplies pointer to the newly created supplies.

Implemented in FoodFac, MedicalFac, and WeaponsFac.

4.11.3 Member Data Documentation

4.11.3.1 supply

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

pointer to the supplies

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

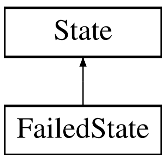
- Factory.h
- Factory.cpp

4.12 FailedState Class Reference

A concrete state class. A class that is one of the alternative concrete states for Economy of Kingdom.

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

Inheritance diagram for FailedState:



Public Member Functions

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

Additional Inherited Members

4.12.1 Detailed Description

A concrete state class. A class that is one of the alternative concrete states for Economy of Kingdom.

Author

Morgan Bentley

Date

October 2022

4.12.2 Member Function Documentation

4.12.2.1 decreaseCurrency()

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

prints out string stating that current State is FailedState

Implements State.

4.12.2.2 getState()

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

tells what the State of the Economy is.

Returns

returns a string that says in text what the State is.

Implements State.

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

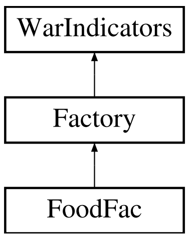
- FailedState.h
- FailedState.cpp

4.13 FoodFac Class Reference

Factory class to serve as a concrete creator class used to create Food supplies.

`#include <FoodFac.h>`

Inheritance diagram for FoodFac:



Public Member Functions

- ArmySupplies ∗ make ()

    make() is the factory Method called in operation() to make the Food supplies.
- ∼**FoodFac** ()

    destructor for the FoodFac class to delete and free the supply variable.

Additional Inherited Members

4.13.1 Detailed Description

Factory class to serve as a concrete creator class used to create Food supplies.

Author

Ronin Brookes 19069686

4.13.2 Member Function Documentation

4.13.2.1 make()

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

make() is the factory Method called in operation() to make the Food supplies.

Returns

ArmySupplies pointer to the newly created Food supplies.

Implements Factory.

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

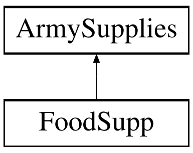
- FoodFac.h
- FoodFac.cpp

4.14 FoodSupp Class Reference

Concrete Product Class containing the information on the Food supplies.

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

Inheritance diagram for FoodSupp:



Public Member Functions

- int getAmount ()

  getAmount() is used to return the amount of Food supplies.

4.14.1 Detailed Description

Concrete Product Class containing the information on the Food supplies.

Author

Ronin Brookes 19069686

4.14.2 Member Function Documentation

4.14.2.1 getAmount()

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

getAmount() is used to return the amount of Food supplies.

Returns

an integer representation of the amount of Food supplies.

Implements ArmySupplies.

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

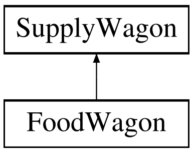
- FoodSupp.h
- FoodSupp.cpp

4.15 FoodWagon Class Reference

Concrete Prototype class to serve as the wagon for Food.

`#include <FoodWagon.h>`

Inheritance diagram for FoodWagon:



Public Member Functions

- void setSup (ArmySupplies ∗sup)

  initialize the supp variable for the FoodWagon.
- SupplyWagon ∗ clone ()

  Clone the Food Supply Wagon object.
- ArmySupplies ∗ getSupplies ()

  Return the supp variable for the Food Wagon.
- ∼**FoodWagon** ()

  destructor for the Food Wagon class to delete and free the supply variable.

Public Attributes

- ArmySupplies ∗ supp =NULL

4.15.1 Detailed Description

Concrete Prototype class to serve as the wagon for Food.

Author

Ronin Brookes 19069686

4.15.2 Member Function Documentation

4.15.2.1 clone()

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

Clone the Food Supply Wagon object.

Returns

the new Food Supply Wagon clone.

Implements SupplyWagon.

4.15.2.2 getSupplies()

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

Return the supp variable for the Food Wagon.

Returns

returns the supp variable. Returns the necessary supplies.

Implements SupplyWagon.

4.15.2.3 setSup()

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

initialize the supp variable for the FoodWagon.

Parameters

sup	is used to set the supp variable,

Implements SupplyWagon.

4.15.3 Member Data Documentation

4.15.3.1 supp

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

pointer to the supplies in the Food wagon

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

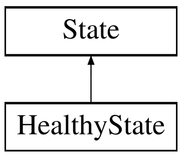
- FoodWagon.h
- FoodWagon.cpp

4.16 HealthyState Class Reference

A concrete state class. A class that is one of the alternative concrete states for Economy of Kingdom.

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

Inheritance diagram for HealthyState:



Public Member Functions

- **HealthyState** ()

    Default constructor. initializes context to null.
- void decreaseCurrency ()

    decreases Economy currency and then checks if the conditions are sufficient for the economy to downgrade to a lower state
- virtual State ∗ getDemotionState ()

    gets lower level state of economy
- string getState ()

    tells what the State of the Economy is.

Additional Inherited Members

4.16.1 Detailed Description

A concrete state class. A class that is one of the alternative concrete states for Economy of Kingdom.

Author

Morgan Bentley

Date

October 2022

4.16.2 Member Function Documentation

4.16.2.1 decreaseCurrency()

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

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

Implements State.

4.16.2.2 getDemotionState()

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

gets lower level state of economy

Returns

concrete State of Economy

Reimplemented from State.

4.16.2.3 getState()

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

tells what the State of the Economy is.

Returns

returns a string that says in text what the State is.

Implements State.

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

- HealthyState.h
- HealthyState.cpp

4.17 Historian Class Reference

Public Member Functions

- void setAlly (Bannerman ∗b)

  sets the bannerman variable
- History ∗ Store ()

  stores the bannerman in a memento object (history).
- Bannerman ∗ restoreAlly (History ∗h)

  reverts the bannerman that was stored to original state for when they return.
- ∼**Historian** ()

  the destructor.

Private Attributes

- Bannerman ∗ bannerman

  The originator class in the memento design pattern.

4.17.1 Member Function Documentation

4.17.1.1 restoreAlly()

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

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

Parameters

h	the history object being restored.

Returns

the bannerman that was restored from the stored history object.

4.17.1.2 setAlly()

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

sets the bannerman variable

Parameters

b	the bannerman being set to the class variable

4.17.1.3 Store()

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

stores the bannerman in a memento object (history).

Returns

the resulting history object with the bannerman.

4.17.2 Member Data Documentation

4.17.2.1 bannerman

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

The originator class in the memento design pattern.

Author

Julianna Venter.

Date

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

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

- Historian.h
- Historian.cpp

4.18 History Class Reference

Public Member Functions

- History (Bannerman ∗b)

    constructor that sets class variable.
- Bannerman ∗ getBannerman ()

    returns the stored bannerman
- ∼**History** ()

    the destructor

Private Attributes

- Bannerman ∗ bannerman

    The memento class in the memento design pattern.

4.18.1 Constructor & Destructor Documentation

4.18.1.1 History()

```
History::History (
             Bannerman ∗ b )
```

constructor that sets class variable.

Parameters

b	parameter that class variable is set to.

4.18.2 Member Function Documentation

4.18.2.1 getBannerman()

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

returns the stored bannerman

Returns

the stored bannerman

4.18.3 Member Data Documentation

4.18.3.1 bannerman

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

The memento class in the memento design pattern.

Author

Julianna Venter

Date

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

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

- History.h
- History.cpp

4.19 HistoryBook Class Reference

Public Member Functions

- void add (History ∗h)

    adds the passed in history object to the class list
- History ∗ getAlly ()

    gets the required history object to be restored

Private Attributes

- list< History ∗ > defectedAllies

    The caretaker class in the memento design pattern.

4.19.1 Member Function Documentation

4.19.1.1 add()

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

adds the passed in history object to the class list

Parameters

h	the history object to be added

4.19.1.2 getAlly()

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

gets the required history object to be restored

Returns

the history object that will be restored

4.19.2 Member Data Documentation

4.19.2.1 defectedAllies

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

The caretaker class in the memento design pattern.

Author

Julianna Venter

Date

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

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

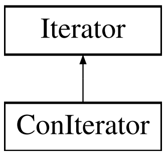
- HistoryBook.h
- HistoryBook.cpp

4.20 Iterator Class Reference

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

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

Inheritance diagram for Iterator:



Public Member Functions

- virtual Bannerman $*$ Current ()=0

  Abstract. Returns a pointer to the current bannerman object in groundForces being accessed, , primitive operations can be performed on the returned bannerman object.
- virtual bool hasNext ()=0

  Abstract. determines whether or not the current bannerman object being accessed in groundForces is the last element in the list or not.
- virtual Bannerman $*$ next ()=0

  Abstract. Sets the current pointer to the next bannerman object in groundForces to be accessed and returns it, , primitive operations can be performed on the returned bannerman object.
- virtual bool isActive ()=0

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

4.20.1 Detailed Description

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

Author

Thapelo Thoka

Date

October 2022

4.20.2 Member Function Documentation

4.20.2.1 Current()

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

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

Returns

Bannerman∗

Implemented in ConIterator.

4.20.2.2 hasNext()

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

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

Returns

false

true

Implemented in ConIterator.

4.20.2.3 isActive()

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

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

Returns

true
false

Implemented in ConIterator.

4.20.2.4 next()

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

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

Returns

Bannerman∗

Implemented in ConIterator.

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

- Iterator.h

4.21 Kingdom Class Reference

A Kingdom class that has all the fighting Bannerman and Economy.

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

Public Member Functions

- Kingdom (Economy ∗economy)

  Constructor. initializes Economy pointer and vector list.
- void remove (Bannerman ∗b)

  removes specified Bannerman from vector list.
- void add (Bannerman ∗b)

  adds specified Bannerman into vector list.
- virtual ∼**Kingdom** ()

  destructor. deallocates all pointers of this class
- int getSize ()

  returns current number of Bannerman in the Kingdom.
- Bannerman ∗ getAlly (string n)

  returns a particular allied Bannerman given a unique string name.
- list< Bannerman ∗ > getKingdom ()

  returns a list of Bannerman pointers in the Kingdom.

Private Attributes

- list< Bannerman ∗ > bannerman
- Economy ∗ economy

4.21.1 Detailed Description

A Kingdom class that has all the fighting Bannerman and Economy.

Author

Morgan Bentley

Date

October 2022

4.21.2 Constructor & Destructor Documentation

4.21.2.1 Kingdom()

```
Kingdom::Kingdom (
            Economy ∗ economy )
```

Constructor. initializes Economy pointer and vector list.

Parameters

economy	- Economy pointer to player's Economy object.

4.21.3 Member Function Documentation

4.21.3.1 add()

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

adds specified Bannerman into vector list.

Parameters

b	- Bannerman that has defected from enemyKingdom

4.21.3.2 getAlly()

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

returns a particular allied Bannerman given a unique string name.

Parameters

n	- unique name of a particular Bannerman.

Returns

a particular allied Bannerman.

4.21.3.3 getKingdom()

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

returns a list of Bannerman pointers in the Kingdom.

Returns

vector list of all Bannerman objects a Kingdom owns.

4.21.3.4 getSize()

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

returns current number of Bannerman in the Kingdom.

Returns

integer representing the number of Bannerman in Kingdom.

4.21.3.5 remove()

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

removes specified Bannerman from vector list.

Parameters

b	- Bannerman that has lost a fight or defected to enemyKingdom.

4.21.4 Member Data Documentation

4.21.4.1 bannerman

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

vector list of all Bannerman objects a Kingdom owns

4.21.4.2 economy

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

pointer to the Economy of the Kingdom

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

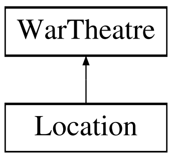
- Kingdom.h
- Kingdom.cpp

4.22 Location Class Reference

concrete object. This is the class/object that will be decorated. Inherits from WarTheatre class

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

Inheritance diagram for Location:



Public Member Functions

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

4.22.1 Detailed Description

concrete object. This is the class/object that will be decorated. Inherits from WarTheatre class

Author

Keabetswe Mothapo

Date

October 2022

4.22.2 Member Function Documentation

4.22.2.1 sendScout()

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

the function that will be called to decorate the war venue

Implements WarTheatre.

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

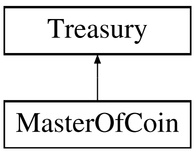
- Location.h
- Location.cpp

4.23 MasterOfCoin Class Reference

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

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

Inheritance diagram for MasterOfCoin:



Public Member Functions

- MasterOfCoin (Economy ∗economy, Raven ∗observer, Strategy ∗strategy)

  Constructor to set the variables in this class.
- void notify (WarIndicators ∗sender)

  This is the method that all objects will use to call the mediator when a change is made.
- void **decreaseCurrency** ()

  Will call the decreaseCurrency method of the Economy.
- void **manufacture** ()

  will call on the obsever to create more supplies
- ∼**MasterOfCoin** ()

  This is the destructer used to deallocate all memory used in this class.

Private Attributes

- Economy ∗ **economy**

  The Economy that the mediator uses.
- Raven ∗ **observer**

  The Observer that the mediator uses.
- Strategy ∗ **strategy**

  The Strategy the mediator uses.

4.23.1 Detailed Description

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

Author

Sameet Keshav u21479373

Date

October 2022

4.23.2 Constructor & Destructor Documentation

4.23.2.1 MasterOfCoin()

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

Constructor to set the variables in this class.

Parameters

economy	holds the Economy that will be using the mediator
observer	holds the observer that will use the mediator
strategy	holds the strategy object that will use the mediator

4.23.3 Member Function Documentation

4.23.3.1 notify()

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

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

Parameters

sender	is the variable that holds the object who sent the notify request

Implements Treasury.

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

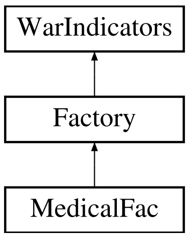
- MasterOfCoin.h
- MasterOfCoin.cpp

4.24 MedicalFac Class Reference

Factory class to serve as a concrete creator class used to create Medical supplies.

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

Inheritance diagram for MedicalFac:



Public Member Functions

- ArmySupplies ∗ make ()

  make() is the factory Method called in operation() to make the Medical supplies.
- ∼**MedicalFac** ()

  destructor for the MedicalFac class to delete and free the supply variable.

Additional Inherited Members

4.24.1 Detailed Description

Factory class to serve as a concrete creator class used to create Medical supplies.

Author

Ronin Brookes 19069686

4.24.2 Member Function Documentation

4.24.2.1 make()

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

make() is the factory Method called in operation() to make the Medical supplies.

Returns

ArmySupplies pointer to the newly created Medical supplies.

Implements Factory.

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

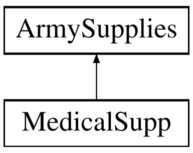
- MedicalFac.h
- MedicalFac.cpp

4.25 MedicalSupp Class Reference

Concrete Product Class containing the information on the Medical supplies.

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

Inheritance diagram for MedicalSupp:



Public Member Functions

- int getAmount ()

  getAmount() is used to return the amount of Medical supplies.

4.25.1 Detailed Description

Concrete Product Class containing the information on the Medical supplies.

Author

Ronin Brookes 19069686

4.25.2 Member Function Documentation

4.25.2.1 getAmount()

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

getAmount() is used to return the amount of Medical supplies.

Returns

an integer representation of the amount of Medical supplies.

Implements ArmySupplies.

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

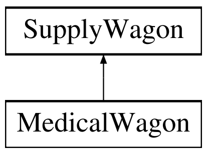
- MedicalSupp.h
- MedicalSupp.cpp

4.26 MedicalWagon Class Reference

Concrete Prototype class to serve as the wagon for Medical.

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

Inheritance diagram for MedicalWagon:



Public Member Functions

- void setSup (ArmySupplies ∗sup)

    initialize the supp variable for the MedicalWagon.
- SupplyWagon ∗ clone ()

    Clone the Medical Supply Wagon object.
- ArmySupplies ∗ getSupplies ()

    Return the supp variable for the Medical Wagon.
- ∼**MedicalWagon** ()

    destructor for the Medical Wagon class to delete and free the supply variable.

Public Attributes

- ArmySupplies ∗ supp

4.26.1 Detailed Description

Concrete Prototype class to serve as the wagon for Medical.

Author

Ronin Brookes 19069686

4.26.2 Member Function Documentation

4.26.2.1 clone()

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

Clone the Medical Supply Wagon object.

Returns

the new Medical Supply Wagon clone.

Implements SupplyWagon.

4.26.2.2 getSupplies()

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

Return the supp variable for the Medical Wagon.

Returns

returns the supp variable. Returns the necessary supplies.

Implements SupplyWagon.

4.26.2.3 setSup()

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

initialize the supp variable for the MedicalWagon.

Parameters

sup	is used to set the supp variable,

Implements SupplyWagon.

4.26.3 Member Data Documentation

4.26.3.1 supp

```
ArmySupplies* MedicalWagon::supp
```

pointer to the supplies in the Medical wagon

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

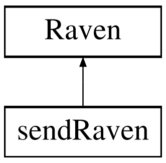
- MedicalWagon.h
- MedicalWagon.cpp

4.27 Raven Class Reference

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

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

Inheritance diagram for Raven:



Public Member Functions

- virtual void update ()=0
  The Pure Virtual function for the update method that the concrete class uses.

4.27.1 Detailed Description

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

Author

Sameet Keshav u21479373

Date

October 2022

4.27.2 Member Function Documentation

4.27.2.1 update()

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

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

Implemented in sendRaven.

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

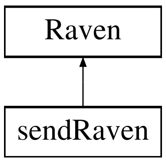
- Raven.h

4.28 sendRaven Class Reference

The Concrete class for the Observer A class that watches the Bannerman for any changes that are made.

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

Inheritance diagram for sendRaven:



Public Member Functions

- sendRaven (SupplyWagon **supplies, Bannerman *subject)

  constructor for the concrete Raven class
- void update ()

  This method will be called from outside the class, this method will get the amount of resources the subject has, then call the check method.
- void **checkSupplies** ()

  This method will check to see if there is too little supplies then call the appropriate Supply Wagon to fill it up.
- ∼**sendRaven** ()

  The Destructor will be used to deference all pointer variables used by this class to stop all memory leaks.

Private Attributes

- int **numFood**

    Amount of food the bannerman has.
- int **numMedical**

    Amount of medical supplies the bannerman has.
- int **numWeapons**

    Amount of medical supplies the bannerman has.
- SupplyWagon ∗∗ **supplies**

    The array of SupplyWagon's used to hold the different types of supplies.
- Bannerman ∗ **subject**

    The Bannerman subject the Observer watches.

4.28.1 Detailed Description

The Concrete class for the Observer A class that watches the Bannerman for any changes that are made.

Author

Sameet Keshav u21479373

Date

October 2022

4.28.2 Constructor & Destructor Documentation

4.28.2.1 sendRaven()

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

constructor for the concrete Raven class

Parameters

supplies	is an array of pointer objects of type SupplyWagon
subject	is a pointer for the Bannerman subject that this Observer will observe

4.28.3 Member Function Documentation

4.28.3.1 update()

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

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

Implements Raven.

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

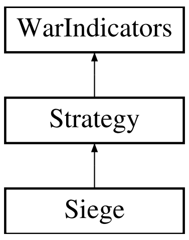
- sendRaven.h
- sendRaven.cpp

4.29 Siege Class Reference

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

`#include <Siege.h>`

Inheritance diagram for Siege:



Public Member Functions

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

  constructor. initializes stealth and calls base class constructor to initialize myKingdom, enemyKingdom, my↩ Bannerman, enemyBannerman, strategy, min and minFavour.
- bool attack (Bannerman ∗myBannerman, Bannerman ∗enemyBannerman)

  makes two bannerman from enemy kingdoms fight each other until one loses
- ∼**Siege** ()

  destructor.

Private Attributes

- int stealth

Additional Inherited Members

4.29.1 Detailed Description

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

Author

Morgan Bentley

Date

October 2022

4.29.2 Constructor & Destructor Documentation

4.29.2.1 Siege()

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

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

Parameters

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

4.29.3 Member Function Documentation

4.29.3.1 attack()

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

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

Parameters

myBannerman	- attacking bannerman object.
enemyBannerman	- Bannerman object being attacked.

Returns

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

Implements Strategy.

4.29.4 Member Data Documentation

4.29.4.1 stealth

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

skill level of Bannerman using this strategy

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

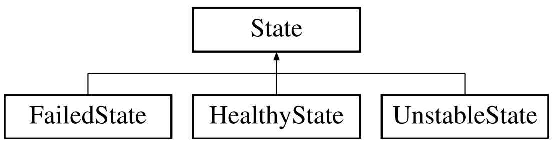
- Siege.h
- Siege.cpp

4.30 State Class Reference

An Abstract state class. A class that provides an interface to the alternative concrete states for Economy of Kingdom.

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

Inheritance diagram for State:



Public Member Functions

- **State** ()

  Default constructor.
- virtual void setContext (Economy ∗context)

  sets context variable.
- virtual void decreaseCurrency ()=0

  Abstract method.
- virtual State ∗ getDemotionState ()

  gets lower level state of Economy with null being returned if no lower State is possible
- virtual ∼**State** ()

  destructor. deallocates context
- virtual string getState ()=0

  Abstract method.

Protected Attributes

- Economy ∗ context

4.30.1 Detailed Description

An Abstract state class. A class that provides an interface to the alternative concrete states for Economy of Kingdom.

Author

Morgan Bentley

Date

October 2022

4.30.2 Member Function Documentation

4.30.2.1 decreaseCurrency()

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

Abstract method.

Implemented in FailedState, HealthyState, and UnstableState.

4.30.2.2 getDemotionState()

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

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

Returns

concrete State of Economy

Reimplemented in HealthyState, and UnstableState.

4.30.2.3 getState()

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

Abstract method.

Returns

name of State as a string.

Implemented in FailedState, HealthyState, and UnstableState.

4.30.2.4 setContext()

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

sets context variable.

Parameters

context	- Economy pointer to player's Economy object.

4.30.3 Member Data Documentation

4.30.3.1 context

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

Economy pointer of player's Kingdom

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

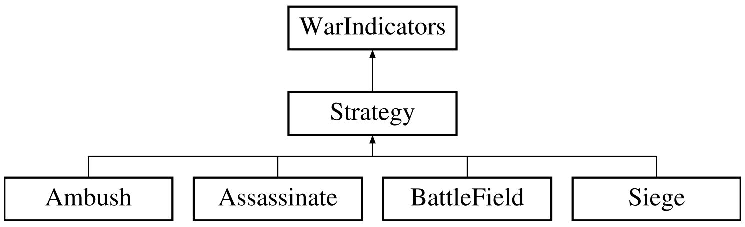
- State.h
- State.cpp

4.31 Strategy Class Reference

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

`#include <Strategy.h>`

Inheritance diagram for Strategy:



Public Member Functions

- Strategy (Kingdom ∗myKingdom, Kingdom ∗enemyKingdom, Bannerman ∗myBannerman, Bannerman ∗enemyBannerman, string name, int min, int minFavour, Historian ∗h, HistoryBook ∗hb)

  constructor. initializes myKingdom, enemyKingdom, myBannerman, enemyBannerman, strategy, minSupplies and minFavour.
- virtual bool attack (Bannerman ∗myBannerman, Bannerman ∗enemyBannerman)=0

  Abstract method.
- virtual string getStrategyName ()

  gets strategy variable
- virtual Bannerman ∗ getMyBannerman ()

  gets attacking Bannerman
- virtual Bannerman ∗ getEnemyBannerman ()

  gets Bannerman being attacked
- virtual ∼**Strategy** ()

  destructor. deallocates all pointers of this class

Protected Attributes

- Kingdom ∗ myKingdom
- Kingdom ∗ enemyKingdom
- Bannerman ∗ myBannerman
- Bannerman ∗ enemyBannerman
- string strategy
- int minFavour
- int minSupplies
- HistoryBook ∗ BookOfDura
- Historian ∗ Greg
- int defectedAllies

4.31.1 Detailed Description

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

Author

Morgan Bentley

Date

October 2022

4.31.2 Constructor & Destructor Documentation

4.31.2.1 Strategy()

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

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

Parameters

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

4.31.3 Member Function Documentation

4.31.3.1 attack()

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

Abstract method.

Parameters

myBannerman	- attacking Bannerman object.
enemyBannerman	- Bannerman object being attacked.

Returns

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

Implemented in Ambush, Assassinate, BattleField, and Siege.

4.31.3.2 getEnemyBannerman()

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

gets Bannerman being attacked

Returns

enemyBannerman pointer

4.31.3.3 getMyBannerman()

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

gets attacking Bannerman

Returns

myBannerman pointer

4.31.3.4 getStrategyName()

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

gets strategy variable

Returns

strategy variable

4.31.4 Member Data Documentation

4.31.4.1 BookOfDura

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

Caretaker for memento implementation

4.31.4.2 defectedAllies

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

amount of defected allies

4.31.4.3 enemyBannerman

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

defending Bannerman

4.31.4.4 enemyKingdom

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

Kingdom pointer of enemyBannerman's Kingdom

4.31.4.5 Greg

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

Historian (originator) to save defected bannemen

4.31.4.6 minFavour

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

minimum favour below which Bannerman change allegiances

4.31.4.7 minSupplies

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

minimum supplies for food,weapons and medicine

4.31.4.8 myBannerman

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

attacking Bannerman

4.31.4.9 myKingdom

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

Kingdom pointer of attacking Bannerman's Kingdom

4.31.4.10 strategy

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

name of concrete strategy

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

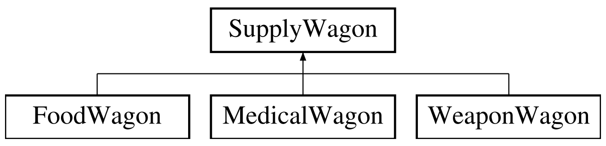
- Strategy.h
- Strategy.cpp

4.32 SupplyWagon Class Reference

Abstract Prototype class SupplyWagon to serve as the base class for all derived concrete prototypes.

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

Inheritance diagram for SupplyWagon:



Public Member Functions

- virtual void setSup (ArmySupplies ∗sup)=0

  initialize the supp variable.
- virtual SupplyWagon ∗ clone ()=0

  Clone the Supply Wagon object.
- virtual ArmySupplies ∗ getSupplies ()=0

  Return the supp variable.

Public Attributes

- ArmySupplies ∗ supp

4.32.1 Detailed Description

Abstract Prototype class SupplyWagon to serve as the base class for all derived concrete prototypes.

Author

Ronin Brookes 19069686

4.32.2 Member Function Documentation

4.32.2.1 clone()

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

Clone the Supply Wagon object.

Returns

the new Supply Wagon clone.

Implemented in FoodWagon, MedicalWagon, and WeaponWagon.

4.32.2.2 getSupplies()

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

Return the supp variable.

Returns

returns the supp variable. Returns the necessary supplies.

Implemented in FoodWagon, MedicalWagon, and WeaponWagon.

4.32.2.3 setSup()

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

initialize the supp variable.

Parameters

sup	used to set the supp variable.

Implemented in FoodWagon, MedicalWagon, and WeaponWagon.

4.32.3 Member Data Documentation

4.32.3.1 supp

```
ArmySupplies* SupplyWagon::supp
```

pointer to the supplies in the wagon

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

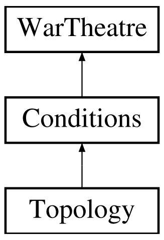
- SupplyWagon.h

4.33 Topology Class Reference

concrete decorator B. This is class that decorates the topology of the venue. Inherits from Conditions class

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

Inheritance diagram for Topology:



Public Member Functions

- void **setTopology** ()

  setter for the topology of the war venue uses the inherited venue variable to adjust difficuty of teh terrian
- **Topology** (WarTheatre ∗myTheatre)

  default constructor that calls the decorator function
- void sendScout ()

  implements the decorator function

4.33.1 Detailed Description

concrete decorator B. This is class that decorates the topology of the venue. Inherits from Conditions class

Author

Keabetswe Mothapo

Date

October 2022

4.33.2 Member Function Documentation

4.33.2.1 sendScout()

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

implements the decorator function

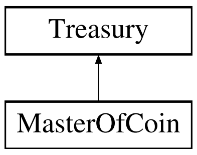
Reimplemented from Conditions.

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

- Topology.h
- Topology.cpp

4.34 Treasury Class Reference

Inheritance diagram for Treasury:



Public Member Functions

- virtual void notify (WarIndicators ∗sender)=0

  The pure virtual function for the notify function.

4.34.1 Member Function Documentation

4.34.1.1 notify()

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

The pure virtual function for the notify function.

Implemented in MasterOfCoin.

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

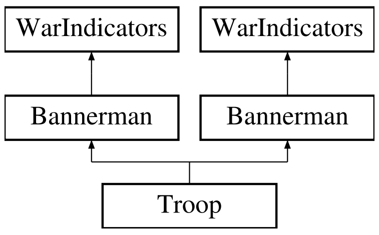
- Treasury.h

4.35 Troop Class Reference

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

`#include <Troop.h>`

Inheritance diagram for Troop:



Public Member Functions

- **Troop** (string name, int favor, int numFood, int Medical, int HP, WarTheatre ∗warZone, Strategy ∗strategy, MasterOfCoin ∗m, bool assassin, int size)

  Constructor. Initializes the name; favour; numFood; Medical; HP; warZone; strategy; m; assassin and size variables of the troop.
- int getHP () override

  Returns the troop's HP.
- int getSize ()

  Returns the troop's size.
- void attach (Raven ∗o) override

  Attaches a Raven observer to the troop.
- void detach (Raven ∗o) override

  Detaches a Raven observer to the troop.
- void increasePower (int boost) override

  increases the damage variables of the troop.
- void attack (Bannerman ∗myBannerman, Bannerman ∗enemyBannerman) override

  uses strategy to make a troop attack another kingdom.
- void decreaseWeapons () override

  decreases the numWeapons variable of the troop by 1.
- void decreaseFood () override

decreases the numFood variable of the troop by 1.

- void decreaseMedical () override

  decreases the numMedical variable of the troop by 1.
- void changeStrategy (Strategy ∗strategy) override

  Changes the attack strategy of the troop.
- void increaseFavour () override

  Increases the loyalty favour level of the troop by 1.
- void decreaseFavour () override

  Decreases the loyalty favour level of the troop by 1.
- int getFavour () override

  Returns the the loyalty favour levels variable of the Troop object.
- void increaseHP (int boost) override

  increases the HP of the troop.
- int getDamage () override

  Accessor. Returns the damage capacity of the troop.
- void receiveDamage (int X) override

  increases the damage capacity of the troop.
- int getWeapons () override

  Accessor. Returns the number of weapons the troop has.
- int getFood () override

  Accessor. Returns the amount of food the troop has.
- int getMedical () override

  Accessor. Returns the number of medical supplies the troop has.
- void setWeapons (int numWeapons) override

  Sets the number of weapon supplies the troop has.
- void setFood (int numFood) override

  Sets the number of food supplies the troop has.
- void decreasePower (int x) override

  Decreases the damage capability of the troop.
- void setMedical (int numMedical) override

  Sets the number of medical supplies the troop has.
- void setRaven (list< Raven ∗ > r) override

  Assigns a list of Raven Observers to the troop's ravenList.
- void setMaster (MasterOfCoin ∗m) override

  Assigns a MasterofCoin mediator to ensure that the troop has the supplies it needs.
- ∼Troop ()
- string getName () override

  Returns the name of this Troop object.
- list< Bannerman ∗ > getTroops () override

  Bannerman pure virtual method, not used by troop.

Private Attributes

- int size
- Strategy ∗ strategy
- int damage
- WarTheatre ∗ warZone

Additional Inherited Members

4.35.1 Detailed Description

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

Author

Thapelo Thoka

Date

October 2022

4.35.2 Constructor & Destructor Documentation

4.35.2.1 ∼Troop()

```
Troop::∼Troop ( )
```

Default destructor.

4.35.3 Member Function Documentation

4.35.3.1 attach()

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

Attaches a Raven observer to the troop.

Parameters

o	- the Raven observer object to attach

Implements Bannerman.

4.35.3.2 attack()

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

uses strategy to make a troop attack another kingdom.

Parameters

myBannerman	- The attacking bannerman object.
enemyBannerman	- The bannerman object being attacked.

Implements Bannerman.

4.35.3.3 changeStrategy()

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

Changes the attack strategy of the troop.

Parameters

strategy	- The new strategy the troop should have.

Implements Bannerman.

4.35.3.4 decreaseFavour()

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

Decreases the loyalty favour level of the troop by 1.

Implements Bannerman.

4.35.3.5 decreaseFood()

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

decreases the numFood variable of the troop by 1.

Implements Bannerman.

4.35.3.6 decreaseMedical()

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

decreases the numMedical variable of the troop by 1.

Implements Bannerman.

4.35.3.7 decreasePower()

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

Decreases the damage capability of the troop.

Parameters

x	- the number by which to decrease damage.

Implements Bannerman.

4.35.3.8 decreaseWeapons()

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

decreases the numWeapons variable of the troop by 1.

Implements Bannerman.

4.35.3.9 detach()

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

Detaches a Raven observer to the troop.

Parameters

o	- the Raven observer object to detach

Implements Bannerman.

4.35.3.10 getDamage()

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

Accessor. Returns the damage capacity of the troop.

Returns

damage.

Implements Bannerman.

4.35.3.11 getFavour()

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

Returns the the loyalty favour levels variable of the Troop object.

Returns

favour

Implements Bannerman.

4.35.3.12 getFood()

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

Accessor. Returns the amount of food the troop has.

Returns

numFood.

Implements Bannerman.

4.35.3.13 getHP()

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

Returns the troop's HP.

Returns

HP.

Implements Bannerman.

4.35.3.14 getMedical()

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

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

Returns

numMedical.

Implements Bannerman.

4.35.3.15 getName()

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

Returns the name of this Troop object.

Returns

name

Implements Bannerman.

4.35.3.16 getSize()

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

Returns the troop's size.

Returns

size.

4.35.3.17 getTroops()

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

Bannerman pure virtual method, not used by troop.

Returns

null

Implements Bannerman.

4.35.3.18 getWeapons()

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

Accessor. Returns the number of weapons the troop has.

Returns

numWeapons.

Implements Bannerman.

4.35.3.19 increaseFavour()

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

Increases the loyalty favour level of the troop by 1.

Implements Bannerman.

4.35.3.20 increaseHP()

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

increases the HP of the troop.

Parameters

boost	- The number by which to increase the HP of the troop.

Implements Bannerman.

4.35.3.21 increasePower()

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

increases the damage variables of the troop.

Parameters

boost	- The number by which to increase the damage of the troop.

Implements Bannerman.

4.35.3.22 receiveDamage()

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

increases the damage capacity of the troop.

Parameters

X	- the number by which to increase the damage capacity of the troop.

Implements Bannerman.

4.35.3.23 setFood()

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

Sets the number of food supplies the troop has.

Parameters

numFood	- The new numFood the troop should have.

Implements Bannerman.

4.35.3.24 setMaster()

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

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

Parameters

m	- the new MasterOfCoin mediator to assign to the troop.

Implements Bannerman.

4.35.3.25 setMedical()

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

Sets the number of medical supplies the troop has.

Parameters

numMedical	- The new numMedical the troop should have.

Implements Bannerman.

4.35.3.26 setRaven()

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

Assigns a list of Raven Observers to the troop's ravenList.

Parameters

r	- the list of Raven Observers to attach to the troop.

Implements Bannerman.

4.35.3.27 setWeapons()

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

Sets the number of weapon supplies the troop has.

Parameters

numWeapons	- The new numWeapons the troop should have.

Implements Bannerman.

4.35.4 Member Data Documentation

4.35.4.1 damage

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

The damage capability which the troop has

4.35.4.2 size

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

The number of soldiers in the troop

4.35.4.3 strategy

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

The attack strategy used by the troop in battle

4.35.4.4 warZone

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

The war theatre at which this troop engages in battle

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

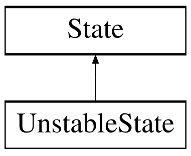
- Troop.h
- Troop.cpp

4.36 UnstableState Class Reference

A concrete state class. A class that is one of the alternative concrete states for Economy of Kingdom.

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

Inheritance diagram for UnstableState:



Public Member Functions

- **UnstableState** ()

  Default constructor. initializes context to null.
- void decreaseCurrency ()

  decreases Economy currency and then checks if the conditions are sufficient for the economy to downgrade to a lower state
- State ∗ getDemotionState ()

  gets lower level state of economy
- string getState ()

  tells what the State of the Economy is.

Additional Inherited Members

4.36.1 Detailed Description

A concrete state class. A class that is one of the alternative concrete states for Economy of Kingdom.

Author

Morgan Bentley

Date

October 2022

4.36.2 Member Function Documentation

4.36.2.1 decreaseCurrency()

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

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

Implements State.

4.36.2.2 getDemotionState()

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

gets lower level state of economy

Returns

concrete State of Economy

Reimplemented from State.

4.36.2.3 getState()

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

tells what the State of the Economy is.

Returns

returns a string that says in text what the State is.

Implements State.

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

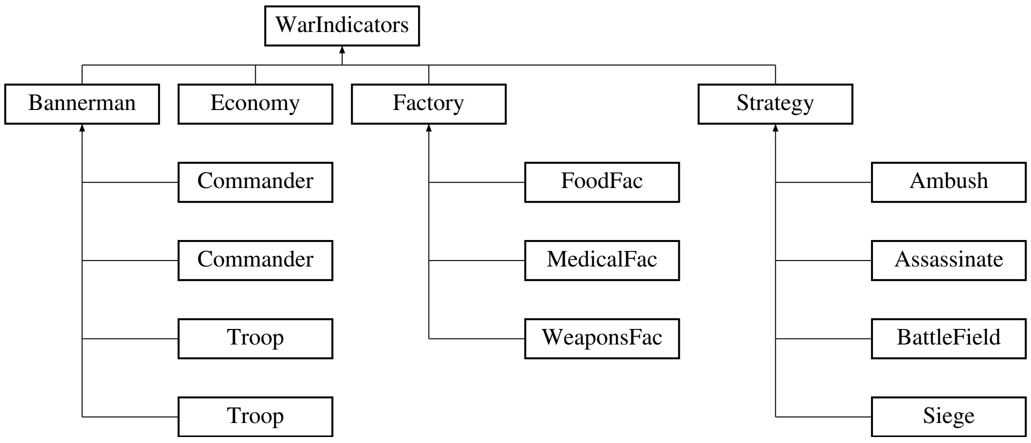
- UnstableState.h
- UnstableState.cpp

4.37 WarIndicators Class Reference

An interface class for all classes that communicate with each other through Treasury mediator.

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

Inheritance diagram for WarIndicators:



Public Member Functions

- **WarIndicators** ()

  Default constructor. initializes m to null.
- WarIndicators (Treasury ∗m)

  constructor. initializes m to the passed in Treasury object.
- void setTreasury (Treasury ∗m)

  sets m (Treasury mediator pointer) to the passed in Treasury pointer.
- virtual ∼**WarIndicators** ()

  destructor. deallocates Treasury pointer of this class

Protected Attributes

- Treasury ∗ m

4.37.1 Detailed Description

An interface class for all classes that communicate with each other through Treasury mediator.

Author

Morgan Bentley

Date

October 2022

4.37.2 Constructor & Destructor Documentation

4.37.2.1 WarIndicators()

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

constructor. initializes m to the passed in Treasury object.

Parameters

m	- mediator which is of Treasury pointer type

4.37.3 Member Function Documentation

4.37.3.1 setTreasury()

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

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

Parameters

m	- mediator which is of Treasury pointer type

4.37.4 Member Data Documentation

4.37.4.1 m

`Treasury* WarIndicators::m [protected]`

Treasury mediator pointer

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

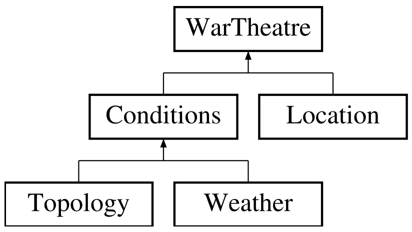
- WarIndicators.h

4.38 WarTheatre Class Reference

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

`#include <WarTheatre.h>`

Inheritance diagram for WarTheatre:



Public Member Functions

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

Private Attributes

- Strategy ∗ strategy
- char venue
- string location
- int difficulty = 1

4.38.1 Detailed Description

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

Author

Keabetswe Mothapo

Date

October 2022

4.38.2 Member Function Documentation

4.38.2.1 decideVenue()

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

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

Parameters

strategy	- a pointer to te strategy object used in the battle

Returns

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

4.38.2.2 getDifficulty()

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

returns the difficulty value of the theatre

Returns

the difficulty value

4.38.2.3 getStrategy()

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

returns the strategy the location is based on

Returns

the strategy pointer

4.38.2.4 getVenue()

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

returns the venue the battle is based in

Returns

the venue variable

4.38.2.5 sendScout()

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

Abstract. Decorates the location of the battle.

Implemented in Conditions, Location, Topology, and Weather.

4.38.2.6 setVenue()

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

sets the venue variable.

Parameters

v	- the char variable used to set the venue.

4.38.3 Member Data Documentation

4.38.3.1 difficulty

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

Integer representing the difficulty of the battle at the decorated venue

4.38.3.2 location

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

The location of the venue as a string

4.38.3.3 strategy

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

Name of Strategy as it determines the venue of the battle

4.38.3.4 venue

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

Char representing the venue. options a to d only

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

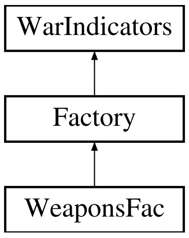
- WarTheatre.h
- WarTheatre.cpp

4.39 WeaponsFac Class Reference

Factory class to serve as a concrete creator class used to create Weapons supplies.

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

Inheritance diagram for WeaponsFac:



Public Member Functions

- ArmySupplies ∗ make ()

  make() is the factory Method called in operation() to make the Weapon supplies.
- ∼**WeaponsFac** ()

  destructor for the WeaponsFac class to delete and free the supply variable.

Additional Inherited Members

4.39.1 Detailed Description

Factory class to serve as a concrete creator class used to create Weapons supplies.

Author

Ronin Brookes 19069686

4.39.2 Member Function Documentation

4.39.2.1 make()

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

make() is the factory Method called in operation() to make the Weapon supplies.

Returns

ArmySupplies pointer to the newly created Weapon supplies.

Implements Factory.

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

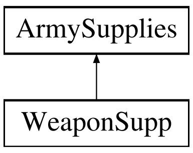
- WeaponsFac.h
- WeaponsFac.cpp

4.40 WeaponSupp Class Reference

Concrete Product Class containing the information on the Weapon supplies.

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

Inheritance diagram for WeaponSupp:



Public Member Functions

- int getAmount ()

  getAmount() is used to return the amount of Weapon supplies.

4.40.1 Detailed Description

Concrete Product Class containing the information on the Weapon supplies.

Author

Ronin Brookes 19069686

4.40.2 Member Function Documentation

4.40.2.1 getAmount()

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

getAmount() is used to return the amount of Weapon supplies.

Returns

an integer representation of the amount of Weapon supplies.

Implements ArmySupplies.

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

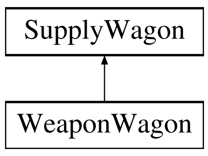
- WeaponSupp.h
- WeaponSupp.cpp

4.41 WeaponWagon Class Reference

Concrete Prototype class to serve as the wagon for Weapons.

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

Inheritance diagram for WeaponWagon:



Public Member Functions

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

Public Attributes

- ArmySupplies * supp

4.41.1 Detailed Description

Concrete Prototype class to serve as the wagon for Weapons.

Author

Ronin Brookes 19069686

4.41.2 Member Function Documentation

4.41.2.1 clone()

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

Clone the Weapon Supply Wagon object.

Returns

the new Weapon Supply Wagon clone.

Implements SupplyWagon.

4.41.2.2 getSupplies()

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

Return the supp variable for the Weapon Wagon.

Returns

returns the supp variable. Returns the necessary supplies.

Implements SupplyWagon.

4.41.2.3 setSup()

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

initialize the supp variable for the WeaponWagon.

Parameters

sup	is used to set the supp variable,

Implements SupplyWagon.

4.41.3 Member Data Documentation

4.41.3.1 supp

`ArmySupplies* WeaponWagon::supp`

pointer to the supplies in the Weapon wagon

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

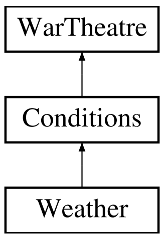
- WeaponWagon.h
- WeaponWagon.cpp

4.42 Weather Class Reference

concrete decorator A. This is the class implements climate effects to the war location. Inherits from Conditions class

`#include <Weather.h>`

Inheritance diagram for Weather:



Public Member Functions

- Weather (WarTheatre *myTheatre)

  Costructor of the weather object.
- int calcEffect ()

  the function that uses the weather components to create a wholistc effect on bannermen
- void setTemp (double t)

  setter for the temperature variable
- void **makeItRain** ()

  setter for the rain variable makes the boolean variable true
- void setWindSpeed (int SP)

  setter for the windspeed variable
- void **weatherReport** ()

  displays the weather conditions of the location
- void sendScout ()

  calls the calEffect function and affects bannermen

Private Attributes

- double temp
- bool rain
- int windspeed

4.42.1 Detailed Description

concrete decorator A. This is the class implements climate effects to the war location. Inherits from Conditions class

Author

Keabetswe Mothapo

Date

October 2022

4.42.2 Constructor & Destructor Documentation

4.42.2.1 Weather()

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

Costructor of the weather object.

Parameters

val	- recieves a generated value which will be used for setters in this class

4.42.3 Member Function Documentation

4.42.3.1 calcEffect()

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

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

Returns

returns the adjusted difficulty value

4.42.3.2 sendScout()

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

calls the calEffect function and affects bannermen

Reimplemented from Conditions.

4.42.3.3 setTemp()

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

setter for the temperature variable

Parameters

t	- the value used to set the temp

4.42.3.4 setWindSpeed()

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

setter for the windspeed variable

Parameters

SP	- the value used to set the wind speed

4.42.4 Member Data Documentation

4.42.4.1 rain

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

Boolean to indicate whether it is raining or not

4.42.4.2 temp

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

Value of the temperature at the venue

4.42.4.3 windspeed

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

Value of the wind speed at the venue

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

- Weather.h
- Weather.cpp

Chapter 5

File Documentation

5.1 Ambush.h

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

class Ambush :public Strategy {
private:
    int stealth;
public:
    Ambush(int stealth,Kingdom* myKingdom,Kingdom* enemyKingdom,Bannerman* myBannerman, Bannerman*
   enemyBannerman,string name,int min,int minFavour, Historian* h, HistoryBook* hb);
    bool attack(Bannerman* myBannerman, Bannerman* enemyBannerman);
    ~Ambush();
};

#endif
```

5.2 ArmySupplies.h

```
#ifndef ARMYSUPPLIES_H
#define ARMYSUPPLIES_H

class ArmySupplies {

public:

    virtual int getAmount() = 0;
};

#endif
```

5.3 Assassinate.h

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

class Assassinate :public Strategy {

private:
    int stealth;
    bool alive;
public:
    Assassinate(int stealth, bool alive,Kingdom* myKingdom,Kingdom* enemyKingdom,Bannerman* myBannerman,
   Bannerman* enemyBannerman,string name,int min,int minFavour, Historian* h, HistoryBook* hb);
    bool attack(Bannerman* myBannerman, Bannerman* enemyBannerman);
    ~Assassinate();
};

#endif
```

5.4 Bannerman.h

```
#ifndef BANNERMAN_H
#define BANNERMAN_H
#include "Strategy.h"
#include "MasterOfCoin.h"
#include "Raven.h"
#include "WarIndicators.h"
#include "WarTheatre.h"
#include <string>
#include <list>

class MasterOfCoin;
class Strategy;
class Bannerman:  public WarIndicators {


protected:

    string name;

    int favour;

    int numWeapons;

    int damage;

    int numFood;

    int numMedical;

    list<Raven*> ravenList;


    MasterOfCoin* m;

    Strategy* strategy;

    int HP;


public:
    bool assassin;
    Bannerman();

    virtual void increaseFavour() = 0;

    virtual void decreaseFavour() = 0;

    virtual void attach(Raven* o)=0;

    virtual void detach(Raven* o)=0;

    virtual void increaseHP(int boost) = 0;//implement

    virtual void changeStrategy(Strategy* strategy) = 0;

    virtual void attack(Bannerman* myBannerman, Bannerman* enemyBannerman) = 0;

    virtual void increasePower(int boost) = 0;

     virtual string getName() = 0;

     virtual int getHP() = 0;

     virtual int getDamage() = 0;


     virtual void receiveDamage(int boost) = 0;

     virtual void decreasePower(int x) = 0;

     virtual void decreaseWeapons() = 0;

     virtual void decreaseFood() = 0;

     virtual void decreaseMedical() = 0;

     virtual int getWeapons() = 0;

     virtual int getFood() = 0;

     virtual int getMedical() = 0;


     virtual void setWeapons(int numWeapons) = 0;

    virtual void setFood(int numFood) = 0;

    virtual void setMedical(int numMedical) = 0;

    virtual void setRaven(list<Raven*> r) = 0;

    virtual void setMaster(MasterOfCoin* m) = 0;

    virtual int getFavour() = 0;
    ~Bannerman();

    virtual list<Bannerman*> getTroops() = 0;
};

#endif
```

5.5 BattleField.h

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

class BattleField :public Strategy {

public:
    BattleField(Kingdom* myKingdom,Kingdom* enemyKingdom,Bannerman* myBannerman, Bannerman*
   enemyBannerman,string name,int min,int minFavour, Historian* h, HistoryBook* hb);
    bool attack(Bannerman* myBannerman, Bannerman* enemyBannerman);
    ~BattleField();
};

#endif
```

5.6 Class.h

```
#ifndef CLASS_H
#define CLASS_H

class Class {
};

#endif
```

5.7 Commander.h

```
#ifndef COMMANDER_H
#define COMMANDER_H

#include "Bannerman.h"
#include "ConIterator.h"
#include <list>
class Iterator;
using namespace std;

class Commander :  public Bannerman
{
private:
    list<Bannerman *> groundForces;

    string name;

public:
    Commander(string name);

    Iterator *createIterator();

    void removeBannerman(Bannerman *x);

    void attack(Bannerman *myBannerman, Bannerman *enemyBannerman) override;

    void addBannerman(Bannerman *b);

    list<Bannerman *> getTroops();

    int getHP() override;

    int getDamage() override;
    void attach(Raven *o) override;

    void detach(Raven *o) override;

    void receiveDamage(int x) override;
    void decreaseWeapons() override;
    void decreaseFood() override;
    void decreaseMedical() override;

    void increasePower(int boost) override;

     int getWeapons() override;

     int getFood() override;

     int getMedical() override;

     void increaseHP(int boost) override;

     void changeStrategy(Strategy *strategy) override;

     void increaseFavour() override;

     void decreaseFavour() override;

     int getFavour() override;


     void setWeapons(int numWeapons) override;

     void setFood(int numFood) override;

     void setMedical(int numMedical) override;

     void decreasePower(int x) override;

     ~Commander();

     // Julianna added:

     void setRaven(list<Raven *> r) override;

     void setMaster(MasterOfCoin *m) override;

     void setStrategy(Strategy *s);

string getName();
};

#endif
```

5.8 Conditions.h

```
#ifndef CONDITIONS_H
#define CONDITIONS_H
#include <cstdlib>
#include "WarTheatre.h"
//#include "Weather.h"
//#include "Topology.h"
#include "Location.h"

class Conditions :  public  WarTheatre {

private:
    WarTheatre* myVenue;

public:

    void sendScout() ;

    Conditions(WarTheatre* myVenue);

    virtual ~Conditions();
};

#endif
```

5.9 ConIterator.h

```
#ifndef CONITERATOR_H
#define CONITERATOR_H
#include "Bannerman.h"
#include "Iterator.h"
#include <list>
using namespace std;

class ConIterator :  public Iterator {
private:
    list<Bannerman*> armyList;

    list<Bannerman*>::iterator it;
public:
    ConIterator(list<Bannerman*> X);

    Bannerman* Current();


    Bannerman* next();

    bool hasNext();

    bool isActive();
};

#endif
```

5.10 Economy.h

```
#ifndef ECONOMY_H
#define ECONOMY_H
#include "WarIndicators.h"

#include <iostream>
#include "State.h"

using namespace std;

class State;
class Economy :public WarIndicators {

private:
    State* state;
    int currency;

public:
    Economy(State* state,int currency);
    void SetState();
    State* getState();
    void decreaseCurrency();
    int getCurrency();
    void removeCurrency(int i);
    virtual ~Economy();
};
#endif
```

5.11 Factory.h

```

#ifndef FACTORY_H
#define FACTORY_H
#include "WarIndicators.h"
#include "ArmySupplies.h"
#include <iostream>
using namespace std;
class Factory :  public WarIndicators {

protected:
    virtual ArmySupplies* make() = 0;
public:
    void operation();
    ArmySupplies* getSupply();
    virtual ~Factory();
private:
    ArmySupplies* supply;
};

#endif
```

5.12 FailedState.h

```
#ifndef FAILEDSTATE_H
#define FAILEDSTATE_H
#include "State.h"
#include "Economy.h"

class FailedState :public State {


public:
    FailedState();
    void decreaseCurrency();
    string getState();
};

#endif
```

5.13 FoodFac.h

```
#ifndef FOODFAC_H
#define FOODFAC_H
#include "Factory.h"
#include "ArmySupplies.h"
#include "FoodSupp.h"

class FoodFac :  public Factory {


public:
    ArmySupplies* make();
    ~FoodFac();
};

#endif
```

5.14 FoodSupp.h

```
#ifndef FOODSUPP_H
#define FOODSUPP_H
#include "ArmySupplies.h"

class FoodSupp :  public ArmySupplies {

public:
    int getAmount();
};

#endif

```

5.15 FoodWagon.h

```
#ifndef FOODWAGON_H
#define FOODWAGON_H
#include "Factory.h"
#include "SupplyWagon.h"
#include "ArmySupplies.h"
#include "FoodSupp.h"

class FoodWagon :  public SupplyWagon {

public:
    void setSup(ArmySupplies* sup);
    SupplyWagon* clone();
    ArmySupplies* getSupplies();
    ArmySupplies* supp=NULL;
    ~FoodWagon();
};

#endif
```

5.16 HealthyState.h

```
#ifndef HEALTHYSTATE_H
#define HEALTHYSTATE_H
#include "State.h"
#include "Economy.h"
#include "UnstableState.h"

class HealthyState :public State {


public:
    HealthyState();
    void decreaseCurrency();
    virtual State* getDemotionState();
    string getState();
};

#endif
```

5.17 Historian.h

```
#ifndef HISTORIAN_H
#define HISTORIAN_H
#include "Bannerman.h"
#include "History.h"
class History;
class Historian {
private:
    Bannerman* bannerman;

 public:
    void setAlly(Bannerman* b);

    History* Store();

    Bannerman* restoreAlly(History* h);

    ~Historian(){};
};

#endif
```

5.18 History.h

```
#ifndef HISTORY_H
#define HISTORY_H
#include "Bannerman.h"
#include <iostream>
using namespace std;
class Bannerman;
class History { //memento
private:
    Bannerman* bannerman;

public:
    History(Bannerman* b);

    Bannerman* getBannerman();

    ~History(){};
};

#endif
```

5.19 HistoryBook.h

```
#ifndef HISTORYBOOK_H
#define HISTORYBOOK_H
#include "History.h"
#include <list>
class History;
class HistoryBook {
private:
    list<History*> defectedAllies;
public:
    void add(History* h);

    History* getAlly();
};

#endif
```

5.20 Iterator.h

```
#ifndef ITERATOR_H
#define ITERATOR_H
#include "Bannerman.h"

class Iterator {


public:
    virtual Bannerman* Current()=0;
    virtual bool hasNext() = 0;

    virtual Bannerman* next() = 0;

    virtual bool isActive() = 0;
};

#endif
```

5.21 Kingdom.h

```
#ifndef KINGDOM_H
#define KINGDOM_H

#include <list>
#include "Bannerman.h"
#include "Economy.h"

using namespace std;

class Bannerman;

class Kingdom {

private:
    list<Bannerman*> bannerman;
    Economy* economy;

public:
    Kingdom(Economy* economy);
    void remove(Bannerman* b);
    void add(Bannerman* b);
    virtual ~Kingdom();
    int getSize();
    Bannerman* getAlly(string n);
    list<Bannerman*> getKingdom();
};

#endif
```

5.22 Location.h

```
#ifndef LOCATION_H
#define LOCATION_H
#include <iostream>
#include "WarTheatre.h"
#include "Conditions.h" //causes circular dependency?

using namespace std;

class Location :  public WarTheatre {


public:

    Location();

    void sendScout();


};

#endif
```

5.23 MasterOfCoin.h

```
#ifndef MASTEROFCOIN_H
#define MASTEROFCOIN_H

#include "Economy.h"
#include "Raven.h"
#include "Strategy.h"
#include "Treasury.h"
#include "WarIndicators.h"

class Strategy;
class MasterOfCoin :  public Treasury {

private:
    Economy* economy;

    Raven* observer;

    Strategy* strategy;

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

    void notify(WarIndicators* sender);

    void decreaseCurrency();

    void manufacture();

    ~MasterOfCoin();
};

#endif
```

5.24 MedicalFac.h

```
#ifndef MEDICALFAC_H
#define MEDICALFAC_H
#include "Factory.h"
#include "ArmySupplies.h"
#include "MedicalSupp.h"

class MedicalFac :  public Factory {

public:
    ArmySupplies* make();
    ~MedicalFac();
};

#endif

```

5.25 MedicalSupp.h

```
#ifndef MEDICALSUPP_H
#define MEDICALSUPP_H
#include "ArmySupplies.h"

class MedicalSupp :  public ArmySupplies {


public:
    int getAmount();
};

#endif
```

5.26 MedicalWagon.h

```
#ifndef MEDICALWAGON_H
#define MEDICALWAGON_H
#include "Factory.h"
#include "SupplyWagon.h"
#include "ArmySupplies.h"
#include "MedicalSupp.h"

class MedicalWagon :  public SupplyWagon {

public:
    void setSup(ArmySupplies* sup);
    SupplyWagon* clone();
    ArmySupplies* getSupplies();
    ArmySupplies* supp;
    ~MedicalWagon();
};

#endif
```

5.27 Raven.h

```
#ifndef RAVEN_H
#define RAVEN_H

class Raven {


public:
    virtual void update() = 0;
};

#endif
```

5.28 sendRaven.h

```
#ifndef SENDRAVEN_H
#define SENDRAVEN_H

#include "SupplyWagon.h"
#include "Commander.h"
#include "MasterOfCoin.h"
#include "Raven.h"
#include "Bannerman.h"


class sendRaven :  public Raven {

private:

    int numFood;

    int numMedical;

    int numWeapons;

    SupplyWagon** supplies;

    Bannerman* subject;

public:
    sendRaven(SupplyWagon** supplies, Bannerman* subject);

    void update();

    void checkSupplies();

    ~sendRaven();
};

#endif
```

5.29 Siege.h

```
#ifndef SIEGE_H
```

```
#define SIEGE_H
#include "Strategy.h"

class Siege :public Strategy {
private:
    int stealth;
public:
    Siege(int stealth,Kingdom* myKingdom,Kingdom* enemyKingdom,Bannerman* myBannerman, Bannerman*
   enemyBannerman,string name,int min,int minFavour, Historian* h, HistoryBook* hb);
    bool attack(Bannerman* myBannerman, Bannerman* enemyBannerman);
    ~Siege();
};

#endif
```

5.30 State.h

```
#ifndef STATE_H
#define STATE_H
#include "Economy.h"
class Economy;
using namespace std;

class State {

protected:
    Economy* context;

public:
    State();
    virtual void setContext(Economy* context);
    virtual void decreaseCurrency()=0;
    virtual State* getDemotionState();
    virtual ~State();
    virtual string getState()=0;
};

#endif
```

5.31 Strategy.h

```
#ifndef STRATEGY_H
#define STRATEGY_H

#include <string>
#include <ctime>
#include "Bannerman.h"
#include "WarIndicators.h"
#include "MasterOfCoin.h"
#include "Kingdom.h"
#include "HistoryBook.h"
#include "Historian.h"

using namespace std;
class Historian;
class HistoryBook;
class Bannerman;
class Kingdom;
class Strategy :  public WarIndicators  {

protected:
    Kingdom* myKingdom;
    Kingdom* enemyKingdom;
    Bannerman* myBannerman;
    Bannerman* enemyBannerman;
    string strategy;
    int minFavour;
    int minSupplies;
    HistoryBook* BookOfDura;
    Historian* Greg;
    int defectedAllies;
public:
    Strategy(Kingdom* myKingdom,Kingdom* enemyKingdom,Bannerman* myBannerman, Bannerman*
   enemyBannerman,string name,int min,int minFavour, Historian* h, HistoryBook* hb);
    virtual bool attack(Bannerman* myBannerman, Bannerman* enemyBannerman)=0;
    virtual string getStrategyName();
    virtual Bannerman* getMyBannerman();
    virtual Bannerman* getEnemyBannerman();
    virtual ~Strategy();
};
#endif
```

5.32 SupplyWagon.h

```
#ifndef SUPPLYWAGON_H
#define SUPPLYWAGON_H
#include "ArmySupplies.h"

class SupplyWagon {

public:

   virtual void setSup(ArmySupplies* sup)=0;
    virtual SupplyWagon* clone() = 0;
    virtual ArmySupplies* getSupplies()=0;
    ArmySupplies* supp;
    //virtual ~SupplyWagon();

};

#endif
```

5.33 Topology.h

```
#ifndef TOPOLOGY_H
#define TOPOLOGY_H
#include "Conditions.h"

class Topology :  public Conditions {

public:

    void setTopology();

    Topology(WarTheatre* myTheatre);

    void sendScout();
};

#endif
```

5.34 Treasury.h

```
#ifndef TREASURY_H
#define TREASURY_H
#include "WarIndicators.h"
class WarIndicators;
class Treasury {


public:
    virtual void notify(WarIndicators* sender) = 0;
};

#endif
```

5.35 Troop.h

```
#ifndef TROOP_H
#define TROOP_H
#include "Kingdom.h"
#include "Strategy.h"
#include "Bannerman.h"

class Troop :  public Bannerman {

private:

    int size;

    Strategy* strategy;

    int damage;

    WarTheatre* warZone;
public:
    Troop(string name, int favor, int numFood, int Medical, int
   HP,WarTheatre*warZone,Strategy*strategy,MasterOfCoin* m, bool assassin, int size);

    int getHP() override;

    int getSize();

    void attach(Raven* o) override;

    void detach(Raven* o) override;

    void increasePower(int boost) override;

    void attack(Bannerman* myBannerman, Bannerman* enemyBannerman) override;

    void decreaseWeapons() override;

    void decreaseFood() override;

    void decreaseMedical() override;

    void changeStrategy(Strategy* strategy) override;

    void increaseFavour() override;

    void decreaseFavour() override;

    int getFavour() override;

    void increaseHP(int boost) override;
    int getDamage() override;

    void receiveDamage(int X) override;

    int getWeapons() override;

    int getFood() override;

    int getMedical() override;

    void setWeapons(int numWeapons) override;

    void setFood(int numFood) override;



    void decreasePower(int x) override;


    void setMedical(int numMedical) override;


    void setRaven(list<Raven*> r) override;

    void setMaster(MasterOfCoin* m) override;

    ~Troop();
    string getName() override;
    list<Bannerman*> getTroops() override;
};

#endif
```

5.36 UnstableState.h

```
#ifndef UNSTABLESTATE_H
#define UNSTABLESTATE_H
#include "State.h"
#include "Economy.h"
#include "FailedState.h"

class UnstableState :public State {


public:
    UnstableState();
    void decreaseCurrency();
    State* getDemotionState();
    string getState();
};

#endif
```

5.37 WarIndicators.h

```
#ifndef WARINDICATORS_H
#define WARINDICATORS_H
#include "Treasury.h"
class Treasury;
class WarIndicators {

protected:
    Treasury* m;

public:
    WarIndicators(){
        m = nullptr;
    }
    WarIndicators(Treasury* m){
        this->m=m;
    }
    void setTreasury(Treasury* m){
        this->m=m;
    }

    virtual ~WarIndicators(){
        //delete m;
    }


};

#endif
```

5.38 WarTheatre.h

```
#ifndef WARTHEATRE_H
#define WARTHEATRE_H
#include "Strategy.h"
#include <string>
#include <iostream>

using namespace std;

class Strategy;
class WarTheatre {

private:
    Strategy* strategy;

    char venue;

    string location;

    int difficulty = 1;


public:
    WarTheatre();

    Strategy* getStrategy(); //may need to make sendScout the virtual one instead

    virtual void sendScout() = 0;

    char decideVenue(Strategy* strategy);

    char getVenue();

    void setStrategy(Strategy* myStrat);

    void setLocation(string loc);

    void setDifficulty(int num);

    int getDifficulty();
    void setVenue(char v);
};

#endif
```

5.39 WeaponsFac.h

```
#ifndef WEAPONSFAC_H
```

```
#define WEAPONSFAC_H
#include "Factory.h"
#include "ArmySupplies.h"
#include "WeaponSupp.h"

class WeaponsFac :  public Factory {

public:

    ArmySupplies* make();

    ~WeaponsFac();
};

#endif
```

5.40 WeaponSupp.h

```
#ifndef WEAPONSUPP_H
#define WEAPONSUPP_H
#include "ArmySupplies.h"

class WeaponSupp :  public ArmySupplies {

public:
    int getAmount();
};

#endif
```

5.41 WeaponWagon.h

```
#ifndef WEAPONWAGON_H
#define WEAPONWAGON_H
#include "Factory.h"
#include "SupplyWagon.h"
#include "ArmySupplies.h"
#include "WeaponSupp.h"

class WeaponWagon :  public SupplyWagon {

public:

    void setSup(ArmySupplies* sup);
    SupplyWagon* clone();
    ArmySupplies* getSupplies();
    ArmySupplies* supp;
    ~WeaponWagon();
};

#endif
```

5.42 Weather.h

```
#ifndef WEATHER_H
#define WEATHER_H
#include "Conditions.h"
#include <ctime>

class Weather:public Conditions{

private:
    double temp;

    bool rain;

    int windspeed;

public:
    Weather(WarTheatre* myTheatre);

    int calcEffect();

    void setTemp(double t);

    void makeItRain();

    void setWindSpeed(int SP);

    void weatherReport(); //ask Jules

    void sendScout();
};

#endif
```

Index