COS214PROJECT

1st

Generated by Doxygen 1.9.5

Chapter 1

Todo List

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

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

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

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

Member Ambush::~Ambush ()

destructor.

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

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

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

makes assassin of attacking Bannerman try to kill enemyBannerman

Member Assassinate::~Assassinate ()

destructor.

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

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

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

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

Member BattleField::~BattleField ()

destructor.

Member FailedState::decreaseCurrency ()

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

Member HealthyState::decreaseCurrency ()

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

Member HealthyState::getDemotionState ()

gets lower level state of economy

Member Kingdom::~Kingdom ()

destructor. deallocates all pointers of this class

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

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

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

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

Member Siege::~Siege ()

destructor.

Member State::decreaseCurrency ()=0

Abstract method.

Member State::getDemotionState ()

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

Member State::setContext (Economy *context)

sets context variable.

Member State::~State ()

destructor. deallocates context

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

Abstract method

Member Strategy::getEnemyBannerman ()

gets Bannerman being attacked

Member Strategy::getMyBannerman ()

gets attacking Bannerman

Member Strategy::getStrategyName ()

gets strategy variable

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

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

Member Strategy::~Strategy ()

destructor. deallocates all pointers of this class

Member UnstableState::decreaseCurrency ()

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

Member UnstableState::getDemotionState ()

gets lower level state of economy

Member WarIndicators::~WarIndicators ()

destructor. deallocates Treasury pointer of this class

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

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

Chapter 5

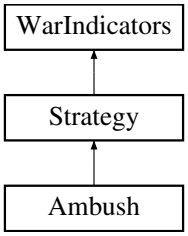
Class Documentation

5.1 Ambush Class Reference

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

`#include <Ambush.h>`

Inheritance diagram for Ambush:



Public Member Functions

- Ambush (int stealth, Kingdom ∗myKingdom, Kingdom ∗enemyKingdom, Bannerman ∗myBannerman, Bannerman ∗enemyBannerman, string name, int min, int minFavour)
- bool attack (Bannerman ∗myBannerman, Bannerman ∗enemyBannerman)
- ∼Ambush ()

Private Attributes

- int stealth

Additional Inherited Members

5.1.1 Detailed Description

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

Author

Morgan Bentley

Date

October 2022

5.1.2 Constructor & Destructor Documentation

5.1.2.1 Ambush()

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

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

Parameters

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

5.1.2.2 ~Ambush()

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

Todo destructor.

5.1.3 Member Function Documentation

5.1.3.1 attack()

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

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

Parameters

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.

5.1.4 Member Data Documentation

5.1.4.1 stealth

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

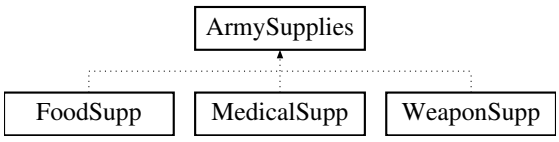
skill level of Bannerman using this strategy

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

- Ambush.h
- Ambush.cpp

5.2 ArmySupplies Class Reference

Inheritance diagram for ArmySupplies:



Public Member Functions

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

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

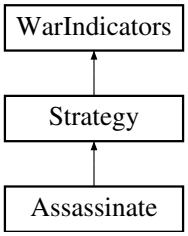
- ArmySupplies.h

5.3 Assassinate Class Reference

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

`#include <Assassinate.h>`

Inheritance diagram for Assassinate:



Public Member Functions

- Assassinate (int stealth, bool alive, Kingdom ∗myKingdom, Kingdom ∗enemyKingdom, Bannerman ∗myBannerman, Bannerman ∗enemyBannerman, string name, int min, int minFavour)
- bool attack (Bannerman ∗myBannerman, Bannerman ∗enemyBannerman)
- ∼Assassinate ()

Private Attributes

- int stealth
- bool alive

Additional Inherited Members

5.3.1 Detailed Description

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

Author

Morgan Bentley

Date

October 2022

5.3.2 Constructor & Destructor Documentation

5.3.2.1 Assassinate()

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

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

Parameters

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

5.3.2.2 ~Assassinate()

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

Todo destructor.

5.3.3 Member Function Documentation

5.3.3.1 attack()

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

Todo makes assassin of attacking Bannerman try to kill enemyBannerman

Parameters

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.

5.3.4 Member Data Documentation

5.3.4.1 alive

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

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

5.3.4.2 stealth

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

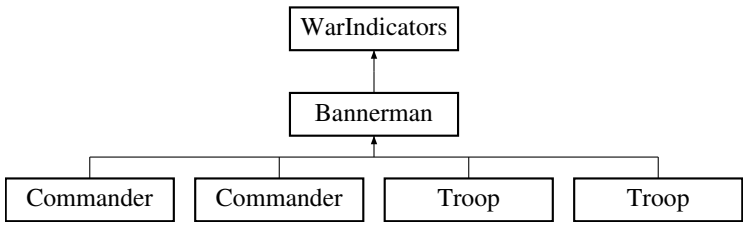
skill level of Assassin from Bannerman using this strategy

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

- Assassinate.h
- Assassinate.cpp

5.4 Bannerman Class Reference

Inheritance diagram for Bannerman:



Public Member Functions

- **Bannerman** ()

  Default constructor.
- virtual void increaseFavour ()=0

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

- virtual int getFavour ()=0

  Abstract. returns the favour of the bannerman.

- ~**Bannerman** ()

  Default destructor.

Protected Attributes

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

5.4.1 Member Function Documentation

5.4.1.1 attach()

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

Abstract. Attaches a Raven Observer on the bannerman.

Parameters

o	- the Raven Observer to attach.

Implemented in Commander, and Troop.

5.4.1.2 attack()

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

Abstract. uses strategy to attack another kingdom.

Parameters

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

Implemented in Commander, and Troop.

5.4.1.3 changeStrategy()

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

Abstract. changes the attack strategy of the bannerman.

Parameters

strategy	- the new attack strategy.

Implemented in Commander, and Troop.

5.4.1.4 decreaseFavour()

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

Abstract. decreases the favour of the bannerman.

Implemented in Commander, and Troop.

5.4.1.5 decreaseFood()

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

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

Implemented in Commander, and Troop.

5.4.1.6 decreaseMedical()

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

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

Implemented in Commander, and Troop.

5.4.1.7 decreasePower()

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

Abstract. decreases the damage capability of the bannerman.

Parameters

x	- the number by which to decrease damage.

Implemented in Commander, and Troop.

5.4.1.8 decreaseWeapons()

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

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

Implemented in Commander, and Troop.

5.4.1.9 detach()

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

Abstract. detaches a Raven Observer from the bannerman.

Parameters

o	- the Raven Observer to detach.

Implemented in Commander, and Troop.

5.4.1.10 getDamage()

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

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

Returns

The damage of the bannerman.

Implemented in Commander, and Troop.

5.4.1.11 getFavour()

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

Abstract. returns the favour of the bannerman.

Implemented in Commander, and Troop.

5.4.1.12 getFood()

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

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

Returns

The numFood of the bannerman.

Implemented in Commander, and Troop.

5.4.1.13 getHP()

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

Abstract. Accessor that returns the name of the component.

Returns

The name of the bannerman.

Implemented in Commander, and Troop.

5.4.1.14 getMedical()

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

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

Returns

The numMedical of the bannerman.

Implemented in Commander, and Troop.

5.4.1.15 getName()

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

Accessor that returns the name of the bannerman.

Returns

The name of the bannerman.

5.4.1.16 getWeapons()

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

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

Returns

The numWeapons of the bannerman.

Implemented in Commander, and Troop.

5.4.1.17 increaseFavour()

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

Abstract. increases the favour of the bannerman.

Implemented in Commander, and Troop.

5.4.1.18 increaseHP()

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

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

Parameters

boost	- the number by which to increase HP.

Implemented in Commander, and Troop.

5.4.1.19 increasePower()

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

Abstract. increases the damage capability of the bannerman.

Parameters

boost	- the number by which to increase damage.

Implemented in Commander, and Troop.

5.4.1.20 receiveDamage()

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

Abstract. Increases the damage capability of the bannerman.

Parameters

boost	- the number by which to increase damage

Implemented in Commander, and Troop.

5.4.1.21 setFood()

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

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

Parameters

numFood	- The new numFood the component should have

Implemented in Commander, and Troop.

5.4.1.22 setMaster()

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

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

Parameters

m	- the new MasterOfCoin mediator.

Implemented in Commander, and Troop.

5.4.1.23 setMedical()

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

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

Parameters

numMedical	- The new numMedical the component should have

Implemented in Commander, and Troop.

5.4.1.24 setRaven()

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

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

Parameters

r	- the list of Raven Observer to attach.

Implemented in Commander, and Troop.

5.4.1.25 setWeapons()

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

Abstract. Sets the number of weapons of the bannerman.

Parameters

numWeapons	- The new name the numWeapons should have

Implemented in Commander, and Troop.

5.4.2 Member Data Documentation

5.4.2.1 assassin

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

Indicates whether or not the bannerman is an assassin

5.4.2.2 damage

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

Damage capability the bannerman has

5.4.2.3 favour

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

Amount of favour the bannerman has

5.4.2.4 HP

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

The health points of the bannerman

5.4.2.5 m

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

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

5.4.2.6 name

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

Name of the bannerman

5.4.2.7 numFood

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

Amount of food the bannerman has

5.4.2.8 numMedical

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

Amount of medical supplies the bannerman has

5.4.2.9 numWeapons

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

Number of weapons the bannerman has

5.4.2.10 ravenList

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

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

5.4.2.11 strategy

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

The attack strategy the bannerman uses

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

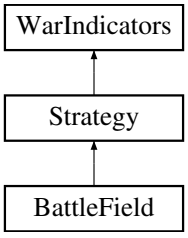
- Bannerman.h
- Bannerman.cpp

5.5 BattleField Class Reference

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

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

Inheritance diagram for BattleField:



Public Member Functions

- BattleField (Kingdom *myKingdom, Kingdom *enemyKingdom, Bannerman *myBannerman, Bannerman *enemyBannerman, string name, int min, int minFavour)
- bool attack (Bannerman *myBannerman, Bannerman *enemyBannerman)
- ~BattleField ()

Additional Inherited Members

5.5.1 Detailed Description

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

Author

Morgan Bentley

Date

October 2022

5.5.2 Constructor & Destructor Documentation

5.5.2.1 BattleField()

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

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

Parameters

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

5.5.2.2 ~BattleField()

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

Todo destructor.

5.5.3 Member Function Documentation

5.5.3.1 attack()

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

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

Parameters

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

5.6 Class Class Reference

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

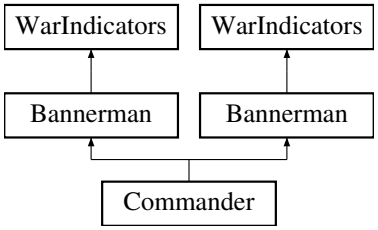
- Class.h

5.7 Commander Class Reference

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

`#include <Commander.h>`

Inheritance diagram for Commander:



Public Member Functions

- Commander (string name)

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

  creates a 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 ()

  Abstract. returns the favour of the bannerman.
- void setWeapons (int numWeapons) override

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

  Sets the number of food supplies of the bannerman objects in groundForces.
- void setMedical (int numMedical) override

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

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

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

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

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

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

Private Attributes

- list< Bannerman * > **groundForces**

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

Additional Inherited Members

5.7.1 Detailed Description

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

Author

Thapelo Thoka

Date

October 2022

5.7.2 Constructor & Destructor Documentation

5.7.2.1 Commander()

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

Commander Constructor which takes in the name of a commander.

Parameters

name	- name of commander

5.7.3 Member Function Documentation

5.7.3.1 attach()

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

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

Parameters

o	- the Raven observer object

Implements Bannerman.

5.7.3.2 attack()

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

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

Parameters

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

Implements Bannerman.

5.7.3.3 changeStrategy()

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

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

Parameters

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

Implements Bannerman.

5.7.3.4 decreaseFavour()

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

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

Implements Bannerman.

5.7.3.5 decreaseFood()

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

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

Implements Bannerman.

5.7.3.6 decreaseMedical()

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

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

Implements Bannerman.

5.7.3.7 decreasePower()

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

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

Parameters

x	- the number by which to decrease damage.

Implements Bannerman.

5.7.3.8 decreaseWeapons()

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

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

Implements Bannerman.

5.7.3.9 detach()

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

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

Parameters

o	- the Raven observer object

Implements Bannerman.

5.7.3.10 getDamage()

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

Accessor. Returns the damage.

Returns

The total groundForces damage.

Implements Bannerman.

5.7.3.11 getFavour()

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

Abstract. returns the favour of the bannerman.

Implements Bannerman.

5.7.3.12 getFood()

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

Accessor. Returns the total groundForces amount of food.

Returns

numFood.

Implements Bannerman.

5.7.3.13 getHP()

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

Accessor. Returns the HP.

Returns

The total groundForces HP.

Implements Bannerman.

5.7.3.14 getMedical()

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

Accessor. Returns the total groundForces amount of medical supplies.

Returns

numMedical.

Implements Bannerman.

5.7.3.15 getTroops()

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

Returns the groundForces bannerman list

5.7.3.16 getWeapons()

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

Accessor. Returns numWeapons.

Returns

The total groundForces number of Weapons.

Implements Bannerman.

5.7.3.17 increaseFavour()

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

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

Implements Bannerman.

5.7.3.18 increaseHP()

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

increases the HP variables of the bannerman objects in groundForces.

Parameters

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

Implements Bannerman.

5.7.3.19 increasePower()

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

increases the damage variables of the bannerman objects in groundForces.

Parameters

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

Implements Bannerman.

5.7.3.20 receiveDamage()

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

increases the damage variables of the groundForces by x

Parameters

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

Implements Bannerman.

5.7.3.21 setFood()

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

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

Parameters

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

Implements Bannerman.

5.7.3.22 setMaster()

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

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

Parameters

m	- the new MasterOfCoin mediator.

Implements Bannerman.

5.7.3.23 setMedical()

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

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

Parameters

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

Implements Bannerman.

5.7.3.24 setRaven()

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

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

Parameters

r	- the Raven observer list to attach

Implements Bannerman.

5.7.3.25 setStrategy()

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

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

Parameters

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

5.7.3.26 setWeapons()

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

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

Parameters

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

Implements Bannerman.

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

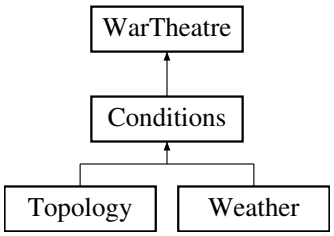
- Commander.h
- Commander.cpp

5.8 Conditions Class Reference

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

```
#include <Conditions.h>
```

Inheritance diagram for Conditions:



Public Member Functions

- void sendScout ()

  Abstract. Decorates the location of the battle.
- Conditions (WarTheatre *myVenue)

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

Public Attributes

- WarTheatre * myVenue

5.8.1 Detailed Description

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

Author

Keabetswe Mothapo

Date

October 2022

5.8.2 Constructor & Destructor Documentation

5.8.2.1 Conditions()

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

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

Parameters

myVenue	- pointer to the object that is to be decoraed

5.8.3 Member Function Documentation

5.8.3.1 sendScout()

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

Abstract. Decorates the location of the battle.

Implements WarTheatre.

Reimplemented in Topology.

5.8.4 Member Data Documentation

5.8.4.1 myVenue

```
WarTheatre* Conditions::myVenue
```

Pointer to the decorated component

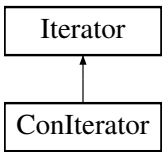
The documentation for this class was generated from the following files:

- Conditions.h
- Conditions.cpp

5.9 ConIterator Class Reference

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

Inheritance diagram for ConIterator:



Public Member Functions

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

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

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

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

Private Attributes

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

5.9.1 Detailed Description

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

Author

Thapelo Thoka

Date

October 2022

5.9.2 Constructor & Destructor Documentation

5.9.2.1 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.

5.9.3 Member Function Documentation

5.9.3.1 Current()

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

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

Returns

Bannerman∗

Implements Iterator.

5.9.3.2 hasNext()

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

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

Returns

false

true

Test

Implements Iterator.

5.9.3.3 isActive()

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

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

Returns

true
false

Test

Implements Iterator.

5.9.3.4 next()

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

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

Returns

Bannerman*

Test

Implements Iterator.

5.9.4 Member Data Documentation

5.9.4.1 armyList

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

The list of bannerman objects to traverse

5.9.4.2 it

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

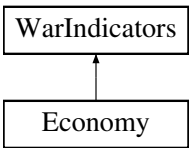
An iterator which is a pointer to the current bannerman object in Commander's groundForces being accessed, primative operations can be performed using this pointer.

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

- ConIterator.h
- ConIterator.cpp

5.10 Economy Class Reference

Inheritance diagram for Economy:



Public Member Functions

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

Private Attributes

- State * state
- int **currency**

Additional Inherited Members

5.10.1 Member Data Documentation

5.10.1.1 state

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

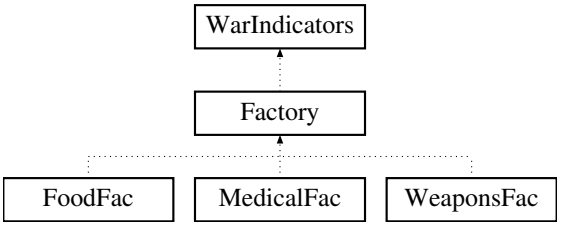
state of economy

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

- Economy.h
- Economy.cpp

5.11 Factory Class Reference

Inheritance diagram for Factory:



Public Member Functions

- void operation ()

  Calls the method needed to manufacture the supplies.
- ArmySupplies * getSupply ()

  Returns supplies.
- void ~Factory ()

  Deletes allocated memory to supply variable.

Protected Member Functions

- virtual ArmySupplies * make ()=0

Private Attributes

- ArmySupplies * **supply**

Additional Inherited Members

5.11.1 Constructor & Destructor Documentation

5.11.1.1 ~Factory()

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

Deletes allocated memory to supply variable.

Author

Ronin Brookes 19069686

5.11.2 Member Function Documentation

5.11.2.1 getSupply()

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

Returns supplies.

Returns

returns the supplies the Factory manufactured

Author

Ronin Brookes 19069686

5.11.2.2 make()

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

Implemented in FoodFac, MedicalFac, and WeaponsFac.

5.11.2.3 operation()

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

Calls the method needed to manufacture the supplies.

Author

Ronin Brookes 19069686

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

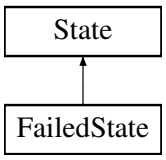
- Factory.h
- Factory.cpp

5.12 FailedState Class Reference

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

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

Inheritance diagram for FailedState:



Public Member Functions

- **FailedState** ()

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

Additional Inherited Members

5.12.1 Detailed Description

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

Author

Morgan Bentley

Date

October 2022

5.12.2 Member Function Documentation

5.12.2.1 decreaseCurrency()

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

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

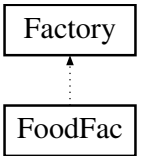
Implements State.

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

- FailedState.h
- FailedState.cpp

5.13 FoodFac Class Reference

Inheritance diagram for FoodFac:



Public Member Functions

- ArmySupplies * make ()

  Food Factory makes Food supplies.

Additional Inherited Members

5.13.1 Member Function Documentation

5.13.1.1 make()

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

Food Factory makes Food supplies.

Returns

returns the supplies the Food Factory just manufactured

Author

Ronin Brookes 19069686

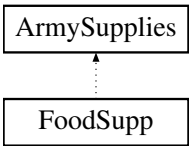
Implements Factory.

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

- FoodFac.h
- FoodFac.cpp

5.14 FoodSupp Class Reference

Inheritance diagram for FoodSupp:



Public Member Functions

- int getAmount ()

Additional Inherited Members

5.14.1 Member Function Documentation

5.14.1.1 getAmount()

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

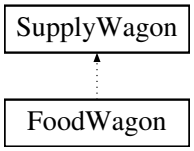
Implements ArmySupplies.

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

- FoodSupp.h

5.15 FoodWagon Class Reference

Inheritance diagram for FoodWagon:



Additional Inherited Members

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

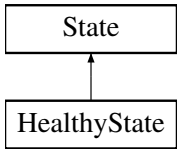
- FoodWagon.h

5.16 HealthyState Class Reference

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

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

Inheritance diagram for HealthyState:



Public Member Functions

- **HealthyState** ()

  Default constructor. initializes context to null.
- void decreaseCurrency ()
- virtual State * getDemotionState ()

Additional Inherited Members

5.16.1 Detailed Description

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

Author

Morgan Bentley

Date

October 2022

5.16.2 Member Function Documentation

5.16.2.1 decreaseCurrency()

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

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

Implements State.

5.16.2.2 getDemotionState()

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

Todo gets lower level state of economy

Returns

concrete State of Economy

Reimplemented from State.

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

- HealthyState.h
- HealthyState.cpp

5.17 Historian Class Reference

Public Member Functions

- Historian (Bannerman ∗b)

  constructor
- History ∗ setAlly ()

  set the memento object of the defecting bannerman
- Bannerman ∗ restoreAlly (History ∗h)

  return the bannerman that defected

Private Attributes

- Bannerman ∗ **bannerman**

5.17.1 Constructor & Destructor Documentation

5.17.1.1 Historian()

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

constructor

Parameters

b	bannerman that is defecting

5.17.2 Member Function Documentation

5.17.2.1 restoreAlly()

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

return the bannerman that defected

Parameters

h	the memento object that needs to be restored

Returns

the bannerman of that object

5.17.2.2 setAlly()

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

set the memento object of the defecting bannerman

Returns

the created memento object for that bannerman

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

- Historian.h
- Historian.cpp

5.18 History Class Reference

Public Member Functions

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

Private Attributes

- Bannerman * **bannerman**

5.18.1 Constructor & Destructor Documentation

5.18.1.1 History()

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

constructor

Parameters

b	initialisor for bannerman object whos state needs to be stored

5.18.2 Member Function Documentation

5.18.2.1 getBannerman()

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

get current saved bannerman

Returns

the saved bannerman that is defecting

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

- History.h
- History.cpp

5.19 HistoryBook Class Reference

Public Member Functions

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

Private Attributes

- list< History * > **defectedAllies**

5.19.1 Member Function Documentation

5.19.1.1 add()

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

add memento object to the list of defecetded allies

Parameters

h	the memento object added

5.19.1.2 restoreAlly()

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

restore the state of an ally returning

Parameters

h	the ally that wants to return

Returns

the state of the ally to be returning

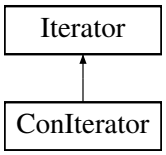
The documentation for this class was generated from the following files:

- HistoryBook.h
- HistoryBook.cpp

5.20 Iterator Class Reference

`#include <Iterator.h>`

Inheritance diagram for Iterator:



Public Member Functions

- virtual Bannerman * Current ()=0

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

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

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

5.20.1 Detailed Description

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

Author

Thapelo Thoka

Date

October 2022

5.20.2 Member Function Documentation

5.20.2.1 Current()

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

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

Returns

Bannerman∗

Implemented in ConIterator.

5.20.2.2 hasNext()

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

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

Returns

false

true

Implemented in ConIterator.

5.20.2.3 isActive()

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

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

Returns

true

false

Implemented in ConIterator.

5.20.2.4 next()

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

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

Returns

Bannerman∗

Implemented in ConIterator.

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

- Iterator.h

5.21 Kingdom Class Reference

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

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

Public Member Functions

- Kingdom (Economy ∗economy, vector< Bannerman ∗ > bannerman)

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

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

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

Private Attributes

- vector< Bannerman * > bannerman
- Economy * economy

5.21.1 Detailed Description

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

Author

Morgan Bentley

Date

October 2022

5.21.2 Constructor & Destructor Documentation

5.21.2.1 Kingdom()

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

Constructor. initializes Economy pointer and vector list.

Parameters

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

5.21.2.2 ~Kingdom()

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

Todo destructor. deallocates all pointers of this class

5.21.3 Member Function Documentation

5.21.3.1 add()

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

adds specified Bannerman into vector list.

Parameters

b	- Bannerman that has defected from enemyKingdom

5.21.3.2 remove()

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

removes specified Bannerman from vector list.

Parameters

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

5.21.4 Member Data Documentation

5.21.4.1 bannerman

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

vector list of all Bannerman objects a Kingdom owns

5.21.4.2 economy

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

pointer to the Economy of the Kingdom

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

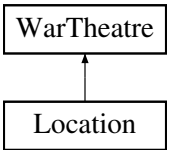
- Kingdom.h
- Kingdom.cpp

5.22 Location Class Reference

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

`#include <Location.h>`

Inheritance diagram for Location:



Public Member Functions

- **Location** ()
  Default constructor.
- void sendScout ()
  the function that will be called to decorate the war venue
- void **createVenue** ()
  creates default war theatre and calls the decorating function
- void ~**Location** ()
  fress memory used in the pattern

5.22.1 Detailed Description

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

Author

Keabetswe Mothapo

Date

October 2022

5.22.2 Member Function Documentation

5.22.2.1 sendScout()

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

the function that will be called to decorate the war venue

Implements WarTheatre.

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

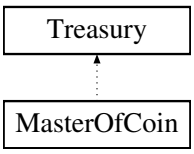
- Location.h
- Location.cpp

5.23 MasterOfCoin Class Reference

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

`#include <MasterOfCoin.h>`

Inheritance diagram for MasterOfCoin:



Public Member Functions

- MasterOfCoin (Economy *economy, Raven *observer, Strategy strategy)
  Constructor to set the variables in this class.
- void notify (WarIndicators sender)
  This is the method that all objects will use to call the mediator when a change is made.
- void **decreaseCurrency** ()
  Will call the decreaseCurrency method of the Economy.
- void **manufacture** ()
  will call on the obsever to create more supplies
- ~**MasterOfCoin** ()
  This is the destructer used to deallocate all memory used in this class.

Private Attributes

- Economy * **economy**
  The Economy that the mediator uses.
- Raven * **observer**
  The Observer that the mediator uses.
- Strategy * **strategy**
  The Strategy the mediator uses.

Additional Inherited Members

5.23.1 Detailed Description

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

Author

Sameet Keshav u21479373

Date

October 2022

5.23.2 Constructor & Destructor Documentation

5.23.2.1 MasterOfCoin()

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

Constructor to set the variables in this class.

Parameters

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

5.23.3 Member Function Documentation

5.23.3.1 notify()

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

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

Parameters

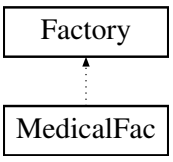
sender	is the variable that holds the object who sent the notify request

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

- MasterOfCoin.h
- MasterOfCoin.cpp

5.24 MedicalFac Class Reference

Inheritance diagram for MedicalFac:



Public Member Functions

- ArmySupplies * make ()

  Medical Factory makes Medical supplies.

Additional Inherited Members

5.24.1 Member Function Documentation

5.24.1.1 make()

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

Medical Factory makes Medical supplies.

Returns

returns the supplies the Medical Factory just manufactured

Author

Ronin Brookes 19069686

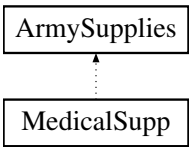
Implements Factory.

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

- MedicalFac.h
- MedicalFac.cpp

5.25 MedicalSupp Class Reference

Inheritance diagram for MedicalSupp:



Public Member Functions

- int getAmount ()

Additional Inherited Members

5.25.1 Member Function Documentation

5.25.1.1 getAmount()

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

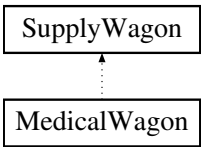
Implements ArmySupplies.

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

- MedicalSupp.h

5.26 MedicalWagon Class Reference

Inheritance diagram for MedicalWagon:



Additional Inherited Members

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

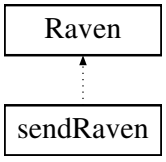
- MedicalWagon.h

5.27 Raven Class Reference

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

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

Inheritance diagram for Raven:



Public Member Functions

- virtual void update ()=0

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

5.27.1 Detailed Description

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

Author

Sameet Keshav u21479373

Date

October 2022

5.27.2 Member Function Documentation

5.27.2.1 update()

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

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

Implemented in sendRaven.

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

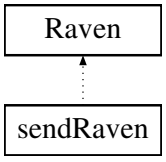
- Raven.h

5.28 sendRaven Class Reference

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

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

Inheritance diagram for sendRaven:



Public Member Functions

- sendRaven (SupplyWagon **supplies, Bannerman *subject)
  constructor for the concrete Raven class
- void update ()
  This method will be called from outside the class, this method will get the amount of resources the subject has, then call the check method.
- void **checkSupplies** ()
  This method will check to see if there is too little supplies then call the appropriate Supply Wagon to fill it up.
- ~**sendRaven** ()
  The Destructor will be used to deference all pointer variables used by this class to stop all memory leaks.

Private Attributes

- int **numFood**
  Amount of food the bannerman has.
- int **numMedical**
  Amount of medical supplies the bannerman has.
- int **numWeapons**
  Amount of medical supplies the bannerman has.
- SupplyWagon ** **supplies**
  The array of SupplyWagon's used to hold the different types of supplies.
- Bannerman * **subject**
  The Bannerman subject the Observer watches.

Additional Inherited Members

5.28.1 Detailed Description

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

Author

Sameet Keshav u21479373

Date

October 2022

5.28.2 Constructor & Destructor Documentation

5.28.2.1 sendRaven()

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

constructor for the concrete Raven class

Parameters

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

5.28.3 Member Function Documentation

5.28.3.1 update()

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

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

Implements Raven.

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

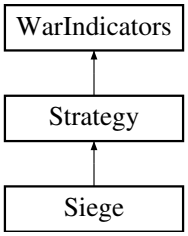
- sendRaven.h
- sendRaven.cpp

5.29 Siege Class Reference

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

```
#include <Siege.h>
```

Inheritance diagram for Siege:



Public Member Functions

- Siege (int stealth, Kingdom ∗myKingdom, Kingdom ∗enemyKingdom, Bannerman ∗myBannerman, Bannerman ∗enemyBannerman, string name, int min, int minFavour)
- bool attack (Bannerman ∗myBannerman, Bannerman ∗enemyBannerman)
- ∼Siege ()

Private Attributes

- int **stealth**

Additional Inherited Members

5.29.1 Detailed Description

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

Author

Morgan Bentley

Date

October 2022

5.29.2 Constructor & Destructor Documentation

5.29.2.1 Siege()

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

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

Parameters

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

5.29.2.2 ~Siege()

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

Todo destructor.

5.29.3 Member Function Documentation

5.29.3.1 attack()

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

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

Parameters

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:

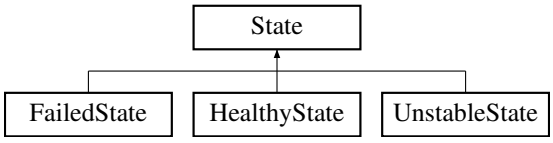
- Siege.h
- Siege.cpp

5.30 State Class Reference

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

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

Inheritance diagram for State:



Public Member Functions

- **State** ()

  Default constructor.
- virtual void setContext (Economy *context)
- virtual void decreaseCurrency ()=0
- virtual State * getDemotionState ()
- virtual ~State ()

Protected Attributes

- Economy * context

5.30.1 Detailed Description

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

Author

Morgan Bentley

Date

October 2022

5.30.2 Constructor & Destructor Documentation

5.30.2.1 ~State()

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

Todo destructor. deallocates context

5.30.3 Member Function Documentation

5.30.3.1 decreaseCurrency()

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

Todo Abstract method.

Implemented in FailedState, HealthyState, and UnstableState.

5.30.3.2 getDemotionState()

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

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

Returns

concrete State of Economy

Reimplemented in HealthyState, and UnstableState.

5.30.3.3 setContext()

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

Todo sets context variable.

Parameters

context	- Economy pointer to player's Economy object.

5.30.4 Member Data Documentation

5.30.4.1 context

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

Economy pointer of player's Kingdom

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

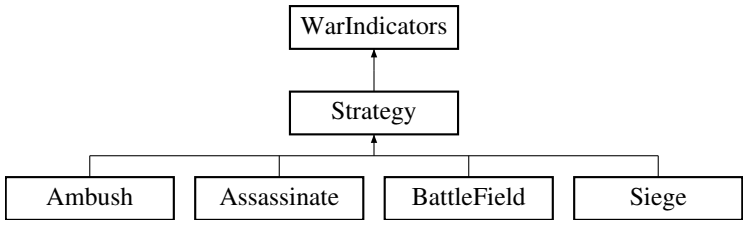
- State.h
- State.cpp

5.31 Strategy Class Reference

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

`#include <Strategy.h>`

Inheritance diagram for Strategy:



Public Member Functions

- Strategy (Kingdom *myKingdom, Kingdom *enemyKingdom, Bannerman *myBannerman, Bannerman *enemyBannerman, string name, int min, int minFavour)
- virtual bool attack (Bannerman *myBannerman, Bannerman *enemyBannerman)=0
- virtual string getStrategyName ()
- virtual Bannerman * getMyBannerman ()
- virtual Bannerman * getEnemyBannerman ()
- virtual ~Strategy ()

Protected Attributes

- Kingdom * myKingdom
- Kingdom * enemyKingdom
- Bannerman * myBannerman
- Bannerman * enemyBannerman
- string strategy
- int minFavour
- int minSupplies

5.31.1 Detailed Description

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

Author

Morgan Bentley

Date

October 2022

5.31.2 Constructor & Destructor Documentation

5.31.2.1 Strategy()

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

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

Parameters

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

5.31.2.2 ~Strategy()

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

Todo destructor. deallocates all pointers of this class

5.31.3 Member Function Documentation

5.31.3.1 attack()

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

Todo Abstract method

Parameters

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.

5.31.3.2 getEnemyBannerman()

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

Todo gets Bannerman being attacked

Returns

enemyBannerman pointer

5.31.3.3 getMyBannerman()

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

Todo gets attacking Bannerman

Returns

myBannerman pointer

5.31.3.4 getStrategyName()

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

Todo gets strategy variable

Returns

strategy variable

5.31.4 Member Data Documentation

5.31.4.1 enemyBannerman

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

defending Bannerman

5.31.4.2 enemyKingdom

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

Kingdom pointer of enemyBannerman's Kingdom

5.31.4.3 minFavour

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

minimum favour below which Bannerman change allegiances

5.31.4.4 minSupplies

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

minimum supplies for food,weapons and medicine

5.31.4.5 myBannerman

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

attacking Bannerman

5.31.4.6 myKingdom

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

Kingdom pointer of attacking Bannerman's Kingdom

5.31.4.7 strategy

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

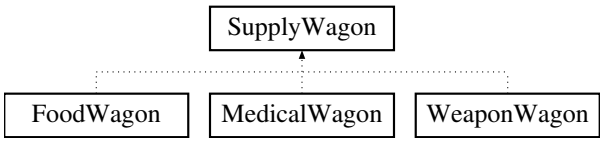
name of concrete strategy

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

- Strategy.h
- Strategy.cpp

5.32 SupplyWagon Class Reference

Inheritance diagram for SupplyWagon:



Public Member Functions

- void setSup (ArmySupplies *sup)=0
  initialize the supp variable.
- virtual SupplyWagon * clone ()=0
  Clone the Supply Wagon object.
- virtual ArmySupplies * getSupplies ()=0
  Return the supp variable.

Public Attributes

- ArmySupplies * **supp**

5.32.1 Member Function Documentation

5.32.1.1 clone()

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

Clone the Supply Wagon object.

Returns

the new Supply Wagon clone.

Author

Ronin Brookes 19069686

5.32.1.2 getSupplies()

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

Return the supp variable.

Returns

returns the supp variable. Returns the necessary supplies.

Author

Ronin Brookes 19069686

5.32.1.3 setSup()

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

initialize the supp variable.

Parameters

sup	is used to set the supp variable,

Author

Ronin Brookes 19069686

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

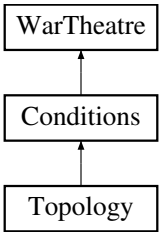
- SupplyWagon.h

5.33 Topology Class Reference

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

`#include <Topology.h>`

Inheritance diagram for Topology:



Public Member Functions

- Topology * getTopology ()
  getter for the Topology object
- void **setTopology** ()
  setter for the topology of the war venue uses the inherited venue variable to adjust difficuty of teh terrian
- **Topology** ()
  default constructor that calls the decorator function
- void sendScout ()
  implements the decorator function

Additional Inherited Members

5.33.1 Detailed Description

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

Author

Keabetswe Mothapo

Date

October 2022

5.33.2 Member Function Documentation

5.33.2.1 getTopology()

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

getter for the Topology object

Returns

- this topology object

5.33.2.2 sendScout()

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

implements the decorator function

Reimplemented from Conditions.

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

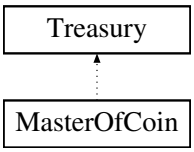
- Topology.h
- Topology.cpp

5.34 Treasury Class Reference

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

`#include <Treasury.h>`

Inheritance diagram for Treasury:



Public Member Functions

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

5.34.1 Detailed Description

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

Author

Sameet Keshav u21479373

Date

October 2022

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

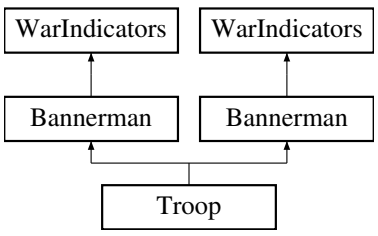
- Treasury.h

5.35 Troop Class Reference

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

`#include <Troop.h>`

Inheritance diagram for Troop:



Public Member Functions

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

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

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

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

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

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

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

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

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

  decreases the numFood variable of the troop by 1.
- void decreaseMedical () override

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

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

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

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

  Abstract. returns the favour of the bannerman.
- void increaseHP (int boost) override

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

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

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

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

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

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

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

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

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

Sets the number of medical supplies the troop has.

- void setRaven (list< Raven * > r)

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

- void setMaster (MasterOfCoin *m)

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

- ~Troop ()

Private Attributes

- int size
- Strategy * strategy
- int damage

Additional Inherited Members

5.35.1 Detailed Description

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

Author

Thapelo Thoka

Date

October 2022

5.35.2 Constructor & Destructor Documentation

5.35.2.1 ~Troop()

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

Default destructor.

5.35.3 Member Function Documentation

5.35.3.1 attach()

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

Attaches a Raven observer to the troop.

Parameters

o	- the Raven observer object to attach

Implements Bannerman.

5.35.3.2 attack()

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

uses strategy to make a troop attack another kingdom.

Parameters

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

Implements Bannerman.

5.35.3.3 changeStrategy()

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

Changes the attack strategy of the troop.

Parameters

strategy	- The new strategy the troop should have.

Implements Bannerman.

5.35.3.4 decreaseFavour()

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

Decreases the loyalty favour level of the troop by 1.

Implements Bannerman.

5.35.3.5 decreaseFood()

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

decreases the numFood variable of the troop by 1.

Implements Bannerman.

5.35.3.6 decreaseMedical()

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

decreases the numMedical variable of the troop by 1.

Implements Bannerman.

5.35.3.7 decreasePower()

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

Decreases the damage capability of the troop.

Parameters

x	- the number by which to decrease damage.

Implements Bannerman.

5.35.3.8 decreaseWeapons()

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

decreases the numWeapons variable of the troop by 1.

Implements Bannerman.

5.35.3.9 detach()

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

Detaches a Raven observer to the troop.

Parameters

o	- the Raven observer object to detach

Implements Bannerman.

5.35.3.10 getDamage()

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

Accessor. Returns the damage capacity of the troop.

Returns

damage.

Implements Bannerman.

5.35.3.11 getFavour()

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

Abstract. returns the favour of the bannerman.

Implements Bannerman.

5.35.3.12 getFood()

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

Accessor. Returns the amount of food the troop has.

Returns

numFood.

Implements Bannerman.

5.35.3.13 getHP()

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

Returns the troop's HP.

Returns

HP.

Implements Bannerman.

5.35.3.14 getMedical()

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

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

Returns

numMedical.

Implements Bannerman.

5.35.3.15 getSize()

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

Returns the troop's size.

Returns

size.

5.35.3.16 getWeapons()

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

Accessor. Returns the number of weapons the troop has.

Returns

numWeapons.

Implements Bannerman.

5.35.3.17 increaseFavour()

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

Increases the loyalty favour level of the troop by 1.

Implements Bannerman.

5.35.3.18 increaseHP()

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

increases the HP of the troop.

Parameters

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

Implements Bannerman.

5.35.3.19 increasePower()

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

increases the damage variables of the troop.

Parameters

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

Implements Bannerman.

5.35.3.20 receiveDamage()

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

increases the damage capacity of the troop.

Parameters

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

Implements Bannerman.

5.35.3.21 setFood()

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

Sets the number of food supplies the troop has.

Parameters

numFood	- The new numFood the troop should have.

Implements Bannerman.

5.35.3.22 setMaster()

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

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

Parameters

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

Implements Bannerman.

5.35.3.23 setMedical()

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

Sets the number of medical supplies the troop has.

Parameters

numMedical	- The new numMedical the troop should have.

Implements Bannerman.

5.35.3.24 setRaven()

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

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

Parameters

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

Implements Bannerman.

5.35.3.25 setWeapons()

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

Sets the number of weapon supplies the troop has.

Parameters

numWeapons	- The new numWeapons the troop should have.

Implements Bannerman.

5.35.4 Member Data Documentation

5.35.4.1 damage

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

The damage capability which the troop has

5.35.4.2 size

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

The number of soldiers in the troop

5.35.4.3 strategy

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

The attack strategy used by the troop in battle

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

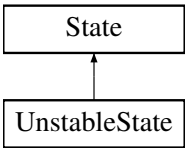
- Troop.h
- Troop.cpp

5.36 UnstableState Class Reference

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

`#include <UnstableState.h>`

Inheritance diagram for UnstableState:



Public Member Functions

- **UnstableState** ()
  Default constructor. initializes context to null.
- void decreaseCurrency ()
- State * getDemotionState ()

Additional Inherited Members

5.36.1 Detailed Description

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

Author

Morgan Bentley

Date

October 2022

5.36.2 Member Function Documentation

5.36.2.1 decreaseCurrency()

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

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

Implements State.

5.36.2.2 getDemotionState()

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

Todo gets lower level state of economy

Returns

concrete State of Economy

Reimplemented from State.

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

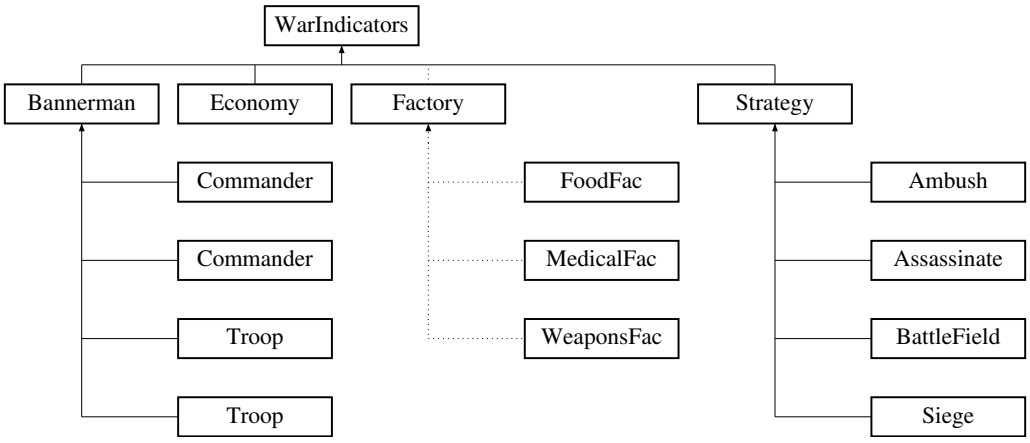
- UnstableState.h
- UnstableState.cpp

5.37 WarIndicators Class Reference

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

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

Inheritance diagram for WarIndicators:



Public Member Functions

- **WarIndicators** ()

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

  constructor. initializes m to the passed in Treasury object.
- virtual ∼WarIndicators ()

Protected Attributes

- Treasury ∗ m

5.37.1 Detailed Description

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

Author

Morgan Bentley

Date

October 2022

5.37.2 Constructor & Destructor Documentation

5.37.2.1 WarIndicators()

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

constructor. initializes m to the passed in Treasury object.

Parameters

m	- mediator which is of Treasury pointer type

5.37.2.2 ∼WarIndicators()

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

Todo destructor. deallocates Treasury pointer of this class

5.37.3 Member Data Documentation

5.37.3.1 m

```
Treasury* WarIndicators::m  [protected]
```

Treasury mediator pointer

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

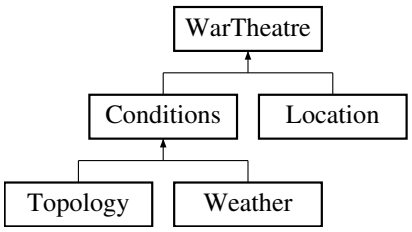
- WarIndicators.h

5.38 WarTheatre Class Reference

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

```
#include <WarTheatre.h>
```

Inheritance diagram for WarTheatre:



Public Member Functions

- WarTheatre (Strategy *myStrat)
  Constructor.
- Strategy * getStrategy ()
  returns the strategy the location is based on
- virtual void sendScout ()=0
  Abstract. Decorates the location of the battle.
- char decideVenue (Strategy *strategy)
  function that uses the parameter to decide the venue of the war
- **WarTheatre** (Strategy myStrat)

Private Attributes

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

5.38.1 Detailed Description

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

Author

Keabetswe Mothapo

Date

October 2022

5.38.2 Constructor & Destructor Documentation

5.38.2.1 WarTheatre()

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

Constructor.

Parameters

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

5.38.3 Member Function Documentation

5.38.3.1 decideVenue()

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

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

Parameters

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

5.38.3.2 getStrategy()

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

returns the strategy the location is based on

Returns

the strategy pointer

5.38.3.3 sendScout()

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

Abstract. Decorates the location of the battle.

Implemented in Conditions, Location, and Topology.

5.38.4 Member Data Documentation

5.38.4.1 difficulty

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

Integer representing the difficulty of the battle at the decorated venue

5.38.4.2 location

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

The location of the venue as a string

5.38.4.3 strategy

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

Name of Strategy as it determines the venue of the battle

5.38.4.4 venue

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

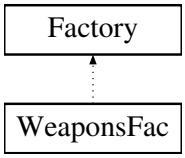
Char representing the venue. options a to d only

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

- WarTheatre.h
- WarTheatre.cpp

5.39 WeaponsFac Class Reference

Inheritance diagram for WeaponsFac:



Public Member Functions

- ArmySupplies * make ()

  Weapon Factory makes Weapon supplies.

Additional Inherited Members

5.39.1 Member Function Documentation

5.39.1.1 make()

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

Weapon Factory makes Weapon supplies.

Returns

returns the supplies the Weapon Factory just manufactured

Author

Ronin Brookes 19069686

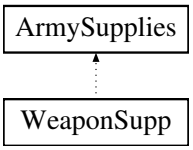
Implements Factory.

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

- WeaponsFac.h
- WeaponsFac.cpp

5.40 WeaponSupp Class Reference

Inheritance diagram for WeaponSupp:



Public Member Functions

- int getAmount ()

Additional Inherited Members

5.40.1 Member Function Documentation

5.40.1.1 getAmount()

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

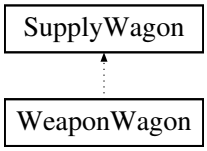
Implements ArmySupplies.

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

- WeaponSupp.h

5.41 WeaponWagon Class Reference

Inheritance diagram for WeaponWagon:



Additional Inherited Members

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

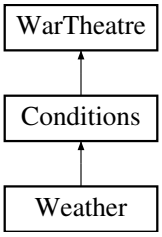
- WeaponWagon.h

5.42 Weather Class Reference

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

`#include <Weather.h>`

Inheritance diagram for Weather:



Public Member Functions

- Weather (int val)

  Costructor of the weather object.
- int calcEffect ()

  the function that uses the weather components to create a 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

Private Attributes

- double temp
- bool rain
- int windspeed

Additional Inherited Members

5.42.1 Detailed Description

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

Author

Keabetswe Mothapo

Date

October 2022

5.42.2 Constructor & Destructor Documentation

5.42.2.1 Weather()

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

Costructor of the weather object.

Parameters

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

5.42.3 Member Function Documentation

5.42.3.1 calcEffect()

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

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

Returns

returns the adjusted difficulty value

5.42.3.2 setTemp()

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

setter for the temperature variable

Parameters

t	- the value used to set the temp

5.42.3.3 setWindSpeed()

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

setter for the windspeed variable

Parameters

SP	- the value used to set the wind speed

5.42.4 Member Data Documentation

5.42.4.1 rain

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

Boolean to indicate whether it is raining or not

5.42.4.2 temp

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

Value of the temperature at the venue

5.42.4.3 windspeed

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

Value of the wind speed at the venue

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

- Weather.h
- Weather.cpp

Chapter 6

File Documentation

6.1 Ambush.h

```
1 #ifndef AMBUSH_H
2 #define AMBUSH_H
3 #include "Strategy.h"
4
10 class Ambush :public Strategy {
11 private:
13     int stealth;
14 public:
21     Ambush(int stealth,Kingdom* myKingdom,Kingdom* enemyKingdom,Bannerman* myBannerman, Bannerman*
      enemyBannerman,string name,int min,int minFavour);
27     bool attack(Bannerman* myBannerman, Bannerman* enemyBannerman);
29     ~Ambush();
30 };
31
32 #endif
```

6.2 ArmySupplies.h

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

6.3 Assassinate.h

```
1 #ifndef ASSASSINATE_H
2 #define ASSASSINATE_H
3 #include "Strategy.h"
4
10 class Assassinate :public Strategy {
11
12 private:
14     int stealth;
16     bool alive;
17 public:
24     Assassinate(int stealth, bool alive,Kingdom* myKingdom,Kingdom* enemyKingdom,Bannerman* myBannerman,
      Bannerman* enemyBannerman,string name,int min,int minFavour);
30     bool attack(Bannerman* myBannerman, Bannerman* enemyBannerman);
32     ~Assassinate();
33 };
34
35 #endif
```

6.4 Bannerman.h

```
1 #ifndef BANNERMAN_H
2 #define BANNERMAN_H
3 #include "Strategy.h"
4 #include "MasterOfCoin.h"
5 #include "Raven.h"
6 #include "WarIndicators.h"
7 #include "WarTheatre.h"
8 #include <string>
9 #include <list>
10
11
17 class Strategy;
18 class Bannerman:  public WarIndicators {
19
20 protected:
21
23     string name;
24
26     int favour;
27
29     int numWeapons;
30
32     int damage;
33
35     int numFood;
36
38     int numMedical;
39
41     list<Raven*> ravenList;
42
43
45     bool assassin;
46
48     MasterOfCoin* m;
49
51     Strategy* strategy;
52
54     int HP;
55
56
57 public:
60     Bannerman();
61
64     virtual void increaseFavour() = 0;
65
68     virtual void decreaseFavour() = 0;
69
73     virtual void attach(Raven* o)=0;
74
78     virtual void detach(Raven* o)=0;
79
83     virtual void increaseHP(int boost) = 0;//implement
84
88     virtual void changeStrategy(Strategy* strategy) = 0;
89
94     virtual void attack(Bannerman* myBannerman, Bannerman* enemyBannerman) = 0;
95
99     virtual void increasePower(int boost) = 0;
100
104     string getName();
105
109     virtual int getHP() = 0;
110
114     virtual int getDamage() = 0;
115
116
121     virtual void receiveDamage(int boost) = 0;
122
126     virtual void decreasePower(int x) = 0;
127
129     virtual void decreaseWeapons() = 0;
130
132     virtual void decreaseFood() = 0;
133
135     virtual void decreaseMedical() = 0;
136
140     virtual int getWeapons() = 0;
141
145     virtual int getFood() = 0;
146
150     virtual int getMedical() = 0;
151
152
156     virtual void setWeapons(int numWeapons) = 0;
157
```

```
161     virtual void setFood(int numFood) = 0;
162
166     virtual void setMedical(int numMedical) = 0;
167
171     virtual void setRaven(list<Raven*> r) = 0;
172
176     virtual void setMaster(MasterOfCoin* m) = 0;
177
180     virtual int getFavour() = 0;
184     ~Bannerman();
185 };
186
187 #endif
```

6.5 BattleField.h

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

6.6 Class.h

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

6.7 Commander.h

```
1 #ifndef COMMANDER_H
2 #define COMMANDER_H
3
4 #include "Bannerman.h"
5 #include "ConIterator.h"
6 #include <list>
7 class Iterator;
8 using namespace std;
9
15 class Commander :  public Bannerman {
16 private:
19     list<Bannerman*> groundForces;
20     string name;
21 public:
22
26     Commander(string name);
27
30     Iterator* createIterator();
31
34     void removeBannerman(Bannerman* x);
35
40     void attack(Bannerman* myBannerman, Bannerman* enemyBannerman) override;
41
44     void addBannerman(Bannerman* b);
45
47     list<Bannerman*> getTroops();
48
54     int getHP() override;
55
61     int getDamage() override;
65     void attach(Raven* o) override;
66
```

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

6.8 Conditions.h

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

6.9 ConIterator.h

```
1 #ifndef CONITERATOR_H
2 #define CONITERATOR_H
3 #include "Bannerman.h"
4 #include "Iterator.h"
5 #include <list>
6 using namespace std;
7
15 class ConIterator :  public Iterator {
16 private:
19      list<Bannerman*> armyList;
```

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

6.10 Economy.h

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

6.11 Factory.h

```
1
2  #ifndef FACTORY_H
3  #define FACTORY_H
4  #include "WarIndicators.h"
5  #include "ArmySupplies.h"
6
7  class Factory :  WarIndicators {
8
9  public:
12      void operation();
16      ArmySupplies* getSupply();
19      void ~Factory();
20 protected:
21      virtual ArmySupplies* make() = 0;
22 private:
23      ArmySupplies* supply;
24
25 };
26
27 #endif
```

6.12 FailedState.h

```
1  #ifndef FAILEDSTATE_H
2  #define FAILEDSTATE_H
3  #include "State.h"
```

```
4 #include "Economy.h"
5
11 class FailedState :public State {
12
13
14 public:
17     FailedState();
20     void decreaseCurrency();
21 };
22
23 #endif
```

6.13 FoodFac.h

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

6.14 FoodSupp.h

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

6.15 FoodWagon.h

```
1 #ifndef FOODWAGON_H
2 #define FOODWAGON_H
3 #include "Factory.h"
4 #include "SupplyWagon.h"
5 #include "ArmySupplies.h"
6 #include "FoodSupp.h"
7
8 class FoodWagon :  SupplyWagon {
9
10 public:
11     ~FoodWagon()
12 };
13
14 #endif
```

6.16 HealthyState.h

```
1 #ifndef HEALTHYSTATE_H
2 #define HEALTHYSTATE_H
3 #include "State.h"
4 #include "Economy.h"
5 #include "UnstableState.h"
6
12 class HealthyState :public State {
```

```
13
14
15 public:
18     HealthyState();
21     void decreaseCurrency();
24     virtual State* getDemotionState();
25 };
26
27 #endif
```

6.17 Historian.h

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

6.18 History.h

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

6.19 HistoryBook.h

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

6.20 Iterator.h

```
1 #ifndef ITERATOR_H
2 #define ITERATOR_H
3 #include "Bannerman.h"
4
11 class Iterator {
12
13
14 public:
20     virtual Bannerman* Current()=0;
26     virtual bool hasNext() = 0;
27
33     virtual Bannerman* next() = 0;
34
39     virtual bool isActive() = 0;
40 };
41
42 #endif
```

6.21 Kingdom.h

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

6.22 Location.h

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

6.23 MasterOfCoin.h

```
1 #ifndef MASTEROFCOIN_H
2 #define MASTEROFCOIN_H
3
4 #include "Economy.h"
```

```
5 #include "Raven.h"
6 #include "Strategy.h"
7 #include "Treasury.h"
8 #include "WarIndicators.h"
9
10
17 class MasterOfCoin :  Treasury {
18
19 private:
21     Economy* economy;
22
24     Raven* observer;
25
27     Strategy* strategy;
28
29 public:
34     MasterOfCoin(Economy* economy, Raven* observer, Strategy strategy);
35
38     void notify(WarIndicators sender);
39
41     void decreaseCurrency();
42
44     void manufacture();
45
47     ~MasterOfCoin();
48 };
49
50 #endif
```

6.24 MedicalFac.h

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

6.25 MedicalSupp.h

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

6.26 MedicalWagon.h

```
1 #ifndef MEDICALWAGON_H
2 #define MEDICALWAGON_H
3 #include "Factory.h"
4 #include "SupplyWagon.h"
5 #include "ArmySupplies.h"
6 #include "MedicalSupp.h"
7 class MedicalWagon :  SupplyWagon {
8
9 public:
12     ~MedicalWagon()
13 };
14
15 #endif
```

6.27 Raven.h

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

6.28 sendRaven.h

```
1 #ifndef SENDRAVEN_H
2 #define SENDRAVEN_H
3
4 #include "SupplyWagon.h"
5 #include "Commander.h"
6 #include "MasterOfCoin.h"
7 #include "Raven.h"
8 #include "Bannerman.h"
9
10
16 class sendRaven :  Raven {
17
18 private:
19
21     int numFood;
22
24     int numMedical;
25
27     int numWeapons;
28
30     SupplyWagon** supplies;
31
33     Bannerman* subject;
34
35 public:
39     sendRaven(SupplyWagon** supplies, Bannerman* subject);
40
42     void update();
43
45     void checkSupplies();
46
48     ~sendRaven();
49 };
50
51 #endif
```

6.29 Siege.h

```
1 #ifndef SIEGE_H
2 #define SIEGE_H
3 #include "Strategy.h"
4
10 class Siege :public Strategy {
11 private:
12     int stealth;
13 public:
20     Siege(int stealth,Kingdom* myKingdom,Kingdom* enemyKingdom,Bannerman* myBannerman, Bannerman*
       enemyBannerman,string name,int min,int minFavour);
26     bool attack(Bannerman* myBannerman, Bannerman* enemyBannerman);
28     ~Siege();
29 };
30
31 #endif
```

6.30 State.h

```
1 #ifndef STATE_H
2 #define STATE_H
3 #include "Economy.h"
```

```
4 class Economy;
5 using namespace std;
6
12 class State {
13
14 protected:
16     Economy* context;
17
18 public:
21     State();
24     virtual void setContext(Economy* context);
26     virtual void decreaseCurrency()=0;
29     virtual State* getDemotionState();
31     virtual ~State();
32 };
33
34 #endif
```

6.31 Strategy.h

```
1 #ifndef STRATEGY_H
2 #define STRATEGY_H
3 using namespace std;
4 #include <string>
5 #include "Bannerman.h"
6 #include "WarIndicators.h"
7 #include "MasterOfCoin.h"
8 #include "Kingdom.h"
9
16 class Strategy :  public WarIndicators  {
17
18 protected:
20     Kingdom* myKingdom;
22     Kingdom* enemyKingdom;
24     Bannerman* myBannerman;
26     Bannerman* enemyBannerman;
28     string strategy;
30     int minFavour;
32     int minSupplies;
33 public:
40     Strategy(Kingdom* myKingdom,Kingdom* enemyKingdom,Bannerman* myBannerman, Bannerman*
       enemyBannerman,string name,int min,int minFavour);
46     virtual bool attack(Bannerman* myBannerman, Bannerman* enemyBannerman)=0;
49     virtual string getStrategyName();
52     virtual Bannerman* getMyBannerman();
55     virtual Bannerman* getEnemyBannerman();
57     virtual ~Strategy();
58 };
59 #endif
```

6.32 SupplyWagon.h

```
1 #ifndef SUPPLYWAGON_H
2 #define SUPPLYWAGON_H
3 #include "ArmySupplies.h"
4
5 class SupplyWagon {
6
7 public:
11     void setSup(ArmySupplies* sup)=0;
15     virtual SupplyWagon* clone() = 0;
19     virtual ArmySupplies* getSupplies()=0;
20     ArmySupplies* supp;
21
22 };
23
24 #endif
```

6.33 Topology.h

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

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

6.34 Treasury.h

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

6.35 Troop.h

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

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

6.36 UnstableState.h

```
1 #ifndef UNSTABLESTATE_H
2 #define UNSTABLESTATE_H
3 #include "State.h"
4 #include "Economy.h"
5 #include "FailedState.h"
6
12 class UnstableState :public State {
13
14
15 public:
18     UnstableState();
21     void decreaseCurrency();
24     State* getDemotionState();
25 };
26
27 #endif
```

6.37 WarIndicators.h

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

6.38 WarTheatre.h

```
1 #ifndef WARTHEATRE_H
```

```
2 #define WARTHEATRE_H
3
4 using namespace std;
5
12 class WarTheatre {
13
14 private:
16     Strategy* strategy;
18     char venue;
19
21     string location;
22
24     int difficulty;
25
26
27 public:
30     WarTheatre(Strategy* myStrat);
31
34     Strategy* getStrategy(); //may need to make sendScout the virtual one instead
35
37     virtual void sendScout() = 0;
38
43     char decideVenue(Strategy* strategy);
44
45     WarTheatre(Strategy myStrat);
46 };
47
48 #endif
```

6.39 WeaponsFac.h

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

6.40 WeaponSupp.h

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

6.41 WeaponWagon.h

```
1 #ifndef WEAPONWAGON_H
2 #define WEAPONWAGON_H
3 #include "Factory.h"
4 #include "SupplyWagon.h"
5 #include "ArmySupplies.h"
6 #include "WeaponSupp.h"
7
8 class WeaponWagon :  SupplyWagon {
9
10 public:
11
12 ~WeaponWagon()
```

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

6.42 Weather.h

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

Index