# COS 121 Project

Generated by Doxygen 1.8.10

Mon Oct 26 2015 16:15:01

# Contents

# Chapter 1

# Hierarchical Index

## 1.1 Class Hierarchy

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

# Chapter 2

# Class Index

## 2.1 Class List

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

# Chapter 3

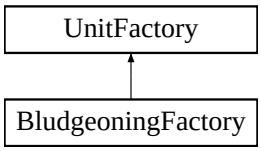
# Class Documentation

## 3.1 BludgeoningFactory Class Reference

Concrete factory that creates Soldier or Ogre objects.

`#include <BludgeoningFactory.h>`

Inheritance diagram for BludgeoningFactory:




### Public Member Functions

- virtual Unit * createPlayer ()

  *Concrete implementation of createPlayer, will create a Soldier.*

- virtual Unit * createMonster ()

  *Concrete impletation of createMonster, will create an Ogre.*

### Additional Inherited Members

### 3.1.1 Detailed Description

Concrete factory that creates Soldier or Ogre objects.

**See also**

UnitFactory

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

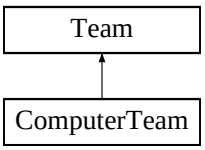
- BludgeoningFactory.h
- BludgeoningFactory.cpp

## 3.2 ComputerTeam Class Reference

A concrete Team with automatic turns.

`#include <ComputerTeam.h>`

Inheritance diagram for ComputerTeam:




### Public Member Functions

- ComputerTeam (GameMaster *inputGameMaster)

  *Constructor that takes a GameMater pointer.*
- virtual void initUnits ()

  *Function to initialize units of a team randomly.*
- virtual void turn ()

  *Function that moves teamates and attack if possible.*
- virtual void update (Team *)

  *Observer function to indicate that a turn is over.*
- virtual void attack ()

  *A collection of actions representing an attack.*

### Additional Inherited Members

#### 3.2.1 Detailed Description

A concrete Team with automatic turns.

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

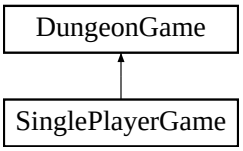
- ComputerTeam.h
- ComputerTeam.cpp

## 3.3 DungeonGame Class Reference

A simple interface for creating typical games.

`#include <DungeonGame.h>`

Inheritance diagram for DungeonGame:



### Public Member Functions

- virtual void setupGame ()=0

  *Pure virtual function, which concrete games will define specific to requirements.*
- virtual void beginGame ()=0

  *Pure virtual function for begining games of all concrete games.*

#### 3.3.1 Detailed Description

A simple interface for creating typical games.

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

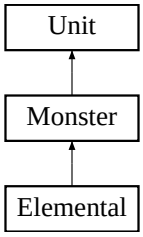
- DungeonGame.h
- DungeonGame.cpp

## 3.4 Elemental Class Reference

A concrete Unit; Inherits from Monster.

`#include <Elemental.h>`

Inheritance diagram for Elemental:




### Public Member Functions

- Elemental ()

### Additional Inherited Members

#### 3.4.1 Detailed Description

A concrete Unit; Inherits from Monster.

**See also**

Monster ()

#### 3.4.2 Constructor & Destructor Documentation

##### 3.4.2.1 Elemental::Elemental ( )

Constructor for Elemental class sets the stats and respective "class" of Elemental.

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

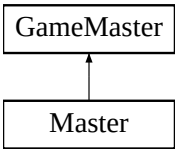
- Elemental.h
- Elemental.cpp

## 3.5 GameMaster Class Reference

An abstract class defining a general manager for a game.

`#include <GameMaster.h>`

Inheritance diagram for GameMaster:




### Public Member Functions

- GameMaster ()

  *Constructor for class GameMaster.*
- virtual void attachTeam (Team *inputTeam)=0

  *Virtual destructor for class Unit.*
- virtual void detachTeam (Team *inputTeam)=0

  *Virtual destructor for class Unit.*
- void **playGame** ()
- virtual bool **moveUnit** (Unit *inputUnit, string direction)=0
- virtual void **notify** (Team *)=0
- void **attack** (Unit *attackingUnit, Unit *defendingUnit)
- int **getNumberTeams** ()
- Team * **getTeamAt** (int index)
- virtual void printMap ()=0

  *Virtual destructor for class Unit.*
- bool **gameOver** ()
- virtual void **addToMap** (Unit *inputUnit, int x, int y)=0
- virtual vector< int > **locateUnit** (Unit *inputUnit)=0
- virtual Unit * **locateUnit** (int row, int col)=0
- virtual vector< int > requestFreeSpace ()=0

  *Pure.*
- virtual void removeDestroyedUnits ()=0

  *Pure virtual function for removing destroyed Units.*

### Protected Attributes

- vector< Team * > **teams**
- unsigned int **currentTurn**

### 3.5.1 Detailed Description

An abstract class defining a general manager for a game.

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

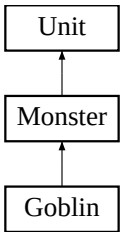
- GameMaster.h
- GameMaster.cpp

## 3.6 Goblin Class Reference

A concrete Unit; Inherits from Monster.

`#include <Goblin.h>`

Inheritance diagram for Goblin:




### Public Member Functions

- Goblin ()

### Additional Inherited Members

### 3.6.1 Detailed Description

A concrete Unit; Inherits from Monster.

**See also**

Monster ()

### 3.6.2 Constructor & Destructor Documentation

#### 3.6.2.1 Goblin::Goblin ( )

Constructor for Goblin class sets the stats and respective "class" of Goblin.

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

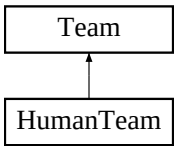
- Goblin.h
- Goblin.cpp

## 3.7 HumanTeam Class Reference

Concrete Team that has a number of Units.

`#include <HumanTeam.h>`

Inheritance diagram for HumanTeam:




### Public Member Functions

- HumanTeam (GameMaster *inputGameMaster)

  *Constructor for class HumanTeam.*
- void initUnits ()

  *Concrete implementation of initiation of units.*
- virtual void turn ()

  *Concrete implementation of turn.*
- virtual void update (Team *)

  *Observer function for updating.*
- virtual void attack ()

  *Concrete implementation of turn.*

### Additional Inherited Members

#### 3.7.1 Detailed Description

Concrete Team that has a number of Units.

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

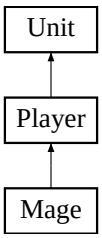
- HumanTeam.h
- HumanTeam.cpp

## 3.8 Mage Class Reference

A concrete Unit; Inherits from Player.

`#include <Mage.h>`

Inheritance diagram for Mage:




### Public Member Functions

- Mage ()

### Additional Inherited Members

#### 3.8.1 Detailed Description

A concrete Unit; Inherits from Player.

**See also**

Player ()

#### 3.8.2 Constructor & Destructor Documentation

##### 3.8.2.1 Mage::Mage ( )

Constructor for Mage class sets the stats and respective "class" of Mage.

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

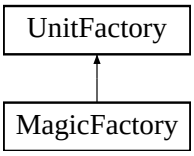
- Mage.h
- Mage.cpp

## 3.9 MagicFactory Class Reference

Concrete Factory that creates a Mage or Elemental object.

`#include <MagicFactory.h>`

Inheritance diagram for MagicFactory:




### Public Member Functions

- virtual Unit * createPlayer ()
  *Concrete implementation of createPlayer, will create a Mage.*
- virtual Unit * createMonster ()
  *Concrete impletation of createMonster, will create an Elemental.*

### Additional Inherited Members

#### 3.9.1 Detailed Description

Concrete Factory that creates a Mage or Elemental object.

**See also**

UnitFactory

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

- MagicFactory.h
- MagicFactory.cpp

## 3.10 Map Class Reference

Enables the concepts of position and movement.

`#include <Map.h>`

### Public Member Functions

- ~Map ()
- Map (char *)
- void printMap ()
- bool Move (int, int, int, int)
- void setMap ()
- int getMapSizeX ()
- int getMapSizeY ()
- char getMapTile (int x, int y)
- void setMapTile (char c, int x, int y)

### 3.10.1 Detailed Description

Enables the concepts of position and movement.

### 3.10.2 Constructor & Destructor Documentation

#### 3.10.2.1 Map::~Map ( )

@ brief Destructor for class map. Fill in Destructor here

#### 3.10.2.2 Map::Map ( char * *fileName* )

@ brief Constructor that takes a filename for reading in map

### 3.10.3 Member Function Documentation

#### 3.10.3.1 int Map::getMapSizeX ( )

@ brief Getter function for maximum x bounds. @ return int containing maximum x value;

#### 3.10.3.2 int Map::getMapSizeY ( )

@ brief Getter function for maximum y bounds. @ return int containing maximum y value;

#### 3.10.3.3 char Map::getMapTile ( int *x*, int *y* )

@ brief Function to retreive a specific char from the map. @ char containing the tile at given coordinates

#### 3.10.3.4 bool Map::Move ( int *x*, int *y*, int *newX*, int *newY* )

@ brief Function to move a specific tile to another location

**Returns**

bool containing whether the movement was successful.

#### 3.10.3.5 void Map::printMap ( )

@ brief Function to display the map in the terminal.

#### 3.10.3.6 void Map::setMap ( )

@ brief

#### 3.10.3.7 void Map::setMapTile ( char *c*, int *x*, int *y* )

@ brief Function to set the tile at specific coordinates.

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

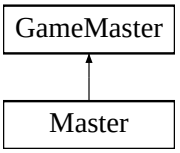
- Map.h
- Map.cpp

## 3.11 Master Class Reference

Controls the flow of operations in the simulation.

`#include <Master.h>`

Inheritance diagram for Master:




### Public Member Functions

- Master ()

  *Constructor for class Master.*
- ~Master ()

  *Destructor for class Master.*
- virtual void attachTeam (Team *inputTeam)

  *Subject function for attatching a team (Observer) to the game.*
- virtual void detachTeam (Team *inputTeam)

  *Subject function for detaching a respective team.*
- virtual bool moveUnit (Unit *inputUnit, string direction)

  *A simplified interface for moving Units, invokes Map.*
- virtual void notify (Team *)

  *Observer function to indicate to Observers to update.*
- virtual void printMap ()

  *Function that invokes the print of map.*
- virtual void addToMap (Unit *inputUnit, int x, int y)

*Adds Unit pointer to given x-y coordinates.*

- virtual vector< int > locateUnit (Unit *inputUnit)

  *Will search map for Unit pointer at given coordinates.*

- virtual vector< int > requestFreeSpace ()

  *Finds an empty space on a map for the purpose.*

- virtual Unit * locateUnit (int row, int col)

  *Retrieves Unit at given coordinates.*

- virtual void removeDestroyedUnits ()

  *Function to ensure that "dead" units in teams are taken out of the game.*

### Additional Inherited Members

### 3.11.1 Detailed Description

Controls the flow of operations in the simulation.

### 3.11.2 Member Function Documentation

#### 3.11.2.1 vector< int > Master::locateUnit ( Unit * *inputUnit* ) [virtual]

Will search map for Unit pointer at given coordinates.

**Returns**

vector containing coordinates of given Unit pointer. Will

Implements GameMaster.

#### 3.11.2.2 Unit * Master::locateUnit ( int *row,* int *col* ) [virtual]

Retrieves Unit at given coordinates.

**Returns**

Unit* at a given row and column coordinate.

Implements GameMaster.

#### 3.11.2.3 bool Master::moveUnit ( Unit * *inputUnit,* string *direction* ) [virtual]

A simplified interface for moving Units, invokes Map.

**Returns**

boolean stating whether movement was correct.

Implements GameMaster.

#### 3.11.2.4 vector< int > Master::requestFreeSpace ( ) [virtual]

Finds an empty space on a map for the purpose.

**Returns**

vector containing empty coordinates on the map.

Implements GameMaster.

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

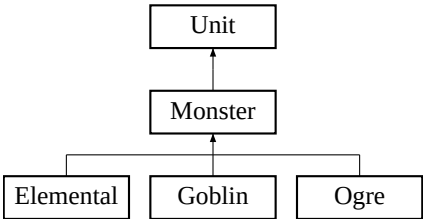
- Master.h
- Master.cpp

## 3.12 Monster Class Reference

Is the class from which all concrete Monsters derive inherites from Unit.

`#include <Monster.h>`

Inheritance diagram for Monster:




### Public Member Functions

- Unit * clone ()

  *Implementation of inherited virtual function.*

### Additional Inherited Members

### 3.12.1 Detailed Description

Is the class from which all concrete Monsters derive inherites from Unit.

**See also**

Unit

### 3.12.2 Member Function Documentation

#### 3.12.2.1 Unit * Monster::clone ( ) [virtual]

Implementation of inherited virtual function.

**Returns**

Unit* containing a deep copy of this object.

Implements Unit.

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

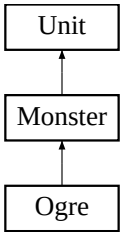
- Monster.h
- Monster.cpp

## 3.13 Ogre Class Reference

A concrete Unit; Inherits from Monster.

`#include <Ogre.h>`

Inheritance diagram for Ogre:




### Public Member Functions

- Ogre ()

### Additional Inherited Members

### 3.13.1 Detailed Description

A concrete Unit; Inherits from Monster.

**See also**

Monster ()

### 3.13.2 Constructor & Destructor Documentation

#### 3.13.2.1 Ogre::Ogre ( )

Constructor for Ogre class sets the stats and respective "class" of Orgre.

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

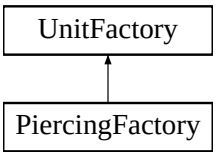
- Ogre.h
- Ogre.cpp

## 3.14 PiercingFactory Class Reference

Concrete factory that creates Thief or Goblin objects.

`#include <PiercingFactory.h>`

Inheritance diagram for PiercingFactory:



### Public Member Functions

- virtual Unit * createPlayer ()

  *Concrete implementation of createPlayer, will create a Theif.*
- virtual Unit * createMonster ()

  *Concrete impletation of createMonster, will create a Goblin.*

### Additional Inherited Members

#### 3.14.1 Detailed Description

Concrete factory that creates Thief or Goblin objects.

**See also**

UnitFactory

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

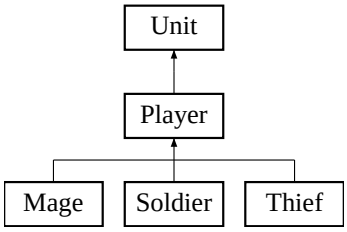
- PiercingFactory.h
- PiercingFactory.cpp

## 3.15 Player Class Reference

Is the class from which all concrete Monsters derive inherites from Unit.

`#include <Player.h>`

Inheritance diagram for Player:




### Public Member Functions

- Unit * clone ()

  *Implementation of inherited virtual function.*

### Additional Inherited Members

#### 3.15.1 Detailed Description

Is the class from which all concrete Monsters derive inherites from Unit.

**See also**

Unit

### 3.15.2 Member Function Documentation

#### 3.15.2.1 Unit * Player::clone ( ) [virtual]

Implementation of inherited virtual function.

**Returns**

Unit* containing a deep copy of this object.

Implements Unit.

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

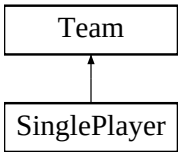
- Player.h
- Player.cpp

## 3.16 SinglePlayer Class Reference

A special case of Team, where there is only one unit.

`#include <SinglePlayer.h>`

Inheritance diagram for SinglePlayer:




### Public Member Functions

- SinglePlayer (GameMaster *gameMaster)

  *Constructor for class SinglePlayer.*
- void initUnits ()

  *Concrete implementation of initiation of units.*
- void update (Team *)

  *Observer function for updating.*
- void attack ()

  *Concrete implementation of turn.*
- void turn ()

  *Concrete implementation of turn.*

### Additional Inherited Members

### 3.16.1 Detailed Description

A special case of Team, where there is only one unit.

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

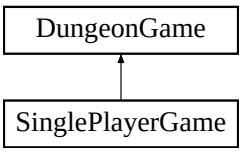
- SinglePlayer.h
- SinglePlayer.cpp

## 3.17 SinglePlayerGame Class Reference

a concrete DungeonGame defining a standard single player game.

`#include <SinglePlayerGame.h>`

Inheritance diagram for SinglePlayerGame:




### Public Member Functions

- void setupGame ()

  *Implementation creating one ComputerTeam and one SinglePlayer team.*
- void beginGame ()

  *Implementation creating one ComputerTeam and one SinglePlayer team.*

### 3.17.1 Detailed Description

a concrete DungeonGame defining a standard single player game.

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

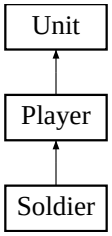
- SinglePlayerGame.h
- SinglePlayerGame.cpp

## 3.18 Soldier Class Reference

A concrete Unit; Inherits from Player.

`#include <Soldier.h>`

Inheritance diagram for Soldier:




### Public Member Functions

- Soldier ()

### Additional Inherited Members

### 3.18.1 Detailed Description

A concrete Unit; Inherits from Player.

**See also**

Player ()

### 3.18.2 Constructor & Destructor Documentation

#### 3.18.2.1 Soldier::Soldier ( )

Constructor for Soldier class sets the stats and respective "class" of Soldier.

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

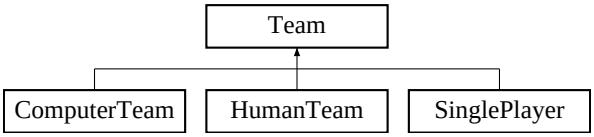
- Soldier.h
- Soldier.cpp

## 3.19 Team Class Reference

Is the class from which all concrete Teams inherit.

`#include <Team.h>`

Inheritance diagram for Team:




### Public Member Functions

- Team (GameMaster *inputGameMaster)

  *Constructor for class Team.*
- virtual ~Team ()

  *Destructor of class Teams.*
- virtual void initUnits ()=0

  *Abtract function fot intitializing the units of a team.*
- virtual void update (Team *)=0

  *Observer functions.*
- virtual void attack ()=0

  *Abstract function for attacking sequence.*
- virtual void turn ()=0

  *Abstract function for turn sequence.*
- void takeDamage (int damage)

  *Function to distribute damage amongst a team.*
- virtual void addUnit (Unit *inputUnit)

  *function to add a Unit to a Team.*

- Unit * getUnitAt (int index)

  *Function to retrieve Unit at specific index.*
- void setGameMaster (GameMaster *)

  *Function to allocate Team to GameMaster.*
- int getSize ()

  *Determines the number of teams in the game.*
- void setMap (Map *)

  *Allocates Map pointer.*

### Protected Attributes

- Map * **map**
- GameMaster * **gameMaster**
- vector< Unit * > **units**

### 3.19.1 Detailed Description

Is the class from which all concrete Teams inherit.

### 3.19.2 Member Function Documentation

#### 3.19.2.1 int Team::getSize ( )

Determines the number of teams in the game.

**Returns**

int containing the number of live teams.

#### 3.19.2.2 Unit * Team::getUnitAt ( int *index* )

Function to retrieve Unit at specific index.

**Returns**

Unit* at given index.

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

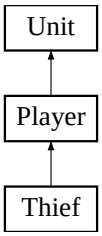
- Team.h
- Team.cpp

## 3.20 Thief Class Reference

A concrete Unit; Inherits from Player.

`#include <Thief.h>`

Inheritance diagram for Thief:




### Public Member Functions

- Thief ()

### Additional Inherited Members

#### 3.20.1 Detailed Description

A concrete Unit; Inherits from Player.

**See also**

Player ()

#### 3.20.2 Constructor & Destructor Documentation

##### 3.20.2.1 Thief::Thief ( )

Constructor for Thief class sets the stats and respective "class" of Thief.

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

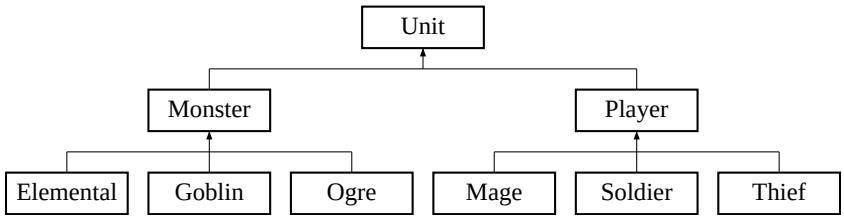
- Thief.h
- Thief.cpp

## 3.21 Unit Class Reference

Is the class from which all concrete Units derive.

`#include <Unit.h>`

Inheritance diagram for Unit:




### Public Member Functions

- virtual ~Unit ()

*Virtual destructor for class Unit.*

- virtual Unit * clone ()=0

  *pure virtual function that allows prototypes of Units to be cloned.*

- int getDamage ()

  *Public interface to damage member variable.*

- int getHealth ()

  *Public interface to health member variable.*

- string getClass ()

  *Public interface to "class" member variable.*

- void **takeDamage** (int inputDamage)

### Protected Member Functions

- void setDamage (int inputDamage)

  *Protected interface to modify damage member.*

- void setHealth (int inputHealth)

  *Protected interface to modify health member.*

- void setClass (string inputClass)

  *Protected interface to modify "class" member.*

### Protected Attributes

- string **unitClass**
- int **damage**
- int **health**

### Friends

- class **Team**
- class **SinglePlayer**

## 3.21.1 Detailed Description

Is the class from which all concrete Units derive.

## 3.21.2 Member Function Documentation

### 3.21.2.1 virtual Unit* Unit::clone ( ) [pure virtual]

pure virtual function that allows prototypes of Units to be cloned.

**Returns**

a new Unit cloned from member variables.

Implemented in Monster, and Player.

#### 3.21.2.2 string Unit::getClass ( )

Public interface to "class" member variable.

**Returns**

string containing the class of object.

#### 3.21.2.3 int Unit::getDamage ( )

Public interface to damage member variable.

**Returns**

int containing value of damage.

#### 3.21.2.4 int Unit::getHealth ( )

Public interface to health member variable.

**Returns**

int containing value of health.

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

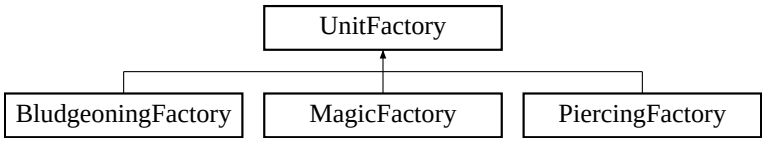
- Unit.h
- Unit.cpp

## 3.22 UnitFactory Class Reference

Defines an interface for creating type Player and type Monster.

`#include <UnitFactory.h>`

Inheritance diagram for UnitFactory:




### Public Member Functions

- UnitFactory ()

  *Constructor for abstract class UnitFactory.*
- virtual ~UnitFactory ()

  *Virtual destructor for class Unit.*
- virtual Unit * createPlayer ()=0

  *Pure virtual function for creating Players.*
- virtual Unit * createMonster ()=0

  *Pure virtual function for creating Mosters.*

### Protected Attributes

- Unit * **modelMonster**
- Unit * **modelPlayer**

## 3.22.1 Detailed Description

Defines an interface for creating type Player and type Monster.

## 3.22.2 Member Function Documentation

### 3.22.2.1 virtual Unit* UnitFactory::createMonster ( ) [pure virtual]

Pure virtual function for creating Mosters.

**Returns**

Unit* of type concrete Monsters

Implemented in MagicFactory, PiercingFactory, and BludgeoningFactory.

### 3.22.2.2 virtual Unit* UnitFactory::createPlayer ( ) [pure virtual]

Pure virtual function for creating Players.

**Returns**

Unit* of type concrete Player

Implemented in MagicFactory, PiercingFactory, and BludgeoningFactory.

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

- UnitFactory.h
- UnitFactory.cpp