My Project

Generated by Doxygen 1.8.6

Tue Oct 27 2015 10:39:53

Contents

Chapter 1

Todo List

Member User::moveUnit (Unit ∗, int, int)

update

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

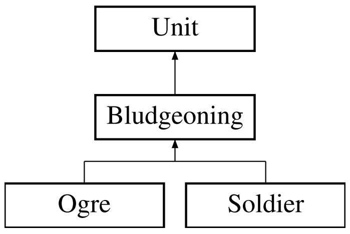
Class Documentation

5.1 Bludgeoning Class Reference

A class that extends Unit and has specialized abilities.

```
#include <Bludgeoning.h>
```

Inheritance diagram for Bludgeoning:



Public Member Functions

- Bludgeoning (int, int, string)

    Constructor calls Unit's constructor.
- **Bludgeoning** (Bludgeoning ∗)
- int attack (Unit ∗)

    Copy constructor used in the prototype pattern.
- void takeDamage (int)
- bool isAlive ()

    Overridden takeDamage function.
- Bludgeoning ∗ clone ()

    clone Clones the current unit using the copy constructor
- int getHealth ()

    returns the health of the unit
- int getDamage ()

    returns the damage of the unit
- string getType ()

    returns the type of unit
- void setHealth (int)

    sets the unit's health
- void setDamage (int)

    sets the units damage. Could be used if an upgrade mechanism is added to the game

- void setType (string)

    sets the type of unit.
- ∼Bludgeoning ()

    the destructor

5.1.1 Detailed Description

A class that extends Unit and has specialized abilities.

5.1.2 License

Copyright belongs to Merissa Joubert and Juan du Preez

5.1.3 Constructor & Destructor Documentation

5.1.3.1 Bludgeoning::Bludgeoning (int *h,* int *d,* string *t*)

Constructor calls Unit's constructor.

File: Bludgeoning.cpp Author: merissa Created on 06 October 2015, 4:33 PM

5.1.4 Member Function Documentation

5.1.4.1 int Bludgeoning::attack (Unit ∗ *target*) `[virtual]`

Copy constructor used in the prototype pattern.

Overridden attack function

Reimplemented from Unit.

5.1.4.2 bool Bludgeoning::isAlive ( ) `[virtual]`

Overridden takeDamage function.

isAlive Checks if the unit is still alive

Implements Unit.

5.1.4.3 void Bludgeoning::takeDamage (int *d*) `[virtual]`

Parameters

target	target is used to check if the current unit does more or less damage to it due to its type

Implements Unit.

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

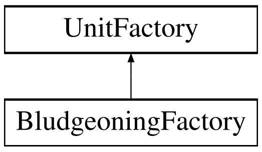
- Bludgeoning.h
- Bludgeoning.cpp

5.2 BludgeoningFactory Class Reference

A class that extends UnitFactory and can create specialized units.

`#include <BludgeoningFactory.h>`

Inheritance diagram for BludgeoningFactory:



Public Member Functions

- Unit ∗ createPlayer ()
- Unit ∗ createMonster ()

5.2.1 Detailed Description

A class that extends UnitFactory and can create specialized units.

5.2.2 License

Copyright belongs to Merissa Joubert and Juan du Preez

5.2.3 Member Function Documentation

5.2.3.1 Unit∗ BludgeoningFactory::createMonster ( ) `[inline]`, `[virtual]`

A pure virtual function for creating a monster

Returns

Unit pointer that was created

Implements UnitFactory.

5.2.3.2 Unit∗ BludgeoningFactory::createPlayer ( ) `[inline]`, `[virtual]`

A pure virtual function for creating a player

Returns

Unit pointer that was created

Implements UnitFactory.

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

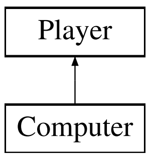
- BludgeoningFactory.h

5.3 Computer Class Reference

An abstract class that defines a computer player.

`#include <Computer.h>`

Inheritance diagram for Computer:



Public Member Functions

- void createUnit (UnitFactory ∗)

    createUnit creates a unit of a specific type
- Unit ∗ createUnit (string type)
- void makeMove ()

    Encapsulates the method that is used to move units across the map and add them to the maps.
- void moveUnit (Unit ∗)

    moves a unit to a new position on the map
- bool moveUp (Unit ∗unit, char toPlace)

    moveUp moves the current unit up one block on the map
- bool moveDown (Unit ∗unit, char toPlace)

    moveDown moves the current unit down one block on the map
- bool moveLeft (Unit ∗unit, char toPlace)

    moveLeft moves the current unit left one block on the map
- bool moveRight (Unit ∗unit, char toPlace)

    moveRight moves the current unit right one block on the map

Additional Inherited Members

5.3.1 Detailed Description

An abstract class that defines a computer player.

5.3.2 Description

Encapsulates the basic properties of computer players.

5.3.3 License

Copyright belongs to Juan du Preez and Merissa Joubert

5.3.4 Member Function Documentation

5.3.4.1 Unit ∗ Computer::createUnit (string *type*)

Parameters

type	holds the type of unit to be created

5.3.4.2 bool Computer::moveDown (Unit ∗ *unit,* char *toPlace*)

moveDown moves the current unit down one block on the map

Parameters

unit	is a unit pointer to the current unit being moved
a	character W/A/S/D that tells the mediator where the unit will move

5.3.4.3 bool Computer::moveLeft (Unit ∗ *unit,* char *toPlace*)

moveLeft moves the current unit left one block on the map

Parameters

unit	is a unit pointer to the current unit being moved
a	character W/A/S/D that tells the mediator where the unit will move

5.3.4.4 bool Computer::moveRight (Unit ∗ *unit,* char *toPlace*)

moveRight moves the current unit right one block on the map

Parameters

unit	is a unit pointer to the current unit being moved
a	character W/A/S/D that tells the mediator where the unit will move

5.3.4.5 void Computer::moveUnit (Unit ∗ *myUnit*)

moves a unit to a new position on the map

-

5.3.4.6 bool Computer::moveUp (Unit ∗ *unit,* char *toPlace*)

moveUp moves the current unit up one block on the map

Parameters

u	is a pointer to the unit that will be moved

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

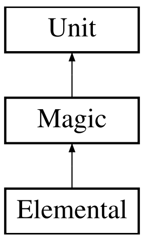
- Computer.h
- Computer.cpp

5.4 Elemental Class Reference

A Magic unit.

```
#include <Elemental.h>
```

Inheritance diagram for Elemental:



Public Member Functions

- Elemental ()

    The constructor calls the Magic constructor.

5.4.1 Detailed Description

A Magic unit.

5.4.2 License

Copyright belongs to Merissa Joubert and Juan du Preez

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

- Elemental.h
- Elemental.cpp

5.5 GamePlay Class Reference

A Facade that encapsulates all game functionality.

```
#include <GamePlay.h>
```

Public Member Functions

- GamePlay ()

    The constructor.
- ∼GamePlay ()

    The destructor.
- void play ()

    play encapsulates all game functionality
- void addPlayer (string name)

    adds a player to the game
- bool GameOver ()

    GameOver checks if there is more than one player left in the game and returns true if there is only one player left in the game.
- void removePlayer (Player ∗p)

    removePlayer removes a player from the game if they have no more units left
- Player ∗ getWinner ()

    getWinner returns a pointer to the winner of the game

Public Attributes

- Map ∗ map

    map is a map pointer
- Mediator ∗ **med**

5.5.1 Detailed Description

A Facade that encapsulates all game functionality.

5.5.2 Description

GamePlay uses all the other defined classes and patterns to execute the game.

5.5.3 License

Copyright belongs to Merissa Joubert and Juan du Preez

5.5.4 Member Function Documentation

5.5.4.1 bool GamePlay::GameOver ( )

GameOver checks if there is more than one player left in the game and returns true if there is only one player left in the game.

Parameters

name	holds the name of the player to be added to the game

5.5.4.2 Player ∗ GamePlay::getWinner ( )

getWinner returns a pointer to the winner of the game

Parameters

p	a pointer to a player in the players vector

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

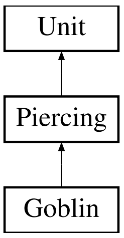
- GamePlay.h
- GamePlay.cpp

5.6 Goblin Class Reference

A Piercing unit.

```
#include <Goblin.h>
```

Inheritance diagram for Goblin:



Public Member Functions

- Goblin ()

    the constructor calls the Piercing constructor

5.6.1 Detailed Description

A Piercing unit.

5.6.2 License

Copyright belongs to Merissa Joubert and Juan du Preez

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

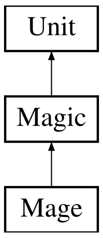
- Goblin.h
- Goblin.cpp

5.7 Mage Class Reference

A Magic unit.

`#include <Mage.h>`

Inheritance diagram for Mage:



Public Member Functions

- Mage ()

    The constructor calls the Magic constructor.

5.7.1 Detailed Description

A Magic unit.

5.7.2 License

Copyright belongs to Merissa Joubert and Juan du Preez

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

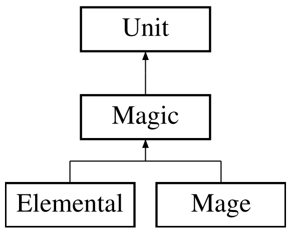
- Mage.h
- Mage.cpp

5.8 Magic Class Reference

A class that extends Unit and has specialized abilities.

`#include <Magic.h>`

Inheritance diagram for Magic:



Public Member Functions

- Magic (int, int, string)

  Constructor calls Unit's constructor.
- **Magic** (Magic ∗)
- int attack (Unit ∗)

  Copy constructor used in the prototype pattern.
- void takeDamage (int d)
- bool isAlive ()

  Overridden takeDamage function.
- Magic ∗ clone ()

  clone Clones the current unit using the copy constructor
- int getHealth ()

  returns the units health
- int getDamage ()

  returns the units damage
- string getType ()

  returns the units Type
- void setHealth (int)

  sets the units health
- void setDamage (int)

  sets the units damage
- void setType (string)

  sets the units Type
- ∼Magic ()

  The Destructor.

5.8.1 Detailed Description

A class that extends Unit and has specialized abilities.

5.8.2 License

Copyright belongs to Merissa Joubert and Juan du Preez

5.8.3 Constructor & Destructor Documentation

5.8.3.1 Magic::Magic (int *h*, int *d*, string *t*)

Constructor calls Unit's constructor.

File: Magic.cpp Author: merissa Created on 06 October 2015, 4:32 PM

5.8.4 Member Function Documentation

5.8.4.1 int Magic::attack (Unit ∗ *target*) `[virtual]`

Copy constructor used in the prototype pattern.

Overridden attack function

Reimplemented from Unit.

5.8.4.2 bool Magic::isAlive ( ) `[virtual]`

Overridden takeDamage function.

isAlive Checks if the unit is still alive

Implements Unit.

5.8.4.3 void Magic::takeDamage (int *d*) `[virtual]`

Parameters

target	target is used to check if the current unit does more or less damage to it due to its type

Implements Unit.

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

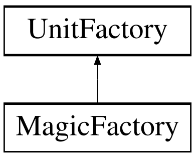
- Magic.h
- Magic.cpp

5.9 MagicFactory Class Reference

Extends UnitFactory to create specialized units.

`#include <MagicFactory.h>`

Inheritance diagram for MagicFactory:



Public Member Functions

- Unit ∗ createPlayer ()
- Unit ∗ createMonster ()

5.9.1 Detailed Description

Extends UnitFactory to create specialized units.

5.9.2 License

Copyright belongs to Merissa Joubert and Juan du Preez

5.9.3 Member Function Documentation

5.9.3.1 **Unit**∗ MagicFactory::createMonster ( ) `[inline], [virtual]`

A pure virtual function for creating a monster

Returns

Unit pointer that was created

Implements UnitFactory.

5.9.3.2 **Unit**∗ MagicFactory::createPlayer ( ) `[inline], [virtual]`

A pure virtual function for creating a player

Returns

Unit pointer that was created

Implements UnitFactory.

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

- MagicFactory.h

5.10 Map Class Reference

Public Member Functions

- **Map** (const char ∗)
- void **printMap** ()
- moveState Move (int, int, int, int, char)
- void **setMap** ()

5.10.1 Member Function Documentation

5.10.1.1 moveState Map::Move (int *fromX,* int *fromY,* int *toX,* int *toY,* char *toPlace*)

Complete the move to allow an update to the map

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

- Map.h
- Map.cpp

5.11 Mediator Class Reference

Defines an interface for communication between the players and computers.

`#include <Mediator.h>`

Public Member Functions

- Mediator (Map ∗)

  The constructor recieves a Map pointer and attaches itself to the map.
- void addPlayer (Player ∗)

  addPlayer adds a new player to the map.
- bool update (Unit ∗, int, int, char)

  update updates the current Unit's position on the map
- void **printMap** ()

5.11.1 Detailed Description

Defines an interface for communication between the players and computers.

5.11.2 Description

The mediator updates the map whenever players move their units around or add new units to the game

5.11.3 License

Copyright belongs to Merissa Joubert and Juan du Preez

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

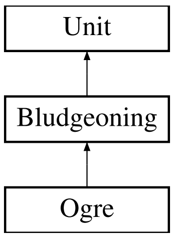
- Mediator.h
- mediator.cpp

5.12 Ogre Class Reference

A Bludgeoning unit.

`#include <Ogre.h>`

Inheritance diagram for Ogre:



Public Member Functions

- Ogre ()

  Calls the Bludgeoning constructor.

5.12.1 Detailed Description

A Bludgeoning unit.

5.12.2 License

Copyright belongs to Merissa Joubert and Juan du Preez

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

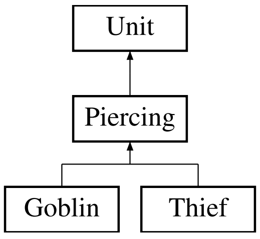
- Ogre.h
- Ogre.cpp

5.13 Piercing Class Reference

A class that extends Unit and has specialized abilities.

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

Inheritance diagram for Piercing:



Public Member Functions

- Piercing (int, int, string)

  Constructor calls Unit's constructor.
- **Piercing** (Piercing $*$)
- int attack (Unit $*$target)

  Copy constructor used in the prototype pattern.
- void takeDamage (int)
- bool isAlive ()

Overridden takeDamage function.

- Piercing ∗ clone ()

  clone Clones the current unit using the copy constructor
- int getHealth ()

  returns the Units health
- int getDamage ()

  returns the damage of the unit
- string getType ()

  returns the type of unit
- void setHealth (int)

  sets the unit's health
- void setDamage (int)

  sets the units damage. Could be used if an upgrade mechanism is added to the game
- void setType (string)

  sets the type of unit.

5.13.1 Detailed Description

A class that extends Unit and has specialized abilities.

5.13.2 License

Copyright belongs to Merissa Joubert and Juan du Preez

5.13.3 Constructor & Destructor Documentation

5.13.3.1 Piercing::Piercing (int *h*, int *d*, string *t*)

Constructor calls Unit's constructor.

File: Piercing.cpp Author: merissa Created on 06 October 2015, 4:33 PM

5.13.4 Member Function Documentation

5.13.4.1 int Piercing::attack (Unit ∗ *target*) [virtual]

Copy constructor used in the prototype pattern.

Overridden attack function

Reimplemented from Unit.

5.13.4.2 bool Piercing::isAlive ( ) [virtual]

Overridden takeDamage function.

isAlive Checks if the unit is still alive

Implements Unit.

5.13.4.3 void Piercing::takeDamage (int *d*) [virtual]

Parameters

target	target is used to check if the current unit does more or less damage to it due to its type

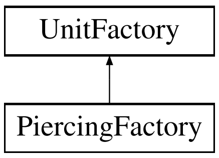
Implements Unit.

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

- Piercing.h
- Piercing.cpp

5.14 PiercingFactory Class Reference

Inheritance diagram for PiercingFactory:



Public Member Functions

- Unit $*$ createPlayer ()
- Unit $*$ createMonster ()

5.14.1 Member Function Documentation

5.14.1.1 Unit$*$ PiercingFactory::createMonster ( ) `[inline]`, `[virtual]`

A pure virtual function for creating a monster

Returns

Unit pointer that was created

Implements UnitFactory.

5.14.1.2 Unit$*$ PiercingFactory::createPlayer ( ) `[inline]`, `[virtual]`

A pure virtual function for creating a player

Returns

Unit pointer that was created

Implements UnitFactory.

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

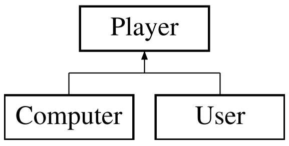
- PiercingFactory.h

5.15 Player Class Reference

An abstract class that defines a player.

`#include <Player.h>`

Inheritance diagram for Player:



Public Member Functions

- void **checkUnits** ()
- Player ()

  The constructor.
- ∼Player ()

  The destructor.
- void addToArmy (Unit ∗u)

  addToArmy adds a new unit to the army
- void printArmy ()

  prints all the units that are currently stored in the player's army
- virtual void createUnit (UnitFactory ∗)=0
- bool IsOut ()

  isOut returns true if the player has no more units in their army
- virtual void makeMove ()=0

  makeMove encapsulates the algorithm that lets players make and move units
- void **setMaxUnits** (int m)
- void setName (string n)

  setMaxUnits sets the maximum units in the game
- string getName ()

  getName returns the name of the player
- void setNumMoves (int n)

  setNumMoves sets the number of moves a player has
- void decreaseMoves ()

  decreaseMoves subtracts one from the player's current number of moves
- bool outOfMoves ()

  returns true if the player has no more moves left and false otherwise
- void setMediator (Mediator ∗)

  setMediator attaches the player to the mediator pointer passed as a parameter

Public Attributes

- vector< Unit ∗ > army

  Stores all the units that the player owns.
- vector< UnitFactory ∗ > baracks

  Stores the three unit factories.

Protected Attributes

- Mediator ∗ mediator

  mediator stores a pointer to the mediator that this player belongs to

5.15.1 Detailed Description

An abstract class that defines a player.

5.15.2 Description

Encapsulates the basic properties of both computer players and human players

5.15.3 License

Copyright belongs to Merissa Joubert and Juan du Preez

5.15.4 Member Function Documentation

5.15.4.1 void Player::addToArmy (Unit ∗ *u*)

addToArmy adds a new unit to the army

<adds the desired unit to the player's army

5.15.4.2 virtual void Player::createUnit (UnitFactory ∗) `[pure virtual]`

createUnit is a pure virtual method that is overidden by computer and user to add the correct type of unit to the player's army

Implemented in Computer, and User.

5.15.4.3 void Player::decreaseMoves ( )

decreaseMoves subtracts one from the player's current number of moves

Parameters

n	an integer that stores a player's number of moves

5.15.4.4 string Player::getName ( )

getName returns the name of the player

Parameters

n	the name of the player

5.15.4.5 void Player::printArmy ( )

prints all the units that are currently stored in the player's army

Parameters

u	is a unit pointer to a unit object that will be added to the player's army

5.15.4.6 void Player::setName (string *n*)

setMaxUnits sets the maximum units in the game

Parameters

m	an integer that stores the maximum number of unitssetName sets the plaeyer's name

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

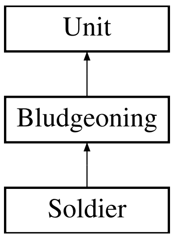
- Player.h
- Player.cpp

5.16 Soldier Class Reference

A Bludgeoning unit.

`#include <Soldier.h>`

Inheritance diagram for Soldier:



Public Member Functions

- Soldier ()

  Calls the Bludgeoning constructor.

5.16.1 Detailed Description

A Bludgeoning unit.

5.16.2 License

Copyright belongs to Merissa Joubert and Juan du Preez

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

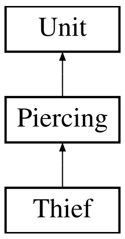
- Soldier.h
- Soldier.cpp

5.17 Thief Class Reference

A Piercing unit.

`#include <Thief.h>`

Inheritance diagram for Thief:



Public Member Functions

- Thief ()

  calls the Piercing constructor

5.17.1 Detailed Description

A Piercing unit.

5.17.2 License

Copyright belongs to Merissa Joubert and Juan du Preez

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

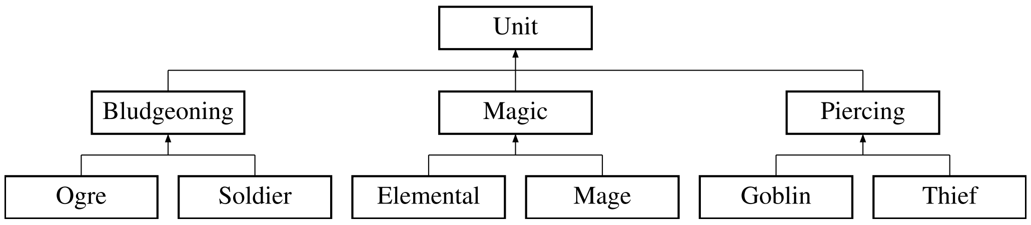
- Thief.h
- Thief.cpp

5.18 Unit Class Reference

An abstract class that describes all units of the game.

`#include <Unit.h>`

Inheritance diagram for Unit:



Public Member Functions

- **Unit** (int h, int d, string t)

- virtual ostream & **printUnit** (ostream &os)
- virtual int attack (Unit ∗a)

    The destructor.
- virtual void takeDamage (int d)=0

    The takeDamage function.
- virtual bool isAlive ()=0

    The is Alive function returns true if the unit is alive and false otherwise.
- virtual Unit ∗ clone ()=0

    The clone function.
- virtual int getHealth ()
- virtual int getDamage ()

    getHealth returns the unit's health
- virtual void setHealth (int h)

    getDamage returns the unit's damage
- virtual void setDamage (int m)

    setHealth sets the unit's damage
- virtual void setType (string t)

    setDamage sets the unit's damage
- virtual string getType ()

    setType can change the unit's type
- virtual void setCost (double c)

    getType returns the unit's type
- virtual double getCost ()

    setCost sets the cost of the unit depending on its type
- virtual int getXpos ()

    getCost returns the cost of the unit
- virtual int getYpos ()

    returns the current xpos of the unit
- virtual void move (int x, int y)

    returns the current ypos of the unit

Friends

- ostream & operator<< (ostream &os, Unit &myUnit)

    The Constructor.

5.18.1 Detailed Description

An abstract class that describes all units of the game.

5.18.2 Description

The unit class is an abstract Product class that provides an interface for other abstract products to interact with.

5.18.3 License

Copyright belongs to Merissa Joubert and Juan du Preez

5.18.4 Member Function Documentation

5.18.4.1 virtual int Unit::attack (Unit * *a*) `[inline]`, `[virtual]`

The destructor.

The attack function

Reimplemented in Magic, Bludgeoning, and Piercing.

5.18.4.2 virtual int Unit::getHealth ( ) `[inline]`, `[virtual]`

The clone function is pure virtual in the unit class making it abstract. This is because each clone needs to be of one of the subclasses

Reimplemented in Magic, Bludgeoning, and Piercing.

5.18.4.3 virtual bool Unit::isAlive ( ) `[pure virtual]`

The is Alive function returns true if the unit is alive and false otherwise.

The take damage function simply subtracts damage dealt to the unit from its health

Implemented in Magic, Bludgeoning, and Piercing.

5.18.4.4 virtual void Unit::takeDamage (int *d*) `[pure virtual]`

The takeDamage function.

The attack function is overridden in all subclasses in order to be specific to each unit

Implemented in Magic, Bludgeoning, and Piercing.

5.18.5 Friends And Related Function Documentation

5.18.5.1 ostream& operator<< (ostream & *os*, Unit & *myUnit*) `[friend]`

The Constructor.

The constructor simpy sets all the member variables to the input parameters provided.

Parameters

h	is an integer that is used to initialize the unit's max health.
d	is an integer that is used to initialize the unit's starting damage.
t	is a string that is used to initialize the unit's type eg. Magic/Piercing/Bludgeoning.

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

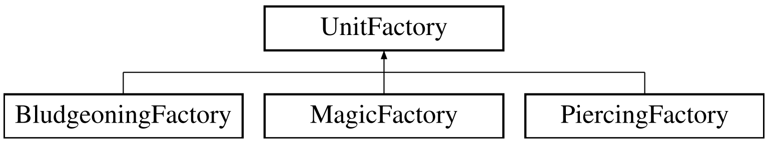
- Unit.h

5.19 UnitFactory Class Reference

An abstract class that provides an interface for the creation of units in the game.

`#include <UnitFactory.h>`

Inheritance diagram for UnitFactory:



Public Member Functions

- UnitFactory ()
- virtual ∼UnitFactory ()
- virtual Unit ∗ createPlayer ()=0
- virtual Unit ∗ createMonster ()=0

5.19.1 Detailed Description

An abstract class that provides an interface for the creation of units in the game.

5.19.2 License

Copyright belongs to Merissa Joubert and Juan du Preez

5.19.3 Constructor & Destructor Documentation

5.19.3.1 UnitFactory::UnitFactory ( ) `[inline]`

A constructor for creating a unit factory

5.19.3.2 virtual UnitFactory::∼UnitFactory ( ) `[inline]`, `[virtual]`

A destructor for deleting a unit factory

5.19.4 Member Function Documentation

5.19.4.1 virtual Unit∗ UnitFactory::createMonster ( ) `[pure virtual]`

A pure virtual function for creating a monster

Returns

Unit pointer that was created

Implemented in BludgeoningFactory, MagicFactory, and PiercingFactory.

5.19.4.2 virtual Unit∗ UnitFactory::createPlayer ( ) `[pure virtual]`

A pure virtual function for creating a player

Returns

Unit pointer that was created

Implemented in BludgeoningFactory, MagicFactory, and PiercingFactory.

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

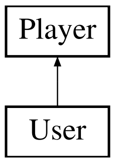
- UnitFactory.h

5.20 User Class Reference

A human player.

```
#include <User.h>
```

Inheritance diagram for User:



Public Member Functions

- void createUnit (UnitFactory $*$)

    createUnit uses a unitFactory to create a new unit and add it to the player's army
- void makeMove ()

    makeMove encapsulates the algorithm used to move units across the map and add them to the map
- void moveUnit (Unit $*$, int, int)
- void moveUp (Unit $*$)

    moveUp moves the current unit one block up on the map
- void moveDown (Unit $*$)

    moveDown moves the current unit one block down on the map
- void moveLeft (Unit $*$)

    moveLeft moves the current unit one block left on the map
- void moveRight (Unit $*$)

    moveRight moves the current unit one block to the right on the map

Additional Inherited Members

5.20.1 Detailed Description

A human player.

5.20.2 Description

A class that defines a human player

5.20.3 License

Copyright belongs to Merissa Joubert and Juan du Preez

5.20.4 Member Function Documentation

5.20.4.1 void User::makeMove ( ) [virtual]

makeMove encapsulates the algorithm used to move units across the map and add them to the map

Parameters

u	is a UnitFactory used to create the new unit;

Implements Player.

5.20.4.2 void User::moveDown (Unit ∗ *unit*)

moveDown moves the current unit one block down on the map

do a switch statement based on Merissa's input.

5.20.4.3 void User::moveLeft (Unit ∗ *unit*)

moveLeft moves the current unit one block left on the map

do a switch statement based on Merissa's input.

5.20.4.4 void User::moveRight (Unit ∗ *unit*)

moveRight moves the current unit one block to the right on the map

do a switch statement based on Merissa's input.

5.20.4.5 void User::moveUnit (Unit ∗ *myUnit,* int *x,* int *y*)

Todo update

<this moves the unit on the map according to what the user enters

5.20.4.6 void User::moveUp (Unit ∗ *unit*)

moveUp moves the current unit one block up on the map

do a switch statement based on Merissa's input.

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

- User.h
- User.cpp

Chapter 6

File Documentation

6.1 Bludgeoning.h File Reference

```
#include "Unit.h"
#include <string>
```

Classes

- class Bludgeoning

  A class that extends Unit and has specialized abilities.

6.2 Computer.h File Reference

```
#include "Player.h"
#include <iostream>
#include <ctime>
#include <cstdlib>
```

Classes

- class Computer

  An abstract class that defines a computer player.

6.3 Elemental.h File Reference

```
#include "Magic.h"
#include <string>
```

Classes

- class Elemental

  A Magic unit.

6.4 GamePlay.h File Reference

```
#include "Player.h"
#include "User.h"
#include "Computer.h"
#include "Unit.h"
#include "UnitFactory.h"
#include "MagicFactory.h"
#include "BludgeoningFactory.h"
#include "PiercingFactory.h"
#include "Thief.h"
#include "Mage.h"
#include "Elemental.h"
#include "Goblin.h"
#include "Ogre.h"
#include "Soldier.h"
#include <vector>
#include "Map.h"
#include "Mediator.h"
```

Classes

- class GamePlay

  A Facade that encapsulates all game functionality.

6.5 Goblin.h File Reference

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

Classes

- class Goblin

  A Piercing unit.

6.6 Mage.h File Reference

```
#include "Magic.h"
#include <string>
```

Classes

- class Mage

  A Magic unit.

6.7 Magic.h File Reference

```
#include "Unit.h"
#include <string>
```

Classes

- class Magic

    A class that extends Unit and has specialized abilities.

6.8 MagicFactory.h File Reference

```
#include "UnitFactory.h"
#include "Mage.h"
#include "Elemental.h"
```

Classes

- class MagicFactory

    Extends UnitFactory to create specialized units.

6.9 Mediator.h File Reference

```
#include "Map.h"
#include "Unit.h"
#include <vector>
```

Classes

- class Mediator

    Defines an interface for communication between the players and computers.

6.10 Ogre.h File Reference

```
#include "Bludgeoning.h"
#include <string>
```

Classes

- class Ogre

    A Bludgeoning unit.

6.11 Piercing.h File Reference

```
#include "Unit.h"
#include <string>
```

Classes

- class Piercing

  A class that extends Unit and has specialized abilities.

6.12 Player.h File Reference

```
#include "Unit.h"
#include "UnitFactory.h"
#include "MagicFactory.h"
#include "PiercingFactory.h"
#include "BludgeoningFactory.h"
#include <vector>
#include <string>
#include "Mediator.h"
```

Classes

- class Player

  An abstract class that defines a player.

6.13 Soldier.h File Reference

```
#include "Bludgeoning.h"
#include <string>
```

Classes

- class Soldier

  A Bludgeoning unit.

6.14 Thief.h File Reference

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

Classes

- class Thief

  A Piercing unit.

6.15 Unit.h File Reference

```
#include <string>
#include <ostream>
```

Classes

- class Unit

  An abstract class that describes all units of the game.

Index