Civ Zero

Generated by Doxygen 1.12.0

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

Chapter 4

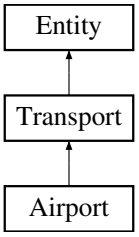
Class Documentation

4.1 Airport Class Reference

Represents an airport entity within the game.

`#include <Airport.h>`

Inheritance diagram for Airport:



Public Member Functions

- Airport (EntityConfig ec, Size size, int xPos, int yPos)
  Constructs an Airport entity with specified attributes.
- Airport (Airport ∗airport)
  Copy constructor for the Airport class.
- virtual ∼**Airport** ()
  Destructor for the Airport class.
- void update ()
  Updates the state of the airport entity.
- Entity ∗ clone ()
  Creates a clone of the airport entity.

Public Member Functions inherited from Transport

- Transport (EntityConfig ec, Size size, int xPos, int yPos)
  Constructs a Transport entity with specified attributes.
- Transport (Transport ∗transport)
  Copy constructor for the Transport class.
- virtual ∼**Transport** ()
  Virtual destructor for the Transport class.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).
- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.1.1 Detailed Description

Represents an airport entity within the game.

The Airport class manages the properties and behavior of airport entities, including their position, size, and functionality related to transportation.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 Airport() [1/2]

```
Airport::Airport (
            EntityConfig ec,
            Size size,
            int xPos,
            int yPos)
```

Constructs an Airport entity with specified attributes.

Parameters

ec	Configuration settings for the entity.
size	Size of the airport entity.
xPos	X-coordinate position of the airport.
yPos	Y-coordinate position of the airport.

4.1.2.2 Airport() [2/2]

```
Airport::Airport (
            Airport * airport)
```

Copy constructor for the Airport class.

Creates a new Airport entity by copying the attributes of an existing Airport.

Parameters

airport	Pointer to the Airport object to be copied.

4.1.3 Member Function Documentation

4.1.3.1 clone()

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

Creates a clone of the airport entity.

Returns

A pointer to the cloned Airport entity.

Implements Transport.

4.1.3.2 update()

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

Updates the state of the airport entity.

Implements Transport.

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

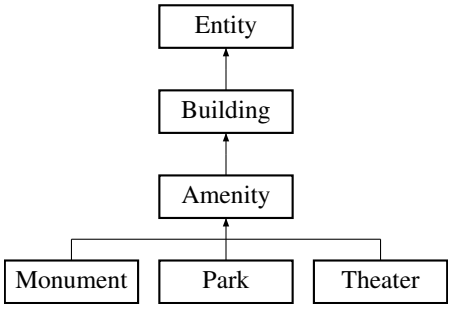
- src/entities/transport/Airport.h
- src/entities/transport/Airport.cpp

4.2 Amenity Class Reference

Represents an abstract base class for amenities within the game, such as parks, shops, or other facilities.

`#include <Amenity.h>`

Inheritance diagram for Amenity:



Public Member Functions

- Amenity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Amenity with specified attributes.
- Amenity (Amenity ∗amenity)

  Copy constructor for the Amenity class.
- virtual ∼**Amenity** ()

  Virtual destructor for the Amenity class.
- virtual void update ()=0

  Updates the amenity's state. Needs to be implemented in derived classes.
- virtual Entity ∗ clone ()=0

  Clones the amenity. Needs to be implemented in derived classes.

Public Member Functions inherited from Building

- Building (EntityConfig ec, Size size, int xPos, int yPos)

  Parameterized constructor for the Building class.
- Building (Building ∗building)

  Copy constructor for the Building class.
- virtual ∼Building ()

  Destructor for the Building class.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity ∗entity)

  Copy constructor for the Entity class.
- virtual ∼**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity ∗entity)

Checks if another entity is within the effect radius of this entity.

- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity ∗entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity ∗entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity ∗ > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).
- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.2.1 Detailed Description

Represents an abstract base class for amenities within the game, such as parks, shops, or other facilities.

This class inherits from Building and provides a foundation for various types of amenities that can be added to the game. It includes basic constructors, a destructor, and pure virtual methods for update and cloning operations.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Amenity() [1/2]

```
Amenity::Amenity (
            EntityConfig ec,
            Size size,
            int xPos,
            int yPos)
```

Constructs an Amenity with specified attributes.

Parameters

ec	Configuration containing resource consumption and other properties.
size	Size of the amenity.
xPos	X-coordinate position of the amenity.
yPos	Y-coordinate position of the amenity.

4.2.2.2 Amenity() [2/2]

```
Amenity::Amenity (
            Amenity * amenity)
```

Copy constructor for the Amenity class.

Creates a new Amenity by copying the attributes of an existing Amenity.

Parameters

amenity	Pointer to the Amenity object to be copied.

4.2.3 Member Function Documentation

4.2.3.1 clone()

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

Clones the amenity. Needs to be implemented in derived classes.

Returns

A pointer to the cloned Amenity.

Implements Building.

Implemented in Monument, Park, and Theater.

4.2.3.2 update()

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

Updates the amenity's state. Needs to be implemented in derived classes.

Implements Building.

Implemented in Monument, Park, and Theater.

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

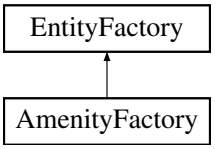
- src/entities/building/amenity/Amenity.h
- src/entities/building/amenity/Amenity.cpp

4.3 AmenityFactory Class Reference

Factory class for creating various amenities, including parks, theaters, and monuments.

`#include <AmenityFactory.h>`

Inheritance diagram for AmenityFactory:



Public Member Functions

- **AmenityFactory** ()

  Default constructor for AmenityFactory.
- ~**AmenityFactory** ()

  Destructor for AmenityFactory.
- virtual Entity * createEntity (EntityType type, Size size, int xPos, int yPos)

  Creates an amenity of the specified type and size at the given position.

Public Member Functions inherited from EntityFactory

- **EntityFactory** ()

  Default constructor for EntityFactory.
- virtual ~**EntityFactory** ()

  Virtual destructor for EntityFactory.

4.3.1 Detailed Description

Factory class for creating various amenities, including parks, theaters, and monuments.

Inherits from EntityFactory and provides methods to create different-sized amenities (small, medium, and large) based on the specified type at given coordinates.

4.3.2 Member Function Documentation

4.3.2.1 createEntity()

```
Entity * AmenityFactory::createEntity (
            EntityType type,
            Size size,
            int xPos,
            int yPos)  [virtual]
```

Creates an amenity of the specified type and size at the given position.

Parameters

type	The type of amenity to create (e.g., Park, Theater, Monument).
size	The size of the amenity (small, medium, or large).
xPos	The x-coordinate for the amenity's position.
yPos	The y-coordinate for the amenity's position.

Returns

A pointer to the created Entity.

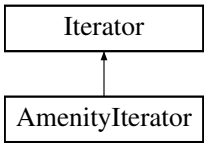
Implements EntityFactory.

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

- src/factory/building/AmenityFactory.h
- src/factory/building/AmenityFactory.cpp

4.4 AmenityIterator Class Reference

Inheritance diagram for AmenityIterator:



Public Member Functions

- **AmenityIterator** ()

  Construct a new Amenity Iterator:: Amenity Iterator object.
- ∼**AmenityIterator** ()

  Destroy the Amenity Iterator:: Amenity Iterator object.
- AmenityIterator (std::vector< std::vector< Entity * > > &grid)

  Construct a new Amenity Iterator:: Amenity Iterator object.
- void first () override

  Sets the iterator to the first unvisited Amenity.
- void next () override

  Advances to the next unvisited Amenity.
- bool hasNext () override

  Checks if there is another unvisited Amenity.
- Entity * current () override

  Returns the current Amenity.

Public Member Functions inherited from Iterator

- **Iterator** ()

  Construct a new Iterator object, initializing row and column to zero.
- virtual ~**Iterator** ()

  Destroy the Iterator object.
- **Iterator** (std::vector< std::vector< Entity * > > &grid)
- virtual int getRow ()

  Get the current row index of the iterator.
- virtual int getCol ()

  Get the current column index of the iterator.

Additional Inherited Members

Protected Member Functions inherited from Iterator

- bool isVisited (Entity *entity)

  Check if the specified entity has been visited.
- void markVisited (Entity *entity)

  Mark the specified entity as visited.

Protected Attributes inherited from Iterator

- std::vector< std::vector< Entity * > > **grid**
- std::vector< std::vector< Entity * > >::iterator **currRow**
- std::vector< Entity * >::iterator **curr**
- int **row**
- int **col**
- std::unordered_set< Entity * > **visitedEntities**

4.4.1 Constructor & Destructor Documentation

4.4.1.1 AmenityIterator()

```
AmenityIterator::AmenityIterator (
            std::vector< std::vector< Entity * > > & grid)
```

Construct a new Amenity Iterator:: Amenity Iterator object.

Parameters

grid	

4.4.2 Member Function Documentation

4.4.2.1 current()

```
Entity * AmenityIterator::current ()  [override], [virtual]
```

Returns the current Amenity.

Returns

Entity*

Implements Iterator.

4.4.2.2 first()

```
void AmenityIterator::first ()  [override], [virtual]
```

Sets the iterator to the first unvisited Amenity.

Implements Iterator.

4.4.2.3 hasNext()

```
bool AmenityIterator::hasNext ()  [override], [virtual]
```

Checks if there is another unvisited Amenity.

Returns

true if there is another unvisited Amenity, false otherwise

Implements Iterator.

4.4.2.4 next()

```
void AmenityIterator::next ()  [override], [virtual]
```

Advances to the next unvisited Amenity.

Implements Iterator.

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

- src/iterators/building/amenity/AmenityIterator.h
- src/iterators/building/amenity/AmenityIterator.cpp

4.5 AmenityManager Class Reference

Responsible for managing amenities by creating and configuring them based on specified types and sizes.

`#include <AmenityManager.h>`

Public Member Functions

- **AmenityManager** ()

  Constructs a new AmenityManager.
- ~**AmenityManager** ()

  Destroys the AmenityManager and releases any allocated resources.
- void buildAmenity (EntityType type, Size size, int xPos, int yPos)

  Builds an amenity of a specified type and size at a given position.

4.5.1 Detailed Description

Responsible for managing amenities by creating and configuring them based on specified types and sizes.

4.5.2 Member Function Documentation

4.5.2.1 buildAmenity()

```
void AmenityManager::buildAmenity (
            EntityType type,
            Size size,
            int xPos,
            int yPos)
```

Builds an amenity of a specified type and size at a given position.

Parameters

type	The type of the amenity to be created.
size	The size specification for the amenity.
xPos	The x-coordinate for the amenity's position.
yPos	The y-coordinate for the amenity's position.

Returns

A pointer to the created Amenity object.

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

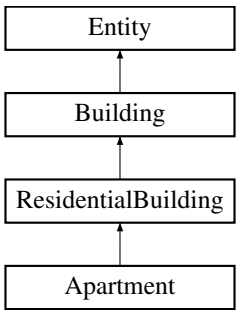
- src/managers/AmenityManager.h
- src/managers/AmenityManager.cpp

4.6 Apartment Class Reference

Represents an apartment building within the game.

`#include <Apartment.h>`

Inheritance diagram for Apartment:



Public Member Functions

- Apartment (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Apartment with specified attributes.
- Apartment (Apartment ∗entity)

  Copy constructor for the Apartment class.
- virtual ∼**Apartment** ()

  Destructor for the Apartment class.
- Entity ∗ clone ()

  Creates a clone of the apartment.

Public Member Functions inherited from ResidentialBuilding

- ResidentialBuilding (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a ResidentialBuilding with specified attributes.
- ResidentialBuilding (ResidentialBuilding ∗entity)

  Copy constructor for the ResidentialBuilding class.
- virtual ∼**ResidentialBuilding** ()

  Destructor for the ResidentialBuilding class.
- void update ()

  Updates the residential building's state.
- void **reset** ()

  Resets the satisfaction factors for the building.
- void **calculateSatisfaction** ()

  Calculates the satisfaction level based on nearby entities.
- float getSatisfaction ()

  Gets the satisfaction level of the building.
- void updateAirport (Entity ∗entity)

  Updates the effect of a nearby airport.
- void updateBusStop (Entity ∗entity)

  Updates the effect of a nearby bus stop.
- void updateTrainStation (Entity ∗entity)

Updates the effect of a nearby train station.

- void updateFactory (Entity *entity)

  Updates the effect of a nearby factory.
- void updateShoppingMall (Entity *entity)

  Updates the effect of a nearby shopping mall.
- void updateOffice (Entity *entity)

  Updates the effect of a nearby office.
- void updateHospital (Entity *entity)

  Updates the effect of a nearby hospital.
- void updatePoliceStation (Entity *entity)

  Updates the effect of a nearby police station.
- void updateSchool (Entity *entity)

  Updates the effect of a nearby school.
- void updateAmenity (Entity *entity)

  Updates the effect of a nearby amenity.
- void updateUtility (Entity *entity)

  Updates the effect of a nearby utility.
- void updateIndustry (Entity *entity)

  Updates the effect of a nearby industry.
- int getCapacity ()

  Gets the capacity of the residential building.
- void setCapacity (int capacity)

  Sets the capacity of the residential building.

Public Member Functions inherited from Building

- Building (EntityConfig ec, Size size, int xPos, int yPos)

  Parameterized constructor for the Building class.
- Building (Building *building)

  Copy constructor for the Building class.
- virtual ~Building ()

  Destructor for the Building class.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.

- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

Width of the entity.

- int **height**

  Height of the entity.

- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).

- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).

- Size **size**

  Size object representing the entity's dimensions.

- EntityType **type**

  The type of entity.

- State * **state**

  Pointer to the current state of the entity.

- int **revenue**

  Revenue generated by the entity.

- float **electricityConsumption**

  Electricity consumption of the entity.

- float **waterConsumption**

  Water consumption of the entity.

- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.6.1 Detailed Description

Represents an apartment building within the game.

The Apartment class is a type of ResidentialBuilding, with attributes and behaviors specific to apartment-style residences. This class includes constructors, a destructor, and an implementation of the clone function.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 Apartment() [1/2]

```
Apartment::Apartment (
            EntityConfig ec,
            Size size,
            int xPos,
            int yPos)
```

Constructs an Apartment with specified attributes.

Parameters

ec	Configuration containing resource consumption and other properties.
size	Size of the apartment.
xPos	X-coordinate position of the apartment.
yPos	Y-coordinate position of the apartment.

4.6.2.2 Apartment() [2/2]

```
Apartment::Apartment (
            Apartment * entity)
```

Copy constructor for the Apartment class.

Creates a new Apartment by copying the attributes of an existing Apartment.

Parameters

entity	Pointer to the Apartment object to be copied.

4.6.3 Member Function Documentation

4.6.3.1 clone()

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

Creates a clone of the apartment.

Returns

A pointer to the cloned Apartment.

Implements ResidentialBuilding.

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

- src/entities/building/residential/Apartment.h
- src/entities/building/residential/Apartment.cpp

4.7 BSPPartitioner Class Reference

Handles Binary Space Partitioning (BSP) for grid partitioning. Splits a grid into smaller rectangular rooms separated by gaps.

```
#include <BSPPartitioner.h>
```

Public Member Functions

- BSPPartitioner (int gridWidth, int gridHeight, int minWidth, int minHeight, int gap)
  Constructs a BSPPartitioner with grid dimensions, minimum room size, and gap width.
- void **partition** ()
  Partitions the grid into rooms separated by gaps.
- const std::vector< Rectangle > & getRooms () const
  Retrieves the list of partitioned rooms.
- const std::vector< Rectangle > & getGaps () const
  Retrieves the list of gap areas.

4.7.1 Detailed Description

Handles Binary Space Partitioning (BSP) for grid partitioning. Splits a grid into smaller rectangular rooms separated by gaps.

4.7.2 Constructor & Destructor Documentation

4.7.2.1 BSPPartitioner()

```
BSPPartitioner::BSPPartitioner (
            int gridWidth,
            int gridHeight,
            int minWidth,
            int minHeight,
            int gap)
```

Constructs a BSPPartitioner with grid dimensions, minimum room size, and gap width.

Parameters

gridWidth	Width of the grid.
gridHeight	Height of the grid.
minWidth	Minimum width of a room.
minHeight	Minimum height of a room.
gap	Width of the gap between partitions.

4.7.3 Member Function Documentation

4.7.3.1 getGaps()

```
const std::vector< Rectangle > & BSPPartitioner::getGaps () const
```

Retrieves the list of gap areas.

Returns

Vector of rectangles representing gaps.

4.7.3.2 getRooms()

```
const std::vector< Rectangle > & BSPPartitioner::getRooms () const
```

Retrieves the list of partitioned rooms.

Returns

Vector of rectangles representing rooms.

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

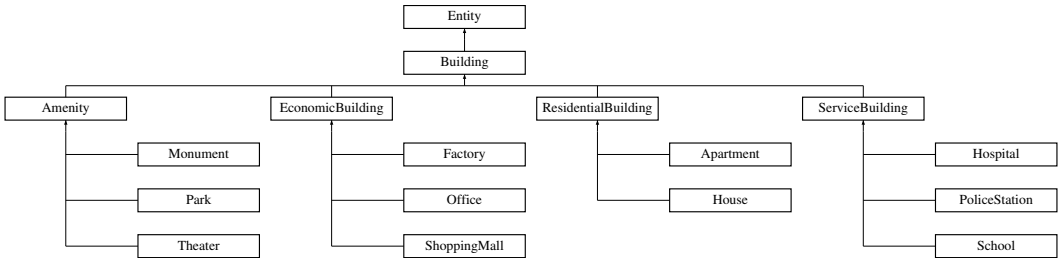
- src/utils/BSPPartitioner.h
- src/utils/BSPPartitioner.cpp

4.8 Building Class Reference

Abstract class representing a general building entity in the city builder/manager game.

`#include <Building.h>`

Inheritance diagram for Building:



Public Member Functions

- Building (EntityConfig ec, Size size, int xPos, int yPos)

  Parameterized constructor for the Building class.
- Building (Building ∗building)

  Copy constructor for the Building class.
- virtual ∼Building ()

  Destructor for the Building class.
- virtual void update ()=0

  Updates the state of the building entity.
- virtual Entity ∗ clone ()=0

  Clones the building entity.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity ∗entity)

  Copy constructor for the Entity class.
- virtual ∼**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity ∗entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.

- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

X-coordinate of the entity's position (bottom left corner).

- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).

- Size **size**

  Size object representing the entity's dimensions.

- EntityType **type**

  The type of entity.

- State * **state**

  Pointer to the current state of the entity.

- int **revenue**

  Revenue generated by the entity.

- float **electricityConsumption**

  Electricity consumption of the entity.

- float **waterConsumption**

  Water consumption of the entity.

- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.8.1 Detailed Description

Abstract class representing a general building entity in the city builder/manager game.

This class serves as the base class for all types of buildings. It inherits from the Entity class and provides an interface for common building functionalities.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 Building() [1/2]

```
Building::Building (
            EntityConfig ec,
            Size size,
            int xPos,
            int yPos)
```

Parameterized constructor for the Building class.

Parameters

ec	The configuration object containing general entity properties.
size	The size of the building entity.
xPos	The x-coordinate position of the building on the map.
yPos	The y-coordinate position of the building on the map.

Initializes a new instance of the Building class with specific values.

4.8.2.2 Building() [2/2]

```
Building::Building (
            Building * building)
```

Copy constructor for the Building class.

Parameters

building	A pointer to an existing Building object to copy from.

Creates a new Building instance as a copy of the provided object.

4.8.2.3 ~Building()

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

Destructor for the Building class.

Ensures proper cleanup of resources when a Building object is destroyed.

4.8.3 Member Function Documentation

4.8.3.1 clone()

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

Clones the building entity.

Returns a deep copy of the current Building object.

Returns

A pointer to the newly cloned Building entity.

Implements Entity.

Implemented in Amenity, Apartment, EconomicBuilding, Factory, Hospital, House, Monument, Office, Park, PoliceStation, ResidentialBuilding, School, ServiceBuilding, ShoppingMall, and Theater.

4.8.3.2 update()

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

Updates the state of the building entity.

A pure virtual function that must be implemented by derived classes to handle changes in the building's state.

Implements Entity.

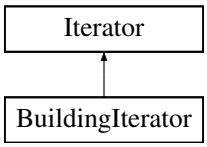
Implemented in Amenity, EconomicBuilding, Factory, Hospital, Monument, Office, Park, PoliceStation, ResidentialBuilding, School, ServiceBuilding, ShoppingMall, and Theater.

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

- src/entities/building/base/Building.h
- src/entities/building/base/Building.cpp

4.9 BuildingIterator Class Reference

Inheritance diagram for BuildingIterator:



Public Member Functions

- **BuildingIterator** ()

  Construct a new Building Iterator:: Building Iterator object.
- ~**BuildingIterator** ()

  Destroy the Building Iterator:: Building Iterator object.
- BuildingIterator (std::vector< std::vector< Entity * > > &grid)

  Construct a new Building Iterator:: Building Iterator object.
- void first ()

  Sets the iterator to the first unvisited Building.
- void next ()

  Advances to the next unvisited Building.
- bool hasNext ()

  Checks if there is another unvisited Building.
- Entity * current ()

  Returns the current Building.

Public Member Functions inherited from Iterator

- **Iterator** ()

  Construct a new Iterator object, initializing row and column to zero.
- virtual ~**Iterator** ()

  Destroy the Iterator object.
- **Iterator** (std::vector< std::vector< Entity * > > &grid)
- virtual int getRow ()

  Get the current row index of the iterator.
- virtual int getCol ()

  Get the current column index of the iterator.

Additional Inherited Members

Protected Member Functions inherited from Iterator

- bool isVisited (Entity *entity)

  Check if the specified entity has been visited.
- void markVisited (Entity *entity)

  Mark the specified entity as visited.

Protected Attributes inherited from Iterator

- std::vector< std::vector< Entity * > > **grid**
- std::vector< std::vector< Entity * > >::iterator **currRow**
- std::vector< Entity * >::iterator **curr**
- int **row**
- int **col**
- std::unordered_set< Entity * > **visitedEntities**

4.9.1 Constructor & Destructor Documentation

4.9.1.1 BuildingIterator()

```
BuildingIterator::BuildingIterator (
            std::vector< std::vector< Entity * > > & grid)
```

Construct a new Building Iterator:: Building Iterator object.

Parameters

grid	

4.9.2 Member Function Documentation

4.9.2.1 current()

```
Entity * BuildingIterator::current ()  [virtual]
```

Returns the current Building.

Returns

Entity*

Implements Iterator.

4.9.2.2 first()

```
void BuildingIterator::first ()  [virtual]
```

Sets the iterator to the first unvisited Building.

Implements Iterator.

4.9.2.3 hasNext()

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

Checks if there is another unvisited Building.

Returns

true if there is another unvisited Building, false otherwise

Implements Iterator.

4.9.2.4 next()

```
void BuildingIterator::next ()  [virtual]
```

Advances to the next unvisited Building.

Implements Iterator.

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

- src/iterators/building/BuildingIterator.h
- src/iterators/building/BuildingIterator.cpp

4.10 BuildingManager Class Reference

Manages the construction of residential buildings in the city.

```
#include <BuildingManager.h>
```

Public Member Functions

- **BuildingManager** ()
  Constructs a new BuildingManager object.
- ~**BuildingManager** ()
  Destroys the BuildingManager object.
- bool buildBuilding (EntityType type, Size size, int x, int y)
  Builds a residential building of specified type and size at given coordinates.

4.10.1 Detailed Description

Manages the construction of residential buildings in the city.

4.10.2 Member Function Documentation

4.10.2.1 buildBuilding()

```
bool BuildingManager::buildBuilding (
            EntityType type,
            Size size,
            int x,
            int y)
```

Builds a residential building of specified type and size at given coordinates.

Parameters

type	The type of the residential building to build.
size	The size of the building (SMALL, MEDIUM, LARGE).
x	The x-coordinate where the building should be placed.
y	The y-coordinate where the building should be placed.

Returns

true if the building was successfully created and added to the city.

false if building creation failed.

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

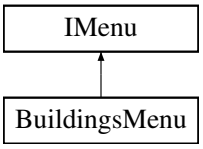
- src/managers/BuildingManager.h
- src/managers/BuildingManager.cpp

4.11 BuildingsMenu Class Reference

Provides a menu interface for managing buildings in the game.

`#include <BuildingsMenu.h>`

Inheritance diagram for BuildingsMenu:



Public Member Functions

- BuildingsMenu ()

  Constructs a BuildingsMenu object.
- ~BuildingsMenu ()

  Destructor for BuildingsMenu.
- void display () const override

  Displays the Buildings menu.
- void handleInput () override

  Handles user input in the Buildings menu.

Public Member Functions inherited from IMenu

- **IMenu** ()=default

  Default constructor for IMenu.
- IMenu (std::string heading)

  Constructor to initialize a menu with a specified heading.
- virtual ~**IMenu** ()=default

  Virtual destructor for IMenu.
- void setHeading (const std::string &heading)

  Sets the heading of the menu.

Additional Inherited Members

Protected Member Functions inherited from IMenu

- std::string repeat (const std::string &str, int times) const

  Utility function to repeat a string multiple times.
- int calculateMaxWidth (const std::string &menuHeading, const std::vector< Section > &sections) const

  Calculates the maximum width required for the menu.
- void printTopBorder (int width) const

  Prints the top border of the menu using box-drawing characters.
- void printBottomBorder (int width) const

  Prints the bottom border of the menu using box-drawing characters.
- void printSectionDivider (int width) const

  Prints a section divider in the menu using box-drawing characters.
- void printDoubleLineDivider (int width) const

  Prints a double-line divider for the main heading of the menu.
- std::string centerText (const std::string &text, int width) const

  Centers text within a specified width using space padding.
- std::string centerTextWithChar (const std::string &text, int width, const std::string &padChar) const

  Centers text within a specified width using a custom character for padding.
- void displayMenu () const

  Displays the formatted menu, including sections and options.
- void **displayChoicePrompt** () const

  Displays the choice prompt for user input.
- void displayChoiceMessagePrompt (const std::string &message) const

  Displays a custom message prompt for user input.
- void **displayInvalidChoice** () const

  Displays an error message when the user makes an invalid choice.
- void displayErrorMessage (const std::string &message) const

  Displays a general error message.
- void displaySuccessMessage (const std::string &message) const

  Displays a success message in green color.
- void **displayPressEnterToContinue** () const

  Displays a message asking the user to press Enter to continue.
- void **clearScreen** () const

  Clears the terminal screen.
- std::string stripColorCodes (const std::string &input) const

  Strips ANSI color codes from a string.
- virtual void displayAvailablePositions (const std::vector< std::vector< int > > &positions) const

  Displays available positions on the city grid for an entity.

Static Protected Member Functions inherited from IMenu

- static char indexToExtendedChar (int index)

  Converts a numeric index (0-99) to a single character in an extended set.
- static std::string coordinatesToLabel (int x, int y)

  Converts x and y coordinates to a labeled string (e.g., "A, 1").

Protected Attributes inherited from IMenu

- std::vector< Section > **sections**

  List of sections contained in the menu.
- std::string **menuHeading**

  The heading/title of the menu.
- bool **hasExited**

  Flag indicating if the menu has been exited.
- CityManager **cityManager**

  Manager for city-related operations.
- bool **displayResources**

  Flag indicating whether to display resources in the menu.
- bool **isInfoMenu**

  Flag indicating whether to display option numbers.

Static Protected Attributes inherited from IMenu

- static const char * **RESET** = "\033[0m"

  ANSI color codes and styles for use in all menus.
- static const char * **BOLD_WHITE** = "\033[1;37m"
- static const char * **NORMAL_WHITE** = "\033[0;37m"
- static const char * **DARK_GRAY** = "\033[1;30m"
- static const char * **BOLD_YELLOW** = "\033[1;33m"
- static const char * **BOLD_GREEN** = "\033[1;32m"
- static const char * **BOLD_RED** = "\033[1;31m"
- static const char * **BOLD_CYAN** = "\033[1;36m"
- static const char * **BLUE** = "\033[34m"

4.11.1 Detailed Description

Provides a menu interface for managing buildings in the game.

The BuildingsMenu class allows players to buy, sell, and view statistics for different types of buildings. It includes methods for displaying the menu, handling user input, and navigating to relevant submenus or actions.

4.11.2 Constructor & Destructor Documentation

4.11.2.1 BuildingsMenu()

```
BuildingsMenu::BuildingsMenu ()
```

Constructs a BuildingsMenu object.

Constructs the BuildingsMenu and initializes its sections. The menu is divided into sections for buying, selling, and viewing stats of buildings.

Initializes the menu sections with options for buying, demolishing, and viewing building statistics.

4.11.2.2 ~BuildingsMenu()

```
BuildingsMenu::~BuildingsMenu ()
```

Destructor for BuildingsMenu.

Destructor for BuildingsMenu. Cleans up any resources used by the menu.

Cleans up any resources or memory used by the BuildingsMenu.

4.11.3 Member Function Documentation

4.11.3.1 display()

```
void BuildingsMenu::display () const  [override], [virtual]
```

Displays the Buildings menu.

Displays the BuildingsMenu to the user. Renders the options and sections using the inherited displayMenu() method.

Overrides the display method from IMenu to render the menu options and sections to the user.

Implements IMenu.

4.11.3.2 handleInput()

```
void BuildingsMenu::handleInput ()  [override], [virtual]
```

Handles user input in the Buildings menu.

Handles user input for the Buildings menu. Responds to user choices by navigating to the appropriate submenu or viewing stats.

Processes the user's input to navigate through different options for buying, selling, or viewing building statistics, updating the current menu as needed.

Implements IMenu.

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

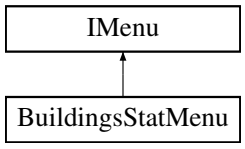
- src/menus/buildings/BuildingsMenu.h
- src/menus/buildings/BuildingsMenu.cpp

4.12 BuildingsStatMenu Class Reference

Provides a menu interface for displaying detailed statistics of various building types.

```
#include <BuildingsStatMenu.h>
```

Inheritance diagram for BuildingsStatMenu:



Public Member Functions

- BuildingsStatMenu (std::vector< EntityType > types)

  Constructs a BuildingsStatMenu object.
- ~BuildingsStatMenu ()

  Destructor for BuildingsStatMenu.
- void display () const override

  Displays the entity type selection menu.
- void handleInput () override

  Handles user input to select a building type and display its statistics.

Public Member Functions inherited from IMenu

- **IMenu** ()=default

  Default constructor for IMenu.
- IMenu (std::string heading)

  Constructor to initialize a menu with a specified heading.
- virtual ~**IMenu** ()=default

  Virtual destructor for IMenu.
- void setHeading (const std::string &heading)

  Sets the heading of the menu.

Additional Inherited Members

Protected Member Functions inherited from IMenu

- std::string repeat (const std::string &str, int times) const

  Utility function to repeat a string multiple times.
- int calculateMaxWidth (const std::string &menuHeading, const std::vector< Section > &sections) const

  Calculates the maximum width required for the menu.
- void printTopBorder (int width) const

  Prints the top border of the menu using box-drawing characters.
- void printBottomBorder (int width) const

  Prints the bottom border of the menu using box-drawing characters.
- void printSectionDivider (int width) const

  Prints a section divider in the menu using box-drawing characters.
- void printDoubleLineDivider (int width) const

  Prints a double-line divider for the main heading of the menu.
- std::string centerText (const std::string &text, int width) const

  Centers text within a specified width using space padding.
- std::string centerTextWithChar (const std::string &text, int width, const std::string &padChar) const

  Centers text within a specified width using a custom character for padding.
- void displayMenu () const

  Displays the formatted menu, including sections and options.
- void **displayChoicePrompt** () const

  Displays the choice prompt for user input.
- void displayChoiceMessagePrompt (const std::string &message) const

  Displays a custom message prompt for user input.
- void **displayInvalidChoice** () const

Displays an error message when the user makes an invalid choice.

- void displayErrorMessage (const std::string &message) const

  Displays a general error message.
- void displaySuccessMessage (const std::string &message) const

  Displays a success message in green color.
- void **displayPressEnterToContinue** () const

  Displays a message asking the user to press Enter to continue.
- void **clearScreen** () const

  Clears the terminal screen.
- std::string stripColorCodes (const std::string &input) const

  Strips ANSI color codes from a string.
- virtual void displayAvailablePositions (const std::vector< std::vector< int > > &positions) const

  Displays available positions on the city grid for an entity.

Static Protected Member Functions inherited from IMenu

- static char indexToExtendedChar (int index)

  Converts a numeric index (0-99) to a single character in an extended set.
- static std::string coordinatesToLabel (int x, int y)

  Converts x and y coordinates to a labeled string (e.g., "A, 1").

Protected Attributes inherited from IMenu

- std::vector< Section > **sections**

  List of sections contained in the menu.
- std::string **menuHeading**

  The heading/title of the menu.
- bool **hasExited**

  Flag indicating if the menu has been exited.
- CityManager **cityManager**

  Manager for city-related operations.
- bool **displayResources**

  Flag indicating whether to display resources in the menu.
- bool **isInfoMenu**

  Flag indicating whether to display option numbers.

Static Protected Attributes inherited from IMenu

- static const char * **RESET** = "\033[0m"

  ANSI color codes and styles for use in all menus.
- static const char * **BOLD_WHITE** = "\033[1;37m"
- static const char * **NORMAL_WHITE** = "\033[0;37m"
- static const char * **DARK_GRAY** = "\033[1;30m"
- static const char * **BOLD_YELLOW** = "\033[1;33m"
- static const char * **BOLD_GREEN** = "\033[1;32m"
- static const char * **BOLD_RED** = "\033[1;31m"
- static const char * **BOLD_CYAN** = "\033[1;36m"
- static const char * **BLUE** = "\033[34m"

4.12.1 Detailed Description

Provides a menu interface for displaying detailed statistics of various building types.

The BuildingsStatMenu class allows players to view comprehensive statistics for selected building types, including cost, utility consumption, and other attributes. The menu displays options to select a building type and shows the associated statistics in a formatted layout.

4.12.2 Constructor & Destructor Documentation

4.12.2.1 BuildingsStatMenu()

```
BuildingsStatMenu::BuildingsStatMenu (
            std::vector< EntityType > types)
```

Constructs a BuildingsStatMenu object.

Constructor for BuildingsStatMenu. Initializes with a list of entity types to display statistics for.

Initializes the menu with a provided list of entity types for which statistics can be displayed.

Parameters

types	A vector containing the entity types to display statistics for.
types	List of entity types to be shown in the stats menu.

4.12.2.2 ~BuildingsStatMenu()

```
BuildingsStatMenu::~BuildingsStatMenu ()
```

Destructor for BuildingsStatMenu.

Cleans up any resources or memory used by the BuildingsStatMenu.

4.12.3 Member Function Documentation

4.12.3.1 display()

```
void BuildingsStatMenu::display () const  [override], [virtual]
```

Displays the entity type selection menu.

Displays the menu allowing the user to select a building type to view stats.

Overrides the display method from IMenu to render the list of available building types for which statistics can be viewed.

Implements IMenu.

4.12.3.2 handleInput()

```
void BuildingsStatMenu::handleInput () [override], [virtual]
```

Handles user input to select a building type and display its statistics.

Handles user input for selecting a building type or navigating back.

Processes user input, allowing navigation between building types or returning to previous menus.

Implements IMenu.

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

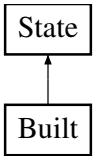
- src/menus/buildings/BuildingsStatMenu.h
- src/menus/buildings/BuildingsStatMenu.cpp

4.13 Built Class Reference

Represents the built state of an entity.

```
#include <Built.h>
```

Inheritance diagram for Built:



Public Member Functions

- Built (int buildTime)

  Constructs a Built state with the specified build time.
- Built (Built *built)

  Copy constructor for the Built class.
- ~**Built** ()

  Destructor for the Built state.
- State * update ()

  Updates the current state.
- State * clone ()

  Creates a deep copy of the Built state.

Public Member Functions inherited from State

- State (int buildTime)

  Constructs a State with the specified build time.
- State (State *state)

  Copy constructor for the State class.
- virtual ~**State** ()

  Destructor for the State.
- int getGameLoopCounter ()

  Gets the current game loop counter.
- int getBuildTime ()

  Gets the build time of the state.
- void **incrementGameLoopCounter** ()

  Increments the game loop counter.

4.13.1 Detailed Description

Represents the built state of an entity.

The Built class inherits from the State class and represents the state of an entity after it has been constructed. It provides methods for updating and initializing the state.

4.13.2 Constructor & Destructor Documentation

4.13.2.1 Built() [1/2]

```
Built::Built (
            int buildTime)
```

Constructs a Built state with the specified build time.

Parameters

buildTime	The time taken to build the entity.

4.13.2.2 Built() [2/2]

```
Built::Built (
            Built * built)
```

Copy constructor for the Built class.

Creates a new Built state by copying the attributes of an existing Built object.

Parameters

built	Pointer to the existing Built object to be copied.

4.13.3 Member Function Documentation

4.13.3.1 clone()

```
State * Built::clone ()  [virtual]
```

Creates a deep copy of the Built state.

This method returns a new Built object that is a copy of the current instance. This allows for proper polymorphic copying of State objects.

Returns

A pointer to a new Built object that is a copy of this instance.

Implements State.

4.13.3.2 update()

```
State * Built::update ()  [virtual]
```

Updates the current state.

Returns

A pointer to the updated state (remains the same in the built state).

Implements State.

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

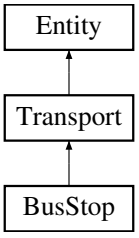
- src/entities/state/Built.h
- src/entities/state/Built.cpp

4.14 BusStop Class Reference

Represents a bus stop entity within the game.

```
#include <BusStop.h>
```

Inheritance diagram for BusStop:



Public Member Functions

- BusStop (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a BusStop entity with specified attributes.
- BusStop (BusStop *busStop)

  Copy constructor for the BusStop class.
- virtual ~**BusStop** ()

  Destructor for the BusStop class.
- void update ()

  Updates the state of the bus stop entity.
- Entity * clone ()

  Creates a clone of the bus stop entity.

Public Member Functions inherited from Transport

- Transport (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a Transport entity with specified attributes.
- Transport (Transport *transport)

  Copy constructor for the Transport class.
- virtual ~**Transport** ()

  Virtual destructor for the Transport class.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

Updates the build state of the entity.

- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).
- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.

- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.14.1 Detailed Description

Represents a bus stop entity within the game.

The BusStop class manages the properties and behavior of bus stop entities, including their position, size, and functionality related to transportation.

4.14.2 Constructor & Destructor Documentation

4.14.2.1 BusStop() [1/2]

```
BusStop::BusStop (
            EntityConfig ec,
            Size size,
            int xPos,
            int yPos)
```

Constructs a BusStop entity with specified attributes.

Parameters

ec	Configuration settings for the entity.
size	Size of the bus stop entity.
xPos	X-coordinate position of the bus stop.
yPos	Y-coordinate position of the bus stop.

4.14.2.2 BusStop() [2/2]

```
BusStop::BusStop (
            BusStop * busStop)
```

Copy constructor for the BusStop class.

Creates a new BusStop entity by copying the attributes of an existing BusStop.

Parameters

busStop	Pointer to the BusStop object to be copied.

4.14.3 Member Function Documentation

4.14.3.1 clone()

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

Creates a clone of the bus stop entity.

Returns

A pointer to the cloned BusStop entity.

Implements Transport.

4.14.3.2 update()

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

Updates the state of the bus stop entity.

Implements Transport.

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

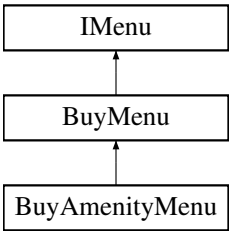
- src/entities/transport/BusStop.h
- src/entities/transport/BusStop.cpp

4.15 BuyAmenityMenu Class Reference

Provides a menu interface for purchasing various types of amenities.

```
#include <BuyAmenityMenu.h>
```

Inheritance diagram for BuyAmenityMenu:



Protected Member Functions

- EntityType chooseEntityType () override

  Displays a list of amenity types for the user to choose from.
- void buildEntity (EntityType type, Size size, int xPos, int yPos) override

  Initiates the construction of the selected amenity.

Protected Member Functions inherited from BuyMenu

- Size chooseBuildingSize (EntityType type)

  Allows the user to select the size of the building to be purchased.
- void chooseBuildingPosition (int &xPos, int &yPos, EntityType type, Size size)

  Allows the user to choose the position of the building on the city grid.
- void confirmPurchase (EntityType type, Size size, int xPos, int yPos)

  Confirms the purchase of the selected building.
- void displayAvailablePositions (const std::vector< std::vector< int > > &positions) const override

  Displays available positions on the city grid with visualization of the entity's size.

Protected Member Functions inherited from IMenu

- std::string repeat (const std::string &str, int times) const

  Utility function to repeat a string multiple times.
- int calculateMaxWidth (const std::string &menuHeading, const std::vector< Section > &sections) const

  Calculates the maximum width required for the menu.
- void printTopBorder (int width) const

  Prints the top border of the menu using box-drawing characters.
- void printBottomBorder (int width) const

  Prints the bottom border of the menu using box-drawing characters.
- void printSectionDivider (int width) const

  Prints a section divider in the menu using box-drawing characters.
- void printDoubleLineDivider (int width) const

  Prints a double-line divider for the main heading of the menu.
- std::string centerText (const std::string &text, int width) const

  Centers text within a specified width using space padding.
- std::string centerTextWithChar (const std::string &text, int width, const std::string &padChar) const

  Centers text within a specified width using a custom character for padding.
- void displayMenu () const

  Displays the formatted menu, including sections and options.
- void **displayChoicePrompt** () const

  Displays the choice prompt for user input.
- void displayChoiceMessagePrompt (const std::string &message) const

  Displays a custom message prompt for user input.
- void **displayInvalidChoice** () const

  Displays an error message when the user makes an invalid choice.
- void displayErrorMessage (const std::string &message) const

  Displays a general error message.
- void displaySuccessMessage (const std::string &message) const

  Displays a success message in green color.
- void **displayPressEnterToContinue** () const

  Displays a message asking the user to press Enter to continue.
- void **clearScreen** () const

  Clears the terminal screen.
- std::string stripColorCodes (const std::string &input) const

  Strips ANSI color codes from a string.

Additional Inherited Members

Public Member Functions inherited from BuyMenu

- BuyMenu ()

  Constructs a new BuyMenu object with default initialization.
- virtual ~BuyMenu ()

  Destructor for the BuyMenu class.
- void display () const override

  Displays the base buy menu.
- void handleInput () override

  Handles user input for the buy menu.

Public Member Functions inherited from IMenu

- **IMenu** ()=default

  Default constructor for IMenu.
- IMenu (std::string heading)

  Constructor to initialize a menu with a specified heading.
- virtual ~**IMenu** ()=default

  Virtual destructor for IMenu.
- void setHeading (const std::string &heading)

  Sets the heading of the menu.

Static Protected Member Functions inherited from IMenu

- static char indexToExtendedChar (int index)

  Converts a numeric index (0-99) to a single character in an extended set.
- static std::string coordinatesToLabel (int x, int y)

  Converts x and y coordinates to a labeled string (e.g., "A, 1").

Protected Attributes inherited from BuyMenu

- EntityType **selectedType**

  The currently selected type of the entity for purchase.
- Size **selectedSize**

  The currently selected size of the entity for purchase.

Protected Attributes inherited from IMenu

- std::vector< Section > **sections**

  List of sections contained in the menu.
- std::string **menuHeading**

  The heading/title of the menu.
- bool **hasExited**

  Flag indicating if the menu has been exited.
- CityManager **cityManager**

  Manager for city-related operations.
- bool **displayResources**

  Flag indicating whether to display resources in the menu.
- bool **isInfoMenu**

  Flag indicating whether to display option numbers.

Static Protected Attributes inherited from IMenu

- static const char * **RESET** = "\033[0m"

  ANSI color codes and styles for use in all menus.
- static const char * **BOLD_WHITE** = "\033[1;37m"
- static const char * **NORMAL_WHITE** = "\033[0;37m"
- static const char * **DARK_GRAY** = "\033[1;30m"
- static const char * **BOLD_YELLOW** = "\033[1;33m"
- static const char * **BOLD_GREEN** = "\033[1;32m"
- static const char * **BOLD_RED** = "\033[1;31m"
- static const char * **BOLD_CYAN** = "\033[1;36m"
- static const char * **BLUE** = "\033[34m"

4.15.1 Detailed Description

Provides a menu interface for purchasing various types of amenities.

The BuyAmenityMenu class allows users to select and purchase different amenities, such as parks, theaters, and monuments. It inherits from the BuyMenu base class and implements methods for choosing the type of amenity and building it at specified locations.

4.15.2 Member Function Documentation

4.15.2.1 buildEntity()

```
void BuyAmenityMenu::buildEntity (
            EntityType type,
            Size size,
            int xPos,
            int yPos)  [override], [protected], [virtual]
```

Initiates the construction of the selected amenity.

Overrides the buildEntity method to handle the construction process for the chosen amenity, utilizing the AmenityManager to build the selected type at the specified coordinates and size.

Parameters

type	The type of amenity to be built.
size	The size of the amenity.
xPos	The x-coordinate for the amenity's position.
yPos	The y-coordinate for the amenity's position.

Implements BuyMenu.

4.15.2.2 chooseEntityType()

```
EntityType BuyAmenityMenu::chooseEntityType ()  [override], [protected], [virtual]
```

Displays a list of amenity types for the user to choose from.

Allows the user to choose an amenity type from the options available. Displays options such as Park, Theater, and Monument.

Overrides the pure virtual method from BuyMenu to provide specific options for selecting amenities. The user can choose among parks, theaters, and monuments.

Returns

The selected EntityType corresponding to the chosen amenity.

The selected EntityType corresponding to the chosen amenity.

Implements BuyMenu.

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

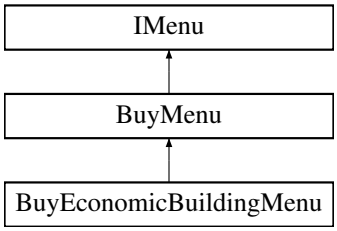
- src/menus/buildings/amenity/BuyAmenityMenu.h
- src/menus/buildings/amenity/BuyAmenityMenu.cpp

4.16 BuyEconomicBuildingMenu Class Reference

Menu for purchasing economic buildings in the game.

```
#include <BuyEconomicBuildingMenu.h>
```

Inheritance diagram for BuyEconomicBuildingMenu:



Public Member Functions

- BuyEconomicBuildingMenu ()

  Constructor for BuyEconomicBuildingMenu. Initializes the menu layout and configurations for economic building purchases.
- ~BuyEconomicBuildingMenu ()

  Destructor for BuyEconomicBuildingMenu. Cleans up any resources or states related to the menu.

Public Member Functions inherited from BuyMenu

- BuyMenu ()

  Constructs a new BuyMenu object with default initialization.
- virtual ~BuyMenu ()

  Destructor for the BuyMenu class.
- void display () const override

  Displays the base buy menu.
- void handleInput () override

  Handles user input for the buy menu.

Public Member Functions inherited from IMenu

- **IMenu** ()=default

  Default constructor for IMenu.
- IMenu (std::string heading)

  Constructor to initialize a menu with a specified heading.
- virtual ~**IMenu** ()=default

  Virtual destructor for IMenu.
- void setHeading (const std::string &heading)

  Sets the heading of the menu.

Protected Member Functions

- EntityType chooseEntityType () override

  Displays a list of economic building types for user selection.
- void buildEntity (EntityType type, Size size, int xPos, int yPos) override

  Initiates the construction of the chosen economic building.

Protected Member Functions inherited from BuyMenu

- Size chooseBuildingSize (EntityType type)

  Allows the user to select the size of the building to be purchased.
- void chooseBuildingPosition (int &xPos, int &yPos, EntityType type, Size size)

  Allows the user to choose the position of the building on the city grid.
- void confirmPurchase (EntityType type, Size size, int xPos, int yPos)

  Confirms the purchase of the selected building.
- void displayAvailablePositions (const std::vector< std::vector< int > > &positions) const override

  Displays available positions on the city grid with visualization of the entity's size.

Protected Member Functions inherited from IMenu

- std::string repeat (const std::string &str, int times) const

  Utility function to repeat a string multiple times.
- int calculateMaxWidth (const std::string &menuHeading, const std::vector< Section > &sections) const

  Calculates the maximum width required for the menu.
- void printTopBorder (int width) const

  Prints the top border of the menu using box-drawing characters.
- void printBottomBorder (int width) const

  Prints the bottom border of the menu using box-drawing characters.
- void printSectionDivider (int width) const

  Prints a section divider in the menu using box-drawing characters.
- void printDoubleLineDivider (int width) const

  Prints a double-line divider for the main heading of the menu.
- std::string centerText (const std::string &text, int width) const

  Centers text within a specified width using space padding.
- std::string centerTextWithChar (const std::string &text, int width, const std::string &padChar) const

  Centers text within a specified width using a custom character for padding.
- void displayMenu () const

  Displays the formatted menu, including sections and options.
- void **displayChoicePrompt** () const

  Displays the choice prompt for user input.
- void displayChoiceMessagePrompt (const std::string &message) const

  Displays a custom message prompt for user input.
- void **displayInvalidChoice** () const

  Displays an error message when the user makes an invalid choice.
- void displayErrorMessage (const std::string &message) const

  Displays a general error message.
- void displaySuccessMessage (const std::string &message) const

  Displays a success message in green color.
- void **displayPressEnterToContinue** () const

  Displays a message asking the user to press Enter to continue.
- void **clearScreen** () const

  Clears the terminal screen.
- std::string stripColorCodes (const std::string &input) const

  Strips ANSI color codes from a string.

Additional Inherited Members

Static Protected Member Functions inherited from IMenu

- static char indexToExtendedChar (int index)

  Converts a numeric index (0-99) to a single character in an extended set.
- static std::string coordinatesToLabel (int x, int y)

  Converts x and y coordinates to a labeled string (e.g., "A, 1").

Protected Attributes inherited from BuyMenu

- EntityType **selectedType**

  The currently selected type of the entity for purchase.
- Size **selectedSize**

  The currently selected size of the entity for purchase.

Protected Attributes inherited from IMenu

- std::vector< Section > **sections**

  List of sections contained in the menu.
- std::string **menuHeading**

  The heading/title of the menu.
- bool **hasExited**

  Flag indicating if the menu has been exited.
- CityManager **cityManager**

  Manager for city-related operations.
- bool **displayResources**

  Flag indicating whether to display resources in the menu.
- bool **isInfoMenu**

  Flag indicating whether to display option numbers.

Static Protected Attributes inherited from IMenu

- static const char * **RESET** = "\033[0m"

  ANSI color codes and styles for use in all menus.
- static const char * **BOLD_WHITE** = "\033[1;37m"
- static const char * **NORMAL_WHITE** = "\033[0;37m"
- static const char * **DARK_GRAY** = "\033[1;30m"
- static const char * **BOLD_YELLOW** = "\033[1;33m"
- static const char * **BOLD_GREEN** = "\033[1;32m"
- static const char * **BOLD_RED** = "\033[1;31m"
- static const char * **BOLD_CYAN** = "\033[1;36m"
- static const char * **BLUE** = "\033[34m"

4.16.1 Detailed Description

Menu for purchasing economic buildings in the game.

This class provides a user interface for selecting and buying different types of economic buildings, such as Offices, Shopping Malls, and Factories. It extends the BuyMenu class to include specific logic for handling economic buildings.

4.16.2 Constructor & Destructor Documentation

4.16.2.1 BuyEconomicBuildingMenu()

```
BuyEconomicBuildingMenu::BuyEconomicBuildingMenu ()
```

Constructor for BuyEconomicBuildingMenu. Initializes the menu layout and configurations for economic building purchases.

Constructor for BuyEconomicBuildingMenu. Initializes the base BuyMenu and sets up any economic building-related configurations.

4.16.2.2 ~BuyEconomicBuildingMenu()

```
BuyEconomicBuildingMenu::~BuyEconomicBuildingMenu ()
```

Destructor for BuyEconomicBuildingMenu. Cleans up any resources or states related to the menu.

Destructor for BuyEconomicBuildingMenu.

4.16.3 Member Function Documentation

4.16.3.1 buildEntity()

```
void BuyEconomicBuildingMenu::buildEntity (
            EntityType type,
            Size size,
            int xPos,
            int yPos)  [override], [protected], [virtual]
```

Initiates the construction of the chosen economic building.

This method handles the building process by interacting with the BuildingManager to create an economic building at the specified location and size.

Parameters

type	The type of economic building to construct (e.g., Office, Shopping Mall).
size	The size category of the building (e.g., Small, Medium, Large).
xPos	The x-coordinate for the building's placement.
yPos	The y-coordinate for the building's placement.

Implements BuyMenu.

4.16.3.2 chooseEntityType()

```
EntityType BuyEconomicBuildingMenu::chooseEntityType ()  [override], [protected], [virtual]
```

Displays a list of economic building types for user selection.

Allows the user to choose an economic building type from the options available. Displays options such as Office, Shopping Mall, and Factory.

Overrides the pure virtual method from BuyMenu to present economic building options to the user. This method handles displaying building types and capturing user input to select an option.

Returns

The selected EntityType representing the chosen economic building.

The selected EntityType corresponding to the chosen economic building.

Implements BuyMenu.

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

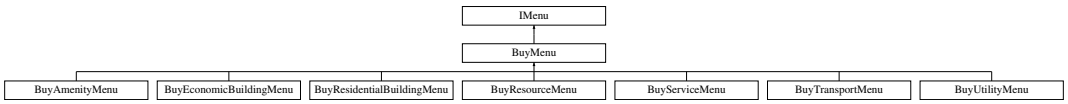
- src/menus/buildings/economic/BuyEconomicBuildingMenu.h
- src/menus/buildings/economic/BuyEconomicBuildingMenu.cpp

4.17 BuyMenu Class Reference

Abstract class representing the Buy Menu in the game.

`#include <BuyMenu.h>`

Inheritance diagram for BuyMenu:



Public Member Functions

- BuyMenu ()

  Constructs a new BuyMenu object with default initialization.
- virtual ~BuyMenu ()

  Destructor for the BuyMenu class.
- void display () const override

  Displays the base buy menu.
- void handleInput () override

  Handles user input for the buy menu.

Public Member Functions inherited from IMenu

- **IMenu** ()=default

  Default constructor for IMenu.
- IMenu (std::string heading)

  Constructor to initialize a menu with a specified heading.
- virtual ~**IMenu** ()=default

  Virtual destructor for IMenu.
- void setHeading (const std::string &heading)

  Sets the heading of the menu.

Protected Member Functions

- virtual EntityType chooseEntityType ()=0

  Selects the type of entity to be purchased.
- Size chooseBuildingSize (EntityType type)

  Allows the user to select the size of the building to be purchased.
- void chooseBuildingPosition (int &xPos, int &yPos, EntityType type, Size size)

  Allows the user to choose the position of the building on the city grid.
- void confirmPurchase (EntityType type, Size size, int xPos, int yPos)

  Confirms the purchase of the selected building.
- virtual void buildEntity (EntityType type, Size size, int xPos, int yPos)=0

  Pure virtual method to handle the building of the entity.
- void displayAvailablePositions (const std::vector< std::vector< int > > &positions) const override

  Displays available positions on the city grid with visualization of the entity's size.

Protected Member Functions inherited from IMenu

- std::string repeat (const std::string &str, int times) const

  Utility function to repeat a string multiple times.
- int calculateMaxWidth (const std::string &menuHeading, const std::vector< Section > &sections) const

  Calculates the maximum width required for the menu.
- void printTopBorder (int width) const

  Prints the top border of the menu using box-drawing characters.
- void printBottomBorder (int width) const

  Prints the bottom border of the menu using box-drawing characters.
- void printSectionDivider (int width) const

  Prints a section divider in the menu using box-drawing characters.
- void printDoubleLineDivider (int width) const

  Prints a double-line divider for the main heading of the menu.
- std::string centerText (const std::string &text, int width) const

  Centers text within a specified width using space padding.
- std::string centerTextWithChar (const std::string &text, int width, const std::string &padChar) const

  Centers text within a specified width using a custom character for padding.
- void displayMenu () const

  Displays the formatted menu, including sections and options.
- void **displayChoicePrompt** () const

  Displays the choice prompt for user input.
- void displayChoiceMessagePrompt (const std::string &message) const

  Displays a custom message prompt for user input.
- void **displayInvalidChoice** () const

  Displays an error message when the user makes an invalid choice.
- void displayErrorMessage (const std::string &message) const

  Displays a general error message.
- void displaySuccessMessage (const std::string &message) const

  Displays a success message in green color.
- void **displayPressEnterToContinue** () const

  Displays a message asking the user to press Enter to continue.
- void **clearScreen** () const

  Clears the terminal screen.
- std::string stripColorCodes (const std::string &input) const

  Strips ANSI color codes from a string.

Protected Attributes

- EntityType **selectedType**

  The currently selected type of the entity for purchase.
- Size **selectedSize**

  The currently selected size of the entity for purchase.

Protected Attributes inherited from IMenu

- std::vector< Section > **sections**

  List of sections contained in the menu.
- std::string **menuHeading**

  The heading/title of the menu.
- bool **hasExited**

  Flag indicating if the menu has been exited.
- CityManager **cityManager**

  Manager for city-related operations.
- bool **displayResources**

  Flag indicating whether to display resources in the menu.
- bool **isInfoMenu**

  Flag indicating whether to display option numbers.

Additional Inherited Members

Static Protected Member Functions inherited from IMenu

- static char indexToExtendedChar (int index)

  Converts a numeric index (0-99) to a single character in an extended set.
- static std::string coordinatesToLabel (int x, int y)

  Converts x and y coordinates to a labeled string (e.g., "A, 1").

Static Protected Attributes inherited from IMenu

- static const char * **RESET** = "\033[0m"

  ANSI color codes and styles for use in all menus.
- static const char * **BOLD_WHITE** = "\033[1;37m"
- static const char * **NORMAL_WHITE** = "\033[0;37m"
- static const char * **DARK_GRAY** = "\033[1;30m"
- static const char * **BOLD_YELLOW** = "\033[1;33m"
- static const char * **BOLD_GREEN** = "\033[1;32m"
- static const char * **BOLD_RED** = "\033[1;31m"
- static const char * **BOLD_CYAN** = "\033[1;36m"
- static const char * **BLUE** = "\033[34m"

4.17.1 Detailed Description

Abstract class representing the Buy Menu in the game.

This class provides the interface and common functionality for menus that handle the purchase of buildings of different types and sizes. Derived classes must implement type-specific logic.

4.17.2 Constructor & Destructor Documentation

4.17.2.1 BuyMenu()

```
BuyMenu::BuyMenu ()
```

Constructs a new BuyMenu object with default initialization.

Constructs a new BuyMenu with default available resources.

This constructor initializes the BuyMenu, setting up the necessary resources for display.

4.17.2.2 ~BuyMenu()

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

Destructor for the BuyMenu class.

Destructor for BuyMenu.

Ensures proper cleanup of resources used by the menu.

4.17.3 Member Function Documentation

4.17.3.1 buildEntity()

```
virtual void BuyMenu::buildEntity (
            EntityType type,
            Size size,
            int xPos,
            int yPos)  [protected], [pure virtual]
```

Pure virtual method to handle the building of the entity.

This method must be implemented by derived classes to specify the logic for constructing the selected entity on the city grid.

Parameters

type	The type of entity to build.
size	The size of the entity.
xPos	The x-coordinate of the building's position.
yPos	The y-coordinate of the building's position.

Implemented in BuyAmenityMenu, BuyEconomicBuildingMenu, BuyResidentialBuildingMenu, BuyResourceMenu, BuyServiceMenu, BuyTransportMenu, and BuyUtilityMenu.

4.17.3.2 chooseBuildingPosition()

```
void BuyMenu::chooseBuildingPosition (
            int & xPos,
            int & yPos,
            EntityType type,
            Size size)  [protected]
```

Allows the user to choose the position of the building on the city grid.

Allows the user to choose the position of the building on the grid. Displays available positions and lets the user select one.

Displays a grid with available positions based on the type and size of the entity, and prompts the user to select a valid position.

Parameters

xPos	Reference to the x-coordinate for the chosen position.
yPos	Reference to the y-coordinate for the chosen position.
type	The type of the building being placed.
size	The size of the building being placed.
xPos	Reference to the x-coordinate for the building's position.
yPos	Reference to the y-coordinate for the building's position.

4.17.3.3 chooseBuildingSize()

```
Size BuyMenu::chooseBuildingSize (
            EntityType type)  [protected]
```

Allows the user to select the size of the building to be purchased.

Allows the user to choose the size of the building based on available resources. Displays an error message if the user cannot afford the selected size.

Ensures that the selected size is affordable based on the city's available resources. Displays an error message if the selected size cannot be afforded.

Parameters

type	The type of the entity being purchased.

Returns

Size The chosen size of the building.

Parameters

type	The EntityType of the building.

Returns

Size The selected size of the building.

4.17.3.4 chooseEntityType()

```
virtual EntityType BuyMenu::chooseEntityType ()  [protected], [pure virtual]
```

Selects the type of entity to be purchased.

This pure virtual function must be implemented by derived classes to allow dynamic selection of entity types.

Returns

EntityType The chosen type of entity.

Implemented in BuyAmenityMenu, BuyEconomicBuildingMenu, BuyResidentialBuildingMenu, BuyResourceMenu, BuyServiceMenu, BuyTransportMenu, and BuyUtilityMenu.

4.17.3.5 confirmPurchase()

```
void BuyMenu::confirmPurchase (
            EntityType type,
            Size size,
            int xPos,
            int yPos)  [protected]
```

Confirms the purchase of the selected building.

Confirms the building purchase and displays a summary including type, size, and cost. Asks for final confirmation before proceeding with the purchase.

Displays a summary of the purchase details, including type, size, and cost. Asks for final confirmation from the user before proceeding with the purchase.

Parameters

type	The type of the building being purchased.
size	The size of the building being purchased.
xPos	The x-coordinate of the building's position.
yPos	The y-coordinate of the building's position.
type	The type of the building.
size	The size of the building.
xPos	The x-coordinate for the building's position.
yPos	The y-coordinate for the building's position.

4.17.3.6 display()

```
void BuyMenu::display () const  [override], [virtual]
```

Displays the base buy menu.

Displays the base menu (empty in this base class). Derived classes should implement their own display logic.

This method should be overridden by derived classes to implement specific display logic.

Implements IMenu.

4.17.3.7 displayAvailablePositions()

```
void BuyMenu::displayAvailablePositions (
            const std::vector< std::vector< int > > & positions) const  [override], [protected],
[virtual]
```

Displays available positions on the city grid with visualization of the entity's size.

Highlights potential positions for placement and ensures proper boundary handling for larger entities.

Parameters

positions	A vector of available grid positions.

Reimplemented from IMenu.

4.17.3.8 handleInput()

```
void BuyMenu::handleInput () [override], [virtual]
```

Handles user input for the buy menu.

Handles the input workflow for purchasing a building. It allows the user to select the building type, size, and position, and then confirms the purchase.

This method manages the workflow for purchasing a building, including type, size, and position selection, and final confirmation of the purchase.

Implements IMenu.

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

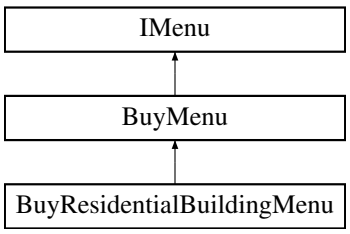
- src/menus/base/BuyMenu.h
- src/menus/base/BuyMenu.cpp

4.18 BuyResidentialBuildingMenu Class Reference

Menu for purchasing residential buildings in the game.

```
#include <BuyResidentialBuildingMenu.h>
```

Inheritance diagram for BuyResidentialBuildingMenu:



Public Member Functions

- BuyResidentialBuildingMenu ()

  Constructor for BuyResidentialBuildingMenu. Initializes the menu layout and configurations for residential building purchases.
- ~BuyResidentialBuildingMenu ()

  Destructor for BuyResidentialBuildingMenu. Cleans up any resources or states related to the menu.

Public Member Functions inherited from BuyMenu

- BuyMenu ()

  Constructs a new BuyMenu object with default initialization.
- virtual ~BuyMenu ()

  Destructor for the BuyMenu class.
- void display () const override

  Displays the base buy menu.
- void handleInput () override

  Handles user input for the buy menu.

Public Member Functions inherited from IMenu

- **IMenu** ()=default

  Default constructor for IMenu.
- IMenu (std::string heading)

  Constructor to initialize a menu with a specified heading.
- virtual ~**IMenu** ()=default

  Virtual destructor for IMenu.
- void setHeading (const std::string &heading)

  Sets the heading of the menu.

Protected Member Functions

- EntityType chooseEntityType () override

  Displays a list of residential building types for user selection.
- void buildEntity (EntityType type, Size size, int xPos, int yPos) override

  Initiates the construction of the chosen residential building.

Protected Member Functions inherited from BuyMenu

- Size chooseBuildingSize (EntityType type)

  Allows the user to select the size of the building to be purchased.
- void chooseBuildingPosition (int &xPos, int &yPos, EntityType type, Size size)

  Allows the user to choose the position of the building on the city grid.
- void confirmPurchase (EntityType type, Size size, int xPos, int yPos)

  Confirms the purchase of the selected building.
- void displayAvailablePositions (const std::vector< std::vector< int > > &positions) const override

  Displays available positions on the city grid with visualization of the entity's size.

Protected Member Functions inherited from IMenu

- std::string repeat (const std::string &str, int times) const

  Utility function to repeat a string multiple times.
- int calculateMaxWidth (const std::string &menuHeading, const std::vector< Section > &sections) const

  Calculates the maximum width required for the menu.
- void printTopBorder (int width) const

  Prints the top border of the menu using box-drawing characters.
- void printBottomBorder (int width) const

  Prints the bottom border of the menu using box-drawing characters.
- void printSectionDivider (int width) const

  Prints a section divider in the menu using box-drawing characters.
- void printDoubleLineDivider (int width) const

  Prints a double-line divider for the main heading of the menu.
- std::string centerText (const std::string &text, int width) const

  Centers text within a specified width using space padding.
- std::string centerTextWithChar (const std::string &text, int width, const std::string &padChar) const

  Centers text within a specified width using a custom character for padding.
- void displayMenu () const

  Displays the formatted menu, including sections and options.
- void **displayChoicePrompt** () const

  Displays the choice prompt for user input.
- void displayChoiceMessagePrompt (const std::string &message) const

  Displays a custom message prompt for user input.
- void **displayInvalidChoice** () const

  Displays an error message when the user makes an invalid choice.
- void displayErrorMessage (const std::string &message) const

  Displays a general error message.
- void displaySuccessMessage (const std::string &message) const

  Displays a success message in green color.
- void **displayPressEnterToContinue** () const

  Displays a message asking the user to press Enter to continue.
- void **clearScreen** () const

  Clears the terminal screen.
- std::string stripColorCodes (const std::string &input) const

  Strips ANSI color codes from a string.

Additional Inherited Members

Static Protected Member Functions inherited from IMenu

- static char indexToExtendedChar (int index)

  Converts a numeric index (0-99) to a single character in an extended set.
- static std::string coordinatesToLabel (int x, int y)

  Converts x and y coordinates to a labeled string (e.g., "A, 1").

Protected Attributes inherited from BuyMenu

- EntityType **selectedType**

  The currently selected type of the entity for purchase.
- Size **selectedSize**

  The currently selected size of the entity for purchase.

Protected Attributes inherited from IMenu

- std::vector< Section > **sections**

  List of sections contained in the menu.
- std::string **menuHeading**

  The heading/title of the menu.
- bool **hasExited**

  Flag indicating if the menu has been exited.
- CityManager **cityManager**

  Manager for city-related operations.
- bool **displayResources**

  Flag indicating whether to display resources in the menu.
- bool **isInfoMenu**

  Flag indicating whether to display option numbers.

Static Protected Attributes inherited from IMenu

- static const char * **RESET** = "\033[0m"

  ANSI color codes and styles for use in all menus.
- static const char * **BOLD_WHITE** = "\033[1;37m"
- static const char * **NORMAL_WHITE** = "\033[0;37m"
- static const char * **DARK_GRAY** = "\033[1;30m"
- static const char * **BOLD_YELLOW** = "\033[1;33m"
- static const char * **BOLD_GREEN** = "\033[1;32m"
- static const char * **BOLD_RED** = "\033[1;31m"
- static const char * **BOLD_CYAN** = "\033[1;36m"
- static const char * **BLUE** = "\033[34m"

4.18.1 Detailed Description

Menu for purchasing residential buildings in the game.

This class provides a user interface for selecting and buying different types of residential buildings, such as Houses and Apartments. It extends the BuyMenu class to include specific logic for handling residential buildings.

4.18.2 Constructor & Destructor Documentation

4.18.2.1 BuyResidentialBuildingMenu()

```
BuyResidentialBuildingMenu::BuyResidentialBuildingMenu ()
```

Constructor for BuyResidentialBuildingMenu. Initializes the menu layout and configurations for residential building purchases.

Constructor for BuyResidentialBuildingMenu. Initializes the base BuyMenu for residential building selection.

4.18.2.2 ~BuyResidentialBuildingMenu()

```
BuyResidentialBuildingMenu::~BuyResidentialBuildingMenu ()
```

Destructor for BuyResidentialBuildingMenu. Cleans up any resources or states related to the menu.

Destructor for BuyResidentialBuildingMenu.

4.18.3 Member Function Documentation

4.18.3.1 buildEntity()

```
void BuyResidentialBuildingMenu::buildEntity (
            EntityType type,
            Size size,
            int xPos,
            int yPos)  [override], [protected], [virtual]
```

Initiates the construction of the chosen residential building.

This method handles the building process by interacting with the BuildingManager to create a residential building at the specified location and size.

Parameters

type	The type of residential building to construct (e.g., House, Apartment).
size	The size category of the building (e.g., Small, Medium, Large).
xPos	The x-coordinate for the building's placement.
yPos	The y-coordinate for the building's placement.

Implements BuyMenu.

4.18.3.2 chooseEntityType()

```
EntityType BuyResidentialBuildingMenu::chooseEntityType ()  [override], [protected], [virtual]
```

Displays a list of residential building types for user selection.

Displays the options for selecting a residential building type. The user can choose between House or Apartment.

Overrides the pure virtual method from BuyMenu to present residential building options to the user. This method handles displaying building types and capturing user input to select an option.

Returns

The selected EntityType representing the chosen residential building.

The selected EntityType for the residential building.

Implements BuyMenu.

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

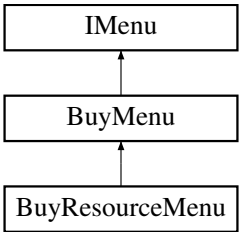
- src/menus/buildings/residential/BuyResidentialBuildingMenu.h
- src/menus/buildings/residential/BuyResidentialBuildingMenu.cpp

4.19 BuyResourceMenu Class Reference

Menu for purchasing resource-producing buildings in the game.

`#include <BuyResourceMenu.h>`

Inheritance diagram for BuyResourceMenu:



Public Member Functions

- BuyResourceMenu ()

  Constructor for BuyResourceMenu. Initializes the menu layout and configurations for resource-producing building purchases.
- ~BuyResourceMenu ()

  Destructor for BuyResourceMenu. Cleans up any resources or states related to the menu.

Public Member Functions inherited from BuyMenu

- BuyMenu ()

  Constructs a new BuyMenu object with default initialization.
- virtual ~BuyMenu ()

  Destructor for the BuyMenu class.
- void display () const override

  Displays the base buy menu.
- void handleInput () override

  Handles user input for the buy menu.

Public Member Functions inherited from IMenu

- **IMenu** ()=default

  Default constructor for IMenu.
- IMenu (std::string heading)

  Constructor to initialize a menu with a specified heading.
- virtual ~**IMenu** ()=default

  Virtual destructor for IMenu.
- void setHeading (const std::string &heading)

  Sets the heading of the menu.

Protected Member Functions

- EntityType chooseEntityType () override

  Displays a list of resource-producing building types for user selection.
- void buildEntity (EntityType type, Size size, int xPos, int yPos) override

  Initiates the construction of the selected resource-producing building.

Protected Member Functions inherited from BuyMenu

- Size chooseBuildingSize (EntityType type)

  Allows the user to select the size of the building to be purchased.
- void chooseBuildingPosition (int &xPos, int &yPos, EntityType type, Size size)

  Allows the user to choose the position of the building on the city grid.
- void confirmPurchase (EntityType type, Size size, int xPos, int yPos)

  Confirms the purchase of the selected building.
- void displayAvailablePositions (const std::vector< std::vector< int > > &positions) const override

  Displays available positions on the city grid with visualization of the entity's size.

Protected Member Functions inherited from IMenu

- std::string repeat (const std::string &str, int times) const

  Utility function to repeat a string multiple times.
- int calculateMaxWidth (const std::string &menuHeading, const std::vector< Section > &sections) const

  Calculates the maximum width required for the menu.
- void printTopBorder (int width) const

  Prints the top border of the menu using box-drawing characters.
- void printBottomBorder (int width) const

  Prints the bottom border of the menu using box-drawing characters.
- void printSectionDivider (int width) const

  Prints a section divider in the menu using box-drawing characters.
- void printDoubleLineDivider (int width) const

  Prints a double-line divider for the main heading of the menu.
- std::string centerText (const std::string &text, int width) const

  Centers text within a specified width using space padding.
- std::string centerTextWithChar (const std::string &text, int width, const std::string &padChar) const

  Centers text within a specified width using a custom character for padding.
- void displayMenu () const

  Displays the formatted menu, including sections and options.
- void **displayChoicePrompt** () const

  Displays the choice prompt for user input.
- void displayChoiceMessagePrompt (const std::string &message) const

  Displays a custom message prompt for user input.
- void **displayInvalidChoice** () const

  Displays an error message when the user makes an invalid choice.
- void displayErrorMessage (const std::string &message) const

  Displays a general error message.
- void displaySuccessMessage (const std::string &message) const

  Displays a success message in green color.
- void **displayPressEnterToContinue** () const

  Displays a message asking the user to press Enter to continue.
- void **clearScreen** () const

  Clears the terminal screen.
- std::string stripColorCodes (const std::string &input) const

  Strips ANSI color codes from a string.

Additional Inherited Members

Static Protected Member Functions inherited from IMenu

- static char indexToExtendedChar (int index)

  Converts a numeric index (0-99) to a single character in an extended set.
- static std::string coordinatesToLabel (int x, int y)

  Converts x and y coordinates to a labeled string (e.g., "A, 1").

Protected Attributes inherited from BuyMenu

- EntityType **selectedType**

  The currently selected type of the entity for purchase.
- Size **selectedSize**

  The currently selected size of the entity for purchase.

Protected Attributes inherited from IMenu

- std::vector< Section > **sections**

  List of sections contained in the menu.
- std::string **menuHeading**

  The heading/title of the menu.
- bool **hasExited**

  Flag indicating if the menu has been exited.
- CityManager **cityManager**

  Manager for city-related operations.
- bool **displayResources**

  Flag indicating whether to display resources in the menu.
- bool **isInfoMenu**

  Flag indicating whether to display option numbers.

Static Protected Attributes inherited from IMenu

- static const char * **RESET** = "\033[0m"

  ANSI color codes and styles for use in all menus.
- static const char * **BOLD_WHITE** = "\033[1;37m"
- static const char * **NORMAL_WHITE** = "\033[0;37m"
- static const char * **DARK_GRAY** = "\033[1;30m"
- static const char * **BOLD_YELLOW** = "\033[1;33m"
- static const char * **BOLD_GREEN** = "\033[1;32m"
- static const char * **BOLD_RED** = "\033[1;31m"
- static const char * **BOLD_CYAN** = "\033[1;36m"
- static const char * **BLUE** = "\033[34m"

4.19.1 Detailed Description

Menu for purchasing resource-producing buildings in the game.

This class provides a user interface for selecting and purchasing different types of resource-producing buildings, such as Wood Production facilities, Stone Quarries, and Concrete Factories. It extends the BuyMenu class to include specific logic for handling resource-based structures.

4.19.2 Constructor & Destructor Documentation

4.19.2.1 BuyResourceMenu()

```
BuyResourceMenu::BuyResourceMenu ()
```

Constructor for BuyResourceMenu. Initializes the menu layout and configurations for resource-producing building purchases.

Constructor for BuyResourceMenu. Initializes the base BuyMenu for resource-producing building selection.

4.19.2.2 ~BuyResourceMenu()

```
BuyResourceMenu::~BuyResourceMenu ()
```

Destructor for BuyResourceMenu. Cleans up any resources or states related to the menu.

Destructor for BuyResourceMenu.

4.19.3 Member Function Documentation

4.19.3.1 buildEntity()

```
void BuyResourceMenu::buildEntity (
            EntityType type,
            Size size,
            int xPos,
            int yPos)  [override], [protected], [virtual]
```

Initiates the construction of the selected resource-producing building.

This method interacts with the ResourceManager to create a building at the specified location and size.

Parameters

type	The type of resource-producing building to construct (e.g., Wood Production, Stone Quarry).
size	The size category of the building (e.g., Small, Medium, Large).
xPos	The x-coordinate for the building's placement.
yPos	The y-coordinate for the building's placement.

Implements BuyMenu.

4.19.3.2 chooseEntityType()

```
EntityType BuyResourceMenu::chooseEntityType ()  [override], [protected], [virtual]
```

Displays a list of resource-producing building types for user selection.

Displays the options for selecting a resource-producing building type. The user can choose between Wood Production, Stone Quarry, or Concrete Factory.

Overrides the pure virtual method from BuyMenu to present specific resource-building options to the user. This method displays building types and handles user input to select an option.

Returns

The selected EntityType representing the chosen resource-producing building.

The selected EntityType for the resource-producing building.

Implements BuyMenu.

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

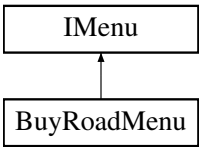
- src/menus/buildings/resource/BuyResourceMenu.h
- src/menus/buildings/resource/BuyResourceMenu.cpp

4.20 BuyRoadMenu Class Reference

Provides functionality for players to purchase roads and place them in the city.

```
#include <BuyRoadMenu.h>
```

Inheritance diagram for BuyRoadMenu:



Public Member Functions

- BuyRoadMenu ()

  Constructs the BuyRoadMenu.
- virtual ~BuyRoadMenu ()

  Destructor for BuyRoadMenu.
- void display () const override

  Displays the road buying menu.
- void handleInput () override

  Handles user input for selecting a position and confirming road purchase.

Public Member Functions inherited from IMenu

- **IMenu** ()=default

  Default constructor for IMenu.
- IMenu (std::string heading)

  Constructor to initialize a menu with a specified heading.
- virtual ~**IMenu** ()=default

  Virtual destructor for IMenu.
- void setHeading (const std::string &heading)

  Sets the heading of the menu.

Additional Inherited Members

Protected Member Functions inherited from IMenu

- std::string repeat (const std::string &str, int times) const

  Utility function to repeat a string multiple times.
- int calculateMaxWidth (const std::string &menuHeading, const std::vector< Section > &sections) const

  Calculates the maximum width required for the menu.
- void printTopBorder (int width) const

  Prints the top border of the menu using box-drawing characters.
- void printBottomBorder (int width) const

  Prints the bottom border of the menu using box-drawing characters.
- void printSectionDivider (int width) const

  Prints a section divider in the menu using box-drawing characters.
- void printDoubleLineDivider (int width) const

  Prints a double-line divider for the main heading of the menu.
- std::string centerText (const std::string &text, int width) const

  Centers text within a specified width using space padding.
- std::string centerTextWithChar (const std::string &text, int width, const std::string &padChar) const

  Centers text within a specified width using a custom character for padding.
- void displayMenu () const

  Displays the formatted menu, including sections and options.
- void **displayChoicePrompt** () const

  Displays the choice prompt for user input.
- void displayChoiceMessagePrompt (const std::string &message) const

  Displays a custom message prompt for user input.
- void **displayInvalidChoice** () const

  Displays an error message when the user makes an invalid choice.
- void displayErrorMessage (const std::string &message) const

  Displays a general error message.
- void displaySuccessMessage (const std::string &message) const

  Displays a success message in green color.
- void **displayPressEnterToContinue** () const

  Displays a message asking the user to press Enter to continue.
- void **clearScreen** () const

  Clears the terminal screen.
- std::string stripColorCodes (const std::string &input) const

  Strips ANSI color codes from a string.
- virtual void displayAvailablePositions (const std::vector< std::vector< int > > &positions) const

  Displays available positions on the city grid for an entity.

Static Protected Member Functions inherited from IMenu

- static char indexToExtendedChar (int index)

  Converts a numeric index (0-99) to a single character in an extended set.
- static std::string coordinatesToLabel (int x, int y)

  Converts x and y coordinates to a labeled string (e.g., "A, 1").

Protected Attributes inherited from IMenu

- std::vector< Section > **sections**

  List of sections contained in the menu.
- std::string **menuHeading**

  The heading/title of the menu.
- bool **hasExited**

  Flag indicating if the menu has been exited.
- CityManager **cityManager**

  Manager for city-related operations.
- bool **displayResources**

  Flag indicating whether to display resources in the menu.
- bool **isInfoMenu**

  Flag indicating whether to display option numbers.

Static Protected Attributes inherited from IMenu

- static const char * **RESET** = "\033[0m"

  ANSI color codes and styles for use in all menus.
- static const char * **BOLD_WHITE** = "\033[1;37m"
- static const char * **NORMAL_WHITE** = "\033[0;37m"
- static const char * **DARK_GRAY** = "\033[1;30m"
- static const char * **BOLD_YELLOW** = "\033[1;33m"
- static const char * **BOLD_GREEN** = "\033[1;32m"
- static const char * **BOLD_RED** = "\033[1;31m"
- static const char * **BOLD_CYAN** = "\033[1;36m"
- static const char * **BLUE** = "\033[34m"

4.20.1 Detailed Description

Provides functionality for players to purchase roads and place them in the city.

The BuyRoadMenu class allows users to select positions in the city to place new roads, confirm their purchase, and handle related input and display operations.

4.20.2 Constructor & Destructor Documentation

4.20.2.1 BuyRoadMenu()

```
BuyRoadMenu::BuyRoadMenu ()
```

Constructs the BuyRoadMenu.

Initializes the menu with relevant sections and headings.

4.20.2.2 ~BuyRoadMenu()

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

Destructor for BuyRoadMenu.

Cleans up resources used by the BuyRoadMenu.

4.20.3 Member Function Documentation

4.20.3.1 display()

```
void BuyRoadMenu::display () const  [override], [virtual]
```

Displays the road buying menu.

Overrides the display method of IMenu to render the road purchase interface.

Implements IMenu.

4.20.3.2 handleInput()

```
void BuyRoadMenu::handleInput ()  [override], [virtual]
```

Handles user input for selecting a position and confirming road purchase.

Processes input to guide users through selecting positions for roads and finalizing their purchase.

Implements IMenu.

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

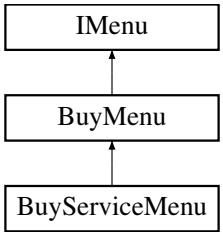
- src/menus/road/BuyRoadMenu.h
- src/menus/road/BuyRoadMenu.cpp

4.21 BuyServiceMenu Class Reference

Menu for purchasing service buildings in the game.

```
#include <BuyServiceMenu.h>
```

Inheritance diagram for BuyServiceMenu:



Public Member Functions

- BuyServiceMenu ()

  Constructor for BuyServiceMenu. Initializes the menu layout and configurations for service building purchases.
- ~BuyServiceMenu ()

  Destructor for BuyServiceMenu. Cleans up any resources or states related to the menu.

Public Member Functions inherited from BuyMenu

- BuyMenu ()

  Constructs a new BuyMenu object with default initialization.
- virtual ~BuyMenu ()

  Destructor for the BuyMenu class.
- void display () const override

  Displays the base buy menu.
- void handleInput () override

  Handles user input for the buy menu.

Public Member Functions inherited from IMenu

- **IMenu** ()=default

  Default constructor for IMenu.
- IMenu (std::string heading)

  Constructor to initialize a menu with a specified heading.
- virtual ~**IMenu** ()=default

  Virtual destructor for IMenu.
- void setHeading (const std::string &heading)

  Sets the heading of the menu.

Protected Member Functions

- EntityType chooseEntityType () override

  Displays a list of service building types for user selection.
- void buildEntity (EntityType type, Size size, int xPos, int yPos) override

  Initiates the construction of the selected service building.

Protected Member Functions inherited from BuyMenu

- Size chooseBuildingSize (EntityType type)

  Allows the user to select the size of the building to be purchased.
- void chooseBuildingPosition (int &xPos, int &yPos, EntityType type, Size size)

  Allows the user to choose the position of the building on the city grid.
- void confirmPurchase (EntityType type, Size size, int xPos, int yPos)

  Confirms the purchase of the selected building.
- void displayAvailablePositions (const std::vector< std::vector< int > > &positions) const override

  Displays available positions on the city grid with visualization of the entity's size.

Protected Member Functions inherited from IMenu

- std::string repeat (const std::string &str, int times) const

  Utility function to repeat a string multiple times.
- int calculateMaxWidth (const std::string &menuHeading, const std::vector< Section > &sections) const

  Calculates the maximum width required for the menu.
- void printTopBorder (int width) const

  Prints the top border of the menu using box-drawing characters.
- void printBottomBorder (int width) const

  Prints the bottom border of the menu using box-drawing characters.
- void printSectionDivider (int width) const

  Prints a section divider in the menu using box-drawing characters.
- void printDoubleLineDivider (int width) const

  Prints a double-line divider for the main heading of the menu.
- std::string centerText (const std::string &text, int width) const

  Centers text within a specified width using space padding.
- std::string centerTextWithChar (const std::string &text, int width, const std::string &padChar) const

  Centers text within a specified width using a custom character for padding.
- void displayMenu () const

  Displays the formatted menu, including sections and options.
- void **displayChoicePrompt** () const

  Displays the choice prompt for user input.
- void displayChoiceMessagePrompt (const std::string &message) const

  Displays a custom message prompt for user input.
- void **displayInvalidChoice** () const

  Displays an error message when the user makes an invalid choice.
- void displayErrorMessage (const std::string &message) const

  Displays a general error message.
- void displaySuccessMessage (const std::string &message) const

  Displays a success message in green color.
- void **displayPressEnterToContinue** () const

  Displays a message asking the user to press Enter to continue.
- void **clearScreen** () const

  Clears the terminal screen.
- std::string stripColorCodes (const std::string &input) const

  Strips ANSI color codes from a string.

Additional Inherited Members

Static Protected Member Functions inherited from IMenu

- static char indexToExtendedChar (int index)

  Converts a numeric index (0-99) to a single character in an extended set.
- static std::string coordinatesToLabel (int x, int y)

  Converts x and y coordinates to a labeled string (e.g., "A, 1").

Protected Attributes inherited from BuyMenu

- EntityType **selectedType**

  The currently selected type of the entity for purchase.
- Size **selectedSize**

  The currently selected size of the entity for purchase.

Protected Attributes inherited from IMenu

- std::vector< Section > **sections**

  List of sections contained in the menu.
- std::string **menuHeading**

  The heading/title of the menu.
- bool **hasExited**

  Flag indicating if the menu has been exited.
- CityManager **cityManager**

  Manager for city-related operations.
- bool **displayResources**

  Flag indicating whether to display resources in the menu.
- bool **isInfoMenu**

  Flag indicating whether to display option numbers.

Static Protected Attributes inherited from IMenu

- static const char * **RESET** = "\033[0m"

  ANSI color codes and styles for use in all menus.
- static const char * **BOLD_WHITE** = "\033[1;37m"
- static const char * **NORMAL_WHITE** = "\033[0;37m"
- static const char * **DARK_GRAY** = "\033[1;30m"
- static const char * **BOLD_YELLOW** = "\033[1;33m"
- static const char * **BOLD_GREEN** = "\033[1;32m"
- static const char * **BOLD_RED** = "\033[1;31m"
- static const char * **BOLD_CYAN** = "\033[1;36m"
- static const char * **BLUE** = "\033[34m"

4.21.1 Detailed Description

Menu for purchasing service buildings in the game.

This class provides a user interface for selecting and purchasing different types of service buildings, such as Police Stations, Schools, and Hospitals. It extends the BuyMenu class to include specific logic for handling service-related structures.

4.21.2 Constructor & Destructor Documentation

4.21.2.1 BuyServiceMenu()

```
BuyServiceMenu::BuyServiceMenu ()
```

Constructor for BuyServiceMenu. Initializes the menu layout and configurations for service building purchases.

Constructor for BuyServiceMenu. Initializes the base class BuyMenu.

4.21.2.2 ~BuyServiceMenu()

```
BuyServiceMenu::~BuyServiceMenu ()
```

Destructor for BuyServiceMenu. Cleans up any resources or states related to the menu.

Destructor for BuyServiceMenu.

4.21.3 Member Function Documentation

4.21.3.1 buildEntity()

```
void BuyServiceMenu::buildEntity (
            EntityType type,
            Size size,
            int xPos,
            int yPos)  [override], [protected], [virtual]
```

Initiates the construction of the selected service building.

This method interacts with the ServiceManager to create a building at the specified location and size.

Parameters

type	The type of service building to construct (e.g., Police Station, School, Hospital).
size	The size category of the building (e.g., Small, Medium, Large).
xPos	The x-coordinate for the building's placement.
yPos	The y-coordinate for the building's placement.

Implements BuyMenu.

4.21.3.2 chooseEntityType()

```
EntityType BuyServiceMenu::chooseEntityType ()  [override], [protected], [virtual]
```

Displays a list of service building types for user selection.

Override the entity type selection for service buildings.

Overrides the pure virtual method from BuyMenu to present specific service-building options to the user. This method displays building types and handles user input to select an option.

Returns

The selected EntityType representing the chosen service building.

The selected EntityType corresponding to Police Station, School, or Hospital.

Implements BuyMenu.

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

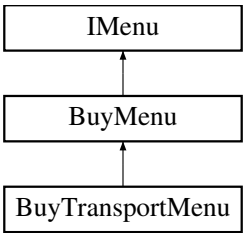
- src/menus/buildings/service/BuyServiceMenu.h
- src/menus/buildings/service/BuyServiceMenu.cpp

4.22 BuyTransportMenu Class Reference

Menu for purchasing transport buildings in the game.

`#include <BuyTransportMenu.h>`

Inheritance diagram for BuyTransportMenu:



Public Member Functions

- BuyTransportMenu ()

  Constructor for BuyTransportMenu. Initializes the menu layout and configurations for transport building purchases.
- ~BuyTransportMenu ()

  Destructor for BuyTransportMenu. Cleans up any resources or states related to the transport menu.

Public Member Functions inherited from BuyMenu

- BuyMenu ()

  Constructs a new BuyMenu object with default initialization.
- virtual ~BuyMenu ()

  Destructor for the BuyMenu class.
- void display () const override

  Displays the base buy menu.
- void handleInput () override

  Handles user input for the buy menu.

Public Member Functions inherited from IMenu

- **IMenu** ()=default

  Default constructor for IMenu.
- IMenu (std::string heading)

  Constructor to initialize a menu with a specified heading.
- virtual ~**IMenu** ()=default

  Virtual destructor for IMenu.
- void setHeading (const std::string &heading)

  Sets the heading of the menu.

Protected Member Functions

- EntityType chooseEntityType () override

  Displays a list of transport building types for user selection.
- void buildEntity (EntityType type, Size size, int xPos, int yPos) override

  Initiates the construction of the selected transport building.

Protected Member Functions inherited from BuyMenu

- Size chooseBuildingSize (EntityType type)

  Allows the user to select the size of the building to be purchased.
- void chooseBuildingPosition (int &xPos, int &yPos, EntityType type, Size size)

  Allows the user to choose the position of the building on the city grid.
- void confirmPurchase (EntityType type, Size size, int xPos, int yPos)

  Confirms the purchase of the selected building.
- void displayAvailablePositions (const std::vector< std::vector< int > > &positions) const override

  Displays available positions on the city grid with visualization of the entity's size.

Protected Member Functions inherited from IMenu

- std::string repeat (const std::string &str, int times) const

  Utility function to repeat a string multiple times.
- int calculateMaxWidth (const std::string &menuHeading, const std::vector< Section > &sections) const

  Calculates the maximum width required for the menu.
- void printTopBorder (int width) const

  Prints the top border of the menu using box-drawing characters.
- void printBottomBorder (int width) const

  Prints the bottom border of the menu using box-drawing characters.
- void printSectionDivider (int width) const

  Prints a section divider in the menu using box-drawing characters.
- void printDoubleLineDivider (int width) const

  Prints a double-line divider for the main heading of the menu.
- std::string centerText (const std::string &text, int width) const

  Centers text within a specified width using space padding.
- std::string centerTextWithChar (const std::string &text, int width, const std::string &padChar) const

  Centers text within a specified width using a custom character for padding.
- void displayMenu () const

  Displays the formatted menu, including sections and options.
- void **displayChoicePrompt** () const

  Displays the choice prompt for user input.
- void displayChoiceMessagePrompt (const std::string &message) const

  Displays a custom message prompt for user input.
- void **displayInvalidChoice** () const

  Displays an error message when the user makes an invalid choice.
- void displayErrorMessage (const std::string &message) const

  Displays a general error message.
- void displaySuccessMessage (const std::string &message) const

  Displays a success message in green color.
- void **displayPressEnterToContinue** () const

  Displays a message asking the user to press Enter to continue.
- void **clearScreen** () const

  Clears the terminal screen.
- std::string stripColorCodes (const std::string &input) const

  Strips ANSI color codes from a string.

Additional Inherited Members

Static Protected Member Functions inherited from IMenu

- static char indexToExtendedChar (int index)

  Converts a numeric index (0-99) to a single character in an extended set.
- static std::string coordinatesToLabel (int x, int y)

  Converts x and y coordinates to a labeled string (e.g., "A, 1").

Protected Attributes inherited from BuyMenu

- EntityType **selectedType**

  The currently selected type of the entity for purchase.
- Size **selectedSize**

  The currently selected size of the entity for purchase.

Protected Attributes inherited from IMenu

- std::vector< Section > **sections**

  List of sections contained in the menu.
- std::string **menuHeading**

  The heading/title of the menu.
- bool **hasExited**

  Flag indicating if the menu has been exited.
- CityManager **cityManager**

  Manager for city-related operations.
- bool **displayResources**

  Flag indicating whether to display resources in the menu.
- bool **isInfoMenu**

  Flag indicating whether to display option numbers.

Static Protected Attributes inherited from IMenu

- static const char * **RESET** = "\033[0m"

  ANSI color codes and styles for use in all menus.
- static const char * **BOLD_WHITE** = "\033[1;37m"
- static const char * **NORMAL_WHITE** = "\033[0;37m"
- static const char * **DARK_GRAY** = "\033[1;30m"
- static const char * **BOLD_YELLOW** = "\033[1;33m"
- static const char * **BOLD_GREEN** = "\033[1;32m"
- static const char * **BOLD_RED** = "\033[1;31m"
- static const char * **BOLD_CYAN** = "\033[1;36m"
- static const char * **BLUE** = "\033[34m"

4.22.1 Detailed Description

Menu for purchasing transport buildings in the game.

This class provides a user interface for selecting and purchasing various types of transport buildings, such as Bus Stops, Airports, and Train Stations. It extends the BuyMenu class to handle transport-specific options and building logic.

4.22.2 Constructor & Destructor Documentation

4.22.2.1 BuyTransportMenu()

```
BuyTransportMenu::BuyTransportMenu ()
```

Constructor for BuyTransportMenu. Initializes the menu layout and configurations for transport building purchases.

Constructor for BuyTransportMenu. Initializes the base BuyMenu for transport-building selection.

4.22.2.2 ~BuyTransportMenu()

```
BuyTransportMenu::~BuyTransportMenu ()
```

Destructor for BuyTransportMenu. Cleans up any resources or states related to the transport menu.

Destructor for BuyTransportMenu.

4.22.3 Member Function Documentation

4.22.3.1 buildEntity()

```
void BuyTransportMenu::buildEntity (
            EntityType type,
            Size size,
            int xPos,
            int yPos)  [override], [protected], [virtual]
```

Initiates the construction of the selected transport building.

This method interacts with the TransportManager to create a building at the specified location and size.

Parameters

type	The type of transport building to construct (e.g., Bus Stop, Airport, Train Station).
size	The size category of the building (e.g., Small, Medium, Large).
xPos	The x-coordinate for the building's placement.
yPos	The y-coordinate for the building's placement.

Implements BuyMenu.

4.22.3.2 chooseEntityType()

```
EntityType BuyTransportMenu::chooseEntityType ()  [override], [protected], [virtual]
```

Displays a list of transport building types for user selection.

Displays the options for selecting a transport building type. The user can choose between Bus Stop, Airport, or Train Station.

Overrides the pure virtual method from BuyMenu to present specific transport-building options to the user. This method displays transport options and processes user input to select an option.

Returns

The selected EntityType representing the chosen transport building.

The selected EntityType for the transport building.

Implements BuyMenu.

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

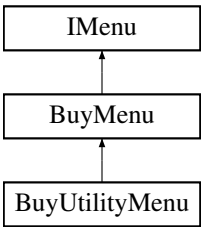
- src/menus/buildings/transport/BuyTransportMenu.h
- src/menus/buildings/transport/BuyTransportMenu.cpp

4.23 BuyUtilityMenu Class Reference

Menu for purchasing utility buildings in the game.

```
#include <BuyUtilityMenu.h>
```

Inheritance diagram for BuyUtilityMenu:



Public Member Functions

- BuyUtilityMenu ()

  Constructor for BuyUtilityMenu. Initializes the menu layout and configurations for utility building purchases.
- ~BuyUtilityMenu ()

  Destructor for BuyUtilityMenu. Cleans up any resources or states related to the utility menu.

Public Member Functions inherited from BuyMenu

- BuyMenu ()

  Constructs a new BuyMenu object with default initialization.
- virtual ~BuyMenu ()

  Destructor for the BuyMenu class.
- void display () const override

  Displays the base buy menu.
- void handleInput () override

  Handles user input for the buy menu.

Public Member Functions inherited from IMenu

- **IMenu** ()=default

  Default constructor for IMenu.
- IMenu (std::string heading)

  Constructor to initialize a menu with a specified heading.
- virtual ~**IMenu** ()=default

  Virtual destructor for IMenu.
- void setHeading (const std::string &heading)

  Sets the heading of the menu.

Protected Member Functions

- EntityType chooseEntityType () override

  Displays a list of utility building types for user selection.
- void buildEntity (EntityType type, Size size, int xPos, int yPos) override

  Initiates the construction of the selected utility building.

Protected Member Functions inherited from BuyMenu

- Size chooseBuildingSize (EntityType type)

  Allows the user to select the size of the building to be purchased.
- void chooseBuildingPosition (int &xPos, int &yPos, EntityType type, Size size)

  Allows the user to choose the position of the building on the city grid.
- void confirmPurchase (EntityType type, Size size, int xPos, int yPos)

  Confirms the purchase of the selected building.
- void displayAvailablePositions (const std::vector< std::vector< int > > &positions) const override

  Displays available positions on the city grid with visualization of the entity's size.

Protected Member Functions inherited from IMenu

- std::string repeat (const std::string &str, int times) const

  Utility function to repeat a string multiple times.
- int calculateMaxWidth (const std::string &menuHeading, const std::vector< Section > &sections) const

  Calculates the maximum width required for the menu.
- void printTopBorder (int width) const

  Prints the top border of the menu using box-drawing characters.
- void printBottomBorder (int width) const

  Prints the bottom border of the menu using box-drawing characters.
- void printSectionDivider (int width) const

  Prints a section divider in the menu using box-drawing characters.
- void printDoubleLineDivider (int width) const

  Prints a double-line divider for the main heading of the menu.
- std::string centerText (const std::string &text, int width) const

  Centers text within a specified width using space padding.
- std::string centerTextWithChar (const std::string &text, int width, const std::string &padChar) const

  Centers text within a specified width using a custom character for padding.
- void displayMenu () const

  Displays the formatted menu, including sections and options.
- void **displayChoicePrompt** () const

  Displays the choice prompt for user input.
- void displayChoiceMessagePrompt (const std::string &message) const

  Displays a custom message prompt for user input.
- void **displayInvalidChoice** () const

  Displays an error message when the user makes an invalid choice.
- void displayErrorMessage (const std::string &message) const

  Displays a general error message.
- void displaySuccessMessage (const std::string &message) const

  Displays a success message in green color.
- void **displayPressEnterToContinue** () const

  Displays a message asking the user to press Enter to continue.
- void **clearScreen** () const

  Clears the terminal screen.
- std::string stripColorCodes (const std::string &input) const

  Strips ANSI color codes from a string.

Additional Inherited Members

Static Protected Member Functions inherited from IMenu

- static char indexToExtendedChar (int index)

  Converts a numeric index (0-99) to a single character in an extended set.
- static std::string coordinatesToLabel (int x, int y)

  Converts x and y coordinates to a labeled string (e.g., "A, 1").

Protected Attributes inherited from BuyMenu

- EntityType **selectedType**

  The currently selected type of the entity for purchase.
- Size **selectedSize**

  The currently selected size of the entity for purchase.

Protected Attributes inherited from IMenu

- std::vector< Section > **sections**

  List of sections contained in the menu.
- std::string **menuHeading**

  The heading/title of the menu.
- bool **hasExited**

  Flag indicating if the menu has been exited.
- CityManager **cityManager**

  Manager for city-related operations.
- bool **displayResources**

  Flag indicating whether to display resources in the menu.
- bool **isInfoMenu**

  Flag indicating whether to display option numbers.

Static Protected Attributes inherited from IMenu

- static const char * **RESET** = "\033[0m"

  ANSI color codes and styles for use in all menus.
- static const char * **BOLD_WHITE** = "\033[1;37m"
- static const char * **NORMAL_WHITE** = "\033[0;37m"
- static const char * **DARK_GRAY** = "\033[1;30m"
- static const char * **BOLD_YELLOW** = "\033[1;33m"
- static const char * **BOLD_GREEN** = "\033[1;32m"
- static const char * **BOLD_RED** = "\033[1;31m"
- static const char * **BOLD_CYAN** = "\033[1;36m"
- static const char * **BLUE** = "\033[34m"

4.23.1 Detailed Description

Menu for purchasing utility buildings in the game.

This class provides a user interface for selecting and purchasing various types of utility buildings, such as Power Plants, Water Supplies, Waste Management facilities, and Sewage Systems. It extends the BuyMenu class to handle utility-specific options and building logic.

4.23.2 Constructor & Destructor Documentation

4.23.2.1 BuyUtilityMenu()

```
BuyUtilityMenu::BuyUtilityMenu ()
```

Constructor for BuyUtilityMenu. Initializes the menu layout and configurations for utility building purchases.

Constructor for BuyUtilityMenu. Initializes the base BuyMenu for utility-building selection.

4.23.2.2 ~BuyUtilityMenu()

```
BuyUtilityMenu::~BuyUtilityMenu ()
```

Destructor for BuyUtilityMenu. Cleans up any resources or states related to the utility menu.

Destructor for BuyUtilityMenu.

4.23.3 Member Function Documentation

4.23.3.1 buildEntity()

```
void BuyUtilityMenu::buildEntity (
            EntityType type,
            Size size,
            int xPos,
            int yPos)  [override], [protected], [virtual]
```

Initiates the construction of the selected utility building.

This method interacts with the UtilityManager to create a building at the specified location and size.

Parameters

type	The type of utility building to construct (e.g., Power Plant, Water Supply, etc.).
size	The size category of the building (e.g., Small, Medium, Large).
xPos	The x-coordinate for the building's placement.
yPos	The y-coordinate for the building's placement.

Implements BuyMenu.

4.23.3.2 chooseEntityType()

```
EntityType BuyUtilityMenu::chooseEntityType ()  [override], [protected], [virtual]
```

Displays a list of utility building types for user selection.

Displays the options for selecting a utility building type. The user can choose between Power Plant, Water Supply, Waste Management, or Sewage System.

Overrides the pure virtual method from BuyMenu to present specific utility-building options to the user. This method displays utility options and processes user input to select an option.

Returns

The selected EntityType representing the chosen utility building.

The selected EntityType for the utility building.

Implements BuyMenu.

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

- src/menus/buildings/utility/BuyUtilityMenu.h
- src/menus/buildings/utility/BuyUtilityMenu.cpp

4.24 Caretaker Class Reference

Class representing a Caretaker for managing Memento objects.

`#include <Caretaker.h>`

Public Member Functions

- void setMemento (Memento *memento)

  Sets a new Memento by adding it to the stored history.
- Memento * getMemento () const

  Retrieves a pointer to the most recent Memento from the stored history.
- ~Caretaker ()

  Destructor to free the memory allocated for mementos.
- std::vector< Memento * > getPastPolicies () const

  Retrieves all past policies stored in the caretaker.

4.24.1 Detailed Description

Class representing a Caretaker for managing Memento objects.

Stores and retrieves the saved states of policies.

4.24.2 Constructor & Destructor Documentation

4.24.2.1 ~Caretaker()

`Caretaker::~Caretaker ()`

Destructor to free the memory allocated for mementos.

Destructor to free memory allocated for Memento pointers.

4.24.3 Member Function Documentation

4.24.3.1 getMemento()

`Memento * Caretaker::getMemento () const`

Retrieves a pointer to the most recent Memento from the stored history.

Returns

Memento* Pointer to the most recent Memento, or nullptr if no Memento exists.

4.24.3.2 getPastPolicies()

```
std::vector< Memento * > Caretaker::getPastPolicies () const
```

Retrieves all past policies stored in the caretaker.

Returns

Vector of pointers to Memento objects representing past policies.

4.24.3.3 setMemento()

```
void Caretaker::setMemento (
            Memento * memento)
```

Sets a new Memento by adding it to the stored history.

Parameters

memento	Pointer to the Memento to be added and set as the most recent.

If a Memento is added, it becomes the most recent entry in the history.

Parameters

memento	Pointer to the new Memento to set.

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

- src/utils/Caretaker.h
- src/utils/Caretaker.cpp

4.25 City Class Reference

Singleton class that represents and manages a simulated city with entities, resources, and policies.

```
#include <City.h>
```

Public Member Functions

- **City** (const City &)=delete

  Deleted copy constructor for singleton pattern.
- City & **operator=** (const City &)=delete

  Deleted assignment operator for singleton pattern.
- Entity * getEntity (int x, int y)

  Retrieves an entity at specified coordinates.
- void addEntity (Entity *entity)

  Adds an entity to the city grid.
- std::vector< std::vector< Entity * > > & getGrid ()

  Provides access to the city grid.
- void deleteEntity (int x, int y)

  Deletes an entity at the specified coordinates.
- void accept (CityVisitor &visitor)

  Accepts a visitor for the city (Visitor Pattern).
- int **getWidth** () const
- int **getHeight** () const
- float **getSatisfaction** () const
- int **getMoney** () const
- int **getWood** () const
- int **getStone** () const
- int **getConcrete** () const
- int **getPopulationCapacity** () const
- int **getPopulation** () const
- int **getElectricityProduction** () const
- int **getElectricityConsumption** () const
- int **getWaterProduction** () const
- int **getWaterConsumption** () const
- int **getWasteProduction** () const
- int **getWasteConsumption** () const
- int **getSewageProduction** () const
- int **getSewageConsumption** () const
- int **getResidentialTax** () const
- int **getEconomicTax** () const
- WaterPolicy * getWaterPolicy () const

  Gets the current water usage policy.
- ElectricityPolicy * getElectricityPolicy () const

  Gets the current electricity usage policy.
- void **setWidth** (int width)
- void **setHeight** (int height)
- void **setSatisfaction** (float satisfaction)
- void **setMoney** (int money)
- void **setWood** (int wood)
- void **setStone** (int stone)
- void **setConcrete** (int concrete)
- void **setPopulationCapacity** (int populationCapacity)
- void **setPopulation** (int population)
- void **setElectricityProduction** (int electricityProduction)
- void **setElectricityConsumption** (int electricityConsumption)
- void **setWaterProduction** (int waterProduction)
- void **setWaterConsumption** (int waterConsumption)
- void **setWasteProduction** (int wasteProduction)

- void **setWasteConsumption** (int wasteConsumption)
- void **setSewageProduction** (int sewageProduction)
- void **setSewageConsumption** (int sewageConsumption)
- void **setResidentialTax** (int residentialTax)
- void **setEconomicTax** (int economicTax)
- void setWaterPolicy (PolicyType policyType)

  Sets the water usage policy for the city.
- void setElectricityPolicy (PolicyType policyType)

  Sets the electricity usage policy for the city.
- void reset (int width, int height)

  Resets the city with current or specified dimensions.
- void **reset** ()
- Iterator * createCityIterator (bool unique)

  Creates various iterators for the city.
- Iterator * **createBuildingIterator** (bool unique)
- Iterator * **createUtilityIterator** (bool unique)
- Iterator * **createIndustryIterator** (bool unique)
- Iterator * **createRoadIterator** (bool unique)
- Iterator * **createTransportIterator** (bool unique)
- Iterator * **createEconomicBuildingIterator** (bool unique)
- Iterator * **createResidentialBuildingIterator** (bool unique)
- Iterator * **createServiceBuildingIterator** (bool unique)
- Iterator * **createAmenityIterator** (bool unique)
- Iterator * **createPowerPlantIterator** (bool unique)
- Iterator * **createWaterSupplyIterator** (bool unique)
- Iterator * **createWasteManagementIterator** (bool unique)
- Iterator * **createSewageSystemIterator** (bool unique)
- Iterator * **createConcreteProducerIterator** (bool unique)
- Iterator * **createStoneProducerIterator** (bool unique)
- Iterator * **createWoodProducerIterator** (bool unique)
- void **createRandomRoad** ()

  Creates a road entity at a random position within the city grid.
- void **displayCity** () const

  Displays the city layout on the console.

Static Public Member Functions

- static City * instance ()

  Returns a pointer to the singleton instance of City.

4.25.1 Detailed Description

Singleton class that represents and manages a simulated city with entities, resources, and policies.

4.25.2 Member Function Documentation

4.25.2.1 accept()

```
void City::accept (
            CityVisitor & visitor)
```

Accepts a visitor for the city (Visitor Pattern).

Parameters

visitor	The CityVisitor object.

4.25.2.2 addEntity()

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

Adds an entity to the city grid.

Parameters

entity	Pointer to the Entity to be added.

4.25.2.3 createCityIterator()

```
Iterator * City::createCityIterator (
            bool unique)
```

Creates various iterators for the city.

Parameters

unique	If true, creates unique iterators.

Returns

A pointer to the created iterator.

4.25.2.4 deleteEntity()

```
void City::deleteEntity (
            int x,
            int y)
```

Deletes an entity at the specified coordinates.

Parameters

x	X-coordinate of the entity.
y	Y-coordinate of the entity.

4.25.2.5 getElectricityPolicy()

```
ElectricityPolicy * City::getElectricityPolicy () const
```

Gets the current electricity usage policy.

Returns

Pointer to the current electricity policy.

4.25.2.6 getEntity()

```
Entity * City::getEntity (
            int x,
            int y)
```

Retrieves an entity at specified coordinates.

Parameters

x	X-coordinate on the grid.
y	Y-coordinate on the grid.

Returns

Pointer to the Entity at the coordinates, or nullptr if empty.

4.25.2.7 getGrid()

```
std::vector< std::vector< Entity * > > & City::getGrid ()
```

Provides access to the city grid.

Returns

A reference to the 2D grid of entities.

4.25.2.8 getWaterPolicy()

```
WaterPolicy * City::getWaterPolicy () const
```

Gets the current water usage policy.

Returns

Pointer to the current water policy.

4.25.2.9 instance()

```
City * City::instance ()  [static]
```

Returns a pointer to the singleton instance of City.

Returns

A pointer to the City instance.

4.25.2.10 reset()

```
void City::reset (
            int width,
            int height)
```

Resets the city with current or specified dimensions.

Parameters

width	Width of the new grid.
height	Height of the new grid.

4.25.2.11 setElectricityPolicy()

```
void City::setElectricityPolicy (
            PolicyType policyType)
```

Sets the electricity usage policy for the city.

Parameters

policyType	The type of electricity policy to enact.

4.25.2.12 setWaterPolicy()

```
void City::setWaterPolicy (
            PolicyType policyType)
```

Sets the water usage policy for the city.

Parameters

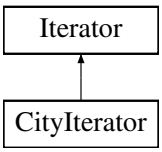
policyType	The type of water policy to enact.

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

- src/city/City.h
- src/city/City.cpp

4.26 CityIterator Class Reference

Inheritance diagram for CityIterator:



Public Member Functions

- **CityIterator** ()

  Default constructor for CityIterator.
- CityIterator (std::vector< std::vector< Entity * > > &grid, bool unique=true)

  Construct a new City Iterator:: City Iterator object with unique option.
- void first () override

  Goes to the first Entity, respecting uniqueness if enabled.
- void next () override

  Advances to the next Entity, respecting uniqueness if enabled.
- bool hasNext () override

  Checks if there is another unvisited Entity.
- Entity * current () override

  Returns the current Entity.

Public Member Functions inherited from Iterator

- **Iterator** ()

  Construct a new Iterator object, initializing row and column to zero.
- virtual ~**Iterator** ()

  Destroy the Iterator object.
- **Iterator** (std::vector< std::vector< Entity * > > &grid)
- virtual int getRow ()

  Get the current row index of the iterator.
- virtual int getCol ()

  Get the current column index of the iterator.

Additional Inherited Members

Protected Member Functions inherited from Iterator

- bool isVisited (Entity *entity)

  Check if the specified entity has been visited.
- void markVisited (Entity *entity)

  Mark the specified entity as visited.

Protected Attributes inherited from Iterator

- std::vector< std::vector< Entity * > > **grid**
- std::vector< std::vector< Entity * > >::iterator **currRow**
- std::vector< Entity * >::iterator **curr**
- int **row**
- int **col**
- std::unordered_set< Entity * > **visitedEntities**

4.26.1 Constructor & Destructor Documentation

4.26.1.1 CityIterator()

```
CityIterator::CityIterator (
            std::vector< std::vector< Entity * > > & grid,
            bool unique = true)
```

Construct a new City Iterator:: City Iterator object with unique option.

Parameters

grid	
unique	

4.26.2 Member Function Documentation

4.26.2.1 current()

```
Entity * CityIterator::current ()  [override], [virtual]
```

Returns the current Entity.

Returns

Entity*

Implements Iterator.

4.26.2.2 first()

```
void CityIterator::first ()  [override], [virtual]
```

Goes to the first Entity, respecting uniqueness if enabled.

Implements Iterator.

4.26.2.3 hasNext()

```
bool CityIterator::hasNext ()  [override], [virtual]
```

Checks if there is another unvisited Entity.

Returns

true if there is another unvisited Entity, false otherwise

Implements Iterator.

4.26.2.4 next()

```
void CityIterator::next ()  [override], [virtual]
```

Advances to the next Entity, respecting uniqueness if enabled.

Implements Iterator.

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

- src/iterators/city/CityIterator.h
- src/iterators/city/CityIterator.cpp

4.27 CityManager Class Reference

Manages and maintains city entities and provides functions for updating city states.

```
#include <CityManager.h>
```

Public Member Functions

- **CityManager** ()
  Constructs a CityManager instance.
- ~**CityManager** ()
  Destroys the CityManager instance.
- void **initializeCity** ()
  Initializes the city, creating required instances.
- void **updateCity** ()
  Updates city attributes and entities for the next simulation loop.
- Entity * getEntity (int x, int y)
  Retrieves an entity at the specified coordinates.
- void sellBuilding (int xPos, int yPos)
  Sells a building at the specified coordinates.
- void sellAllBuildingsOfType (EntityType type)
  Sells all buildings of a specified type.
- std::vector< std::vector< int > > getAvailiablePositions (EntityType type, Size size)
  Gets available positions for placing a specified entity type.

- bool canAffordToBuy (EntityType type, Size size)

  Checks if the city can afford to buy an entity of a specified type and size.
- bool canBuyAt (int xPos, int yPos, EntityType type, Size size)

  Checks if an entity of a specified type and size can be placed at a specific position.
- bool buyEntity (EntityType type, Size size)

  Attempts to purchase and place an entity of a specified type and size.
- void generateCity (std::optional< unsigned int > seed=std::nullopt)

  Generates a random city with an optional seed for reproducibility.
- void generateRandomRoads (int gridWidth, int gridHeight, int minWidth, int minHeight, int roadGap)

  Generates random roads within the specified grid dimensions.
- void generateRandomBuildings (int placementProbability)

  Generates random buildings based on a specified placement probability.

4.27.1 Detailed Description

Manages and maintains city entities and provides functions for updating city states.

4.27.2 Member Function Documentation

4.27.2.1 buyEntity()

```
bool CityManager::buyEntity (
            EntityType type,
            Size size)
```

Attempts to purchase and place an entity of a specified type and size.

Parameters

type	The type of entity.
size	The size of the entity.

Returns

True if the entity was successfully bought, false otherwise.

4.27.2.2 canAffordToBuy()

```
bool CityManager::canAffordToBuy (
            EntityType type,
            Size size)
```

Checks if the city can afford to buy an entity of a specified type and size.

Parameters

type	The type of entity.
size	The size of the entity.

Returns

True if affordable, false otherwise.

4.27.2.3 canBuyAt()

```
bool CityManager::canBuyAt (
            int xPos,
            int yPos,
            EntityType type,
            Size size)
```

Checks if an entity of a specified type and size can be placed at a specific position.

Parameters

xPos	X-coordinate.
yPos	Y-coordinate.
type	The type of entity.
size	The size of the entity.

Returns

True if the entity can be placed, false otherwise.

4.27.2.4 generateCity()

```
void CityManager::generateCity (
            std::optional< unsigned int > seed = std::nullopt)
```

Generates a random city with an optional seed for reproducibility.

This function creates a new city layout based on a random generation algorithm. The seed can be specified to produce consistent results for testing. This function is used for testing and demonstration purposes.

Parameters

seed	An optional seed value for the random number generator. If not provided, a random seed is used.

4.27.2.5 generateRandomBuildings()

```
void CityManager::generateRandomBuildings (
            int placementProbability)
```

Generates random buildings based on a specified placement probability.

This function places buildings randomly across the city grid, with the likelihood of placement determined by the provided probability value.

Parameters

placementProbability	The probability of placing a building in each grid cell.

4.27.2.6 generateRandomRoads()

```
void CityManager::generateRandomRoads (
            int gridWidth,
            int gridHeight,
            int minWidth,
            int minHeight,
            int roadGap)
```

Generates random roads within the specified grid dimensions.

This function creates random roads on the city grid, ensuring they fit within the specified dimensions and have a minimum width and height.

Parameters

gridWidth	The width of the grid in which to generate roads.
gridHeight	The height of the grid in which to generate roads.
minWidth	The minimum width of each road segment.
minHeight	The minimum height of each road segment.
roadGap	The gap between consecutive road segments to avoid overlaps.

4.27.2.7 getAvailiablePositions()

```
std::vector< std::vector< int > > CityManager::getAvailiablePositions (
            EntityType type,
            Size size)
```

Gets available positions for placing a specified entity type.

Parameters

type	The type of entity.
size	The size of the entity.

Returns

Vector of available positions as (x, y) coordinates.

4.27.2.8 getEntity()

```
Entity * CityManager::getEntity (
            int x,
            int y)
```

Retrieves an entity at the specified coordinates.

Parameters

x	X-coordinate.
y	Y-coordinate.

Returns

Pointer to the entity at (x, y).

4.27.2.9 sellAllBuildingsOfType()

```
void CityManager::sellAllBuildingsOfType (
            EntityType type)
```

Sells all buildings of a specified type.

Parameters

type	The type of entity to sell.

Cant sell a road

4.27.2.10 sellBuilding()

```
void CityManager::sellBuilding (
            int xPos,
            int yPos)
```

Sells a building at the specified coordinates.

Parameters

xPos	X-coordinate of the building.
yPos	Y-coordinate of the building.

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

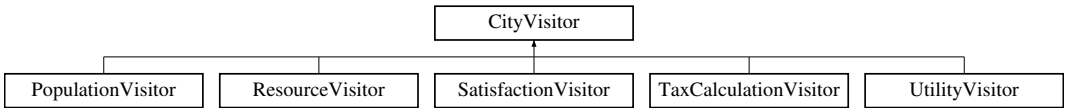
- src/managers/CityManager.h
- src/managers/CityManager.cpp

4.28 CityVisitor Class Reference

Base class for visiting and interacting with City objects.

```
#include <CityVisitor.h>
```

Inheritance diagram for CityVisitor:



Public Member Functions

- **CityVisitor** ()=default
  Default constructor.
- virtual ~**CityVisitor** ()=default
  Default destructor.
- virtual void visit (City *city)=0
  Abstract method to visit a City object.

4.28.1 Detailed Description

Base class for visiting and interacting with City objects.

4.28.2 Member Function Documentation

4.28.2.1 visit()

```
virtual void CityVisitor::visit (
            City * city)  [pure virtual]
```

Abstract method to visit a City object.

Parameters

city	Pointer to the City object to be visited.

Implemented in PopulationVisitor, ResourceVisitor, SatisfactionVisitor, TaxCalculationVisitor, and UtilityVisitor.

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

- src/visitors/base/CityVisitor.h

4.29 CivZero Class Reference

The main game engine class for CivZero.

```
#include <CivZero.h>
```

Public Member Functions

- **CivZero** (const CivZero &)=delete
- CivZero & **operator=** (const CivZero &)=delete
- void startGame (bool generateRandomCity=false, std::optional< unsigned int > seed=std::nullopt)
  Start the game with options for random city generation and a seed.
- void **quit** ()
  Quit the game.
- void **incrementGameLoop** ()
  Increment the game loop counter.
- int getGameLoop ()
  Get the current game loop number.

Static Public Member Functions

- static CivZero & instance ()

  Get the single instance of CivZero.

4.29.1 Detailed Description

The main game engine class for CivZero.

This class implements the singleton pattern to manage game state and control the game loop.

4.29.2 Member Function Documentation

4.29.2.1 getGameLoop()

```
int CivZero::getGameLoop ()
```

Get the current game loop number.

Returns

int The current game loop number.

4.29.2.2 instance()

```
CivZero & CivZero::instance ()  [static]
```

Get the single instance of CivZero.

Returns

CivZero& Reference to the singleton instance of CivZero.

4.29.2.3 startGame()

```
void CivZero::startGame (
            bool generateRandomCity = false,
            std::optional< unsigned int > seed = std::nullopt)
```

Start the game with options for random city generation and a seed.

Parameters

generateRandomCity	If true, generate a random city.
seed	Optional seed for random generation.

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

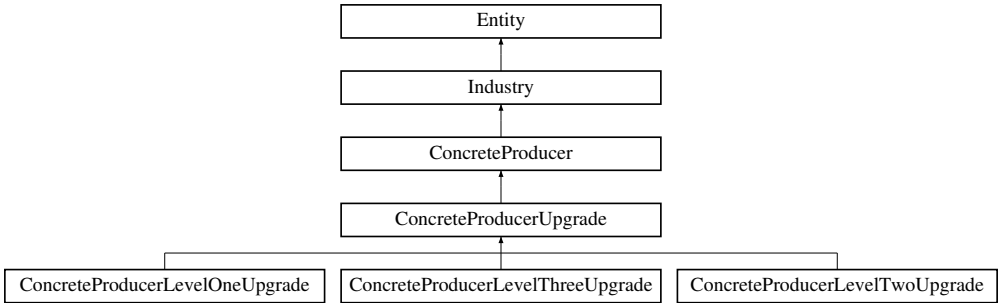
- src/city/CivZero.h
- src/city/CivZero.cpp

4.30 ConcreteProducer Class Reference

Represents a concrete producer industry entity.

`#include <ConcreteProducer.h>`

Inheritance diagram for ConcreteProducer:



Public Member Functions

- ConcreteProducer (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a ConcreteProducer entity with specified attributes.
- ConcreteProducer (ConcreteProducer ∗concreteProducer)

  Copy constructor for the ConcreteProducer class.
- virtual ∼**ConcreteProducer** ()

  Virtual destructor for the ConcreteProducer class.
- void update () override

  Updates the state of the concrete producer.
- Entity ∗ clone () override

  Creates a clone of the ConcreteProducer entity.
- Entity ∗ upgrade () override

  Upgrades the current ConcreteProducer to the next level.

Public Member Functions inherited from Industry

- Industry (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Industry entity with specified attributes.
- Industry (Industry ∗industry)

  Copy constructor for the Industry class.
- virtual ∼**Industry** ()

  Virtual destructor for the Industry class.
- virtual int getOutput ()

  Gets the production output of the industry.
- void setOutput (int output)

  Sets the production output of the industry.
- virtual int getLevel ()

  Gets the current level of the industry.
- virtual Cost getCost ()

  Gets the cost of an upgrade.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).
- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.30.1 Detailed Description

Represents a concrete producer industry entity.

This class manages the concrete production process and interacts with residential buildings in the game.

4.30.2 Constructor & Destructor Documentation

4.30.2.1 ConcreteProducer() [1/2]

```
ConcreteProducer::ConcreteProducer (
            EntityConfig ec,
            Size size,
            int xPos,
            int yPos)
```

Constructs a ConcreteProducer entity with specified attributes.

Parameters

ec	Configuration settings for the entity.
size	Size of the concrete producer entity.
xPos	X-coordinate position of the entity.
yPos	Y-coordinate position of the entity.

4.30.2.2 ConcreteProducer() [2/2]

```
ConcreteProducer::ConcreteProducer (
            ConcreteProducer * concreteProducer)
```

Copy constructor for the ConcreteProducer class.

Creates a new ConcreteProducer entity by copying the attributes of an existing ConcreteProducer.

Parameters

concreteProducer	Pointer to the ConcreteProducer object to be copied.

4.30.3 Member Function Documentation

4.30.3.1 clone()

```
Entity * ConcreteProducer::clone ()  [override], [virtual]
```

Creates a clone of the ConcreteProducer entity.

Returns

A pointer to the cloned ConcreteProducer entity.

Implements Industry.

Reimplemented in ConcreteProducerLevelOneUpgrade, ConcreteProducerLevelThreeUpgrade, ConcreteProducerLevelTwoUpgrade, and ConcreteProducerUpgrade.

4.30.3.2 update()

```
void ConcreteProducer::update ()  [override], [virtual]
```

Updates the state of the concrete producer.

This method notifies residential buildings of changes and updates the build state if necessary.

Implements Industry.

Reimplemented in ConcreteProducerLevelOneUpgrade, ConcreteProducerLevelThreeUpgrade, ConcreteProducerLevelTwoUpgrade, and ConcreteProducerUpgrade.

4.30.3.3 upgrade()

```
Entity * ConcreteProducer::upgrade ()  [override], [virtual]
```

Upgrades the current ConcreteProducer to the next level.

Returns

A pointer to the upgraded ConcreteProducerLevelOneUpgrade instance.

Implements Industry.

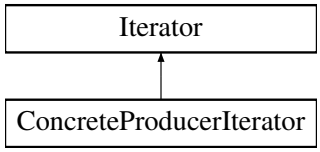
Reimplemented in ConcreteProducerLevelOneUpgrade, ConcreteProducerLevelThreeUpgrade, ConcreteProducerLevelTwoUpgrade, and ConcreteProducerUpgrade.

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

- src/entities/industry/concreteproducer/ConcreteProducer.h
- src/entities/industry/concreteproducer/ConcreteProducer.cpp

4.31 ConcreteProducerIterator Class Reference

Inheritance diagram for ConcreteProducerIterator:



Public Member Functions

- **ConcreteProducerIterator** ()

  Construct a new Concrete Producer Iterator:: Concrete Producer Iterator object.
- ~**ConcreteProducerIterator** ()

  Destroy the Concrete Producer Iterator:: Concrete Producer Iterator object.
- ConcreteProducerIterator (std::vector< std::vector< Entity * > > &grid)

  Construct a new Concrete Producer Iterator:: Concrete Producer Iterator object.
- void first ()

  Sets the iterator to the first unvisited ConcreteProducer.
- void next ()

  Advances to the next unvisited ConcreteProducer.
- bool hasNext ()

  Checks if there is another unvisited ConcreteProducer.
- Entity * current ()

  Returns the current ConcreteProducer.

Public Member Functions inherited from Iterator

- **Iterator** ()

  Construct a new Iterator object, initializing row and column to zero.
- virtual ~**Iterator** ()

  Destroy the Iterator object.
- **Iterator** (std::vector< std::vector< Entity * > > &grid)
- virtual int getRow ()

  Get the current row index of the iterator.
- virtual int getCol ()

  Get the current column index of the iterator.

Additional Inherited Members

Protected Member Functions inherited from Iterator

- bool isVisited (Entity *entity)

  Check if the specified entity has been visited.
- void markVisited (Entity *entity)

  Mark the specified entity as visited.

Protected Attributes inherited from Iterator

- std::vector< std::vector< Entity * > > **grid**
- std::vector< std::vector< Entity * > >::iterator **currRow**
- std::vector< Entity * >::iterator **curr**
- int **row**
- int **col**
- std::unordered_set< Entity * > **visitedEntities**

4.31.1 Constructor & Destructor Documentation

4.31.1.1 ConcreteProducerIterator()

```
ConcreteProducerIterator::ConcreteProducerIterator (
            std::vector< std::vector< Entity * > > & grid)
```

Construct a new Concrete Producer Iterator:: Concrete Producer Iterator object.

Parameters

grid	

4.31.2 Member Function Documentation

4.31.2.1 current()

```
Entity * ConcreteProducerIterator::current ()  [virtual]
```

Returns the current ConcreteProducer.

Returns

Entity*

Implements Iterator.

4.31.2.2 first()

```
void ConcreteProducerIterator::first ()  [virtual]
```

Sets the iterator to the first unvisited ConcreteProducer.

Implements Iterator.

4.31.2.3 hasNext()

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

Checks if there is another unvisited ConcreteProducer.

Returns

true if there is another unvisited ConcreteProducer, false otherwise

Implements Iterator.

4.31.2.4 next()

```
void ConcreteProducerIterator::next ()  [virtual]
```

Advances to the next unvisited ConcreteProducer.

Implements Iterator.

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

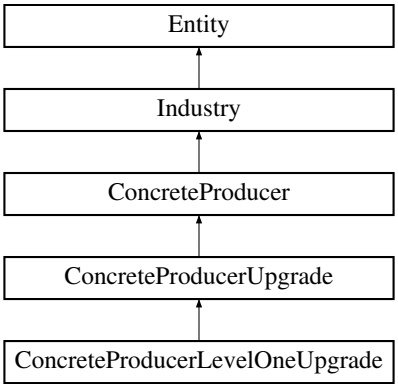
- src/iterators/industry/ConcreteProducerIterator.h
- src/iterators/industry/ConcreteProducerIterator.cpp

4.32 ConcreteProducerLevelOneUpgrade Class Reference

Represents the level one upgrade of a ConcreteProducer entity.

`#include <ConcreteProducerLevelOneUpgrade.h>`

Inheritance diagram for ConcreteProducerLevelOneUpgrade:



Public Member Functions

- ConcreteProducerLevelOneUpgrade (ConcreteProducer *concreteProd)

  Constructs a ConcreteProducerLevelOneUpgrade from a ConcreteProducer instance.
- ConcreteProducerLevelOneUpgrade (ConcreteProducerLevelOneUpgrade *concreteProd)

  Copy constructor for the ConcreteProducerLevelOneUpgrade class.
- ~**ConcreteProducerLevelOneUpgrade** ()

  Destructor for the ConcreteProducerLevelOneUpgrade class.
- void update () override

  Updates the state of the concrete producer.
- int getOutput () override

  Gets the output production of the upgraded concrete producer.
- int getLevel () override

  Gets the upgrade level of the concrete producer.
- Entity * clone () override

  Creates a clone of the ConcreteProducerLevelOneUpgrade entity.
- Entity * upgrade () override

  Upgrades the current concrete producer to the next level.
- Cost getCost () override

  Gets the cost of upgrading to the next level.

Public Member Functions inherited from ConcreteProducerUpgrade

- ConcreteProducerUpgrade (ConcreteProducer *concreteProd)

  Constructs a ConcreteProducerUpgrade from a ConcreteProducer instance.
- ConcreteProducerUpgrade (ConcreteProducerUpgrade *concreteProd)

  Copy constructor for the ConcreteProducerUpgrade class.
- virtual ~**ConcreteProducerUpgrade** ()

  Virtual destructor for the ConcreteProducerUpgrade class.

Public Member Functions inherited from ConcreteProducer

- ConcreteProducer (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a ConcreteProducer entity with specified attributes.
- ConcreteProducer (ConcreteProducer ∗concreteProducer)

  Copy constructor for the ConcreteProducer class.
- virtual ∼**ConcreteProducer** ()

  Virtual destructor for the ConcreteProducer class.

Public Member Functions inherited from Industry

- Industry (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Industry entity with specified attributes.
- Industry (Industry ∗industry)

  Copy constructor for the Industry class.
- virtual ∼**Industry** ()

  Virtual destructor for the Industry class.
- void setOutput (int output)

  Sets the production output of the industry.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity ∗entity)

  Copy constructor for the Entity class.
- virtual ∼**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity ∗entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

Sets the symbol of the entity.

- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from ConcreteProducerUpgrade

- ConcreteProducer * **concreteProducer**

  Pointer to the associated ConcreteProducer instance.

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).

- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.32.1 Detailed Description

Represents the level one upgrade of a ConcreteProducer entity.

This class extends ConcreteProducerUpgrade to provide specific behavior and properties for the first level upgrade of the concrete producer.

4.32.2 Constructor & Destructor Documentation

4.32.2.1 ConcreteProducerLevelOneUpgrade() [1/2]

```
ConcreteProducerLevelOneUpgrade::ConcreteProducerLevelOneUpgrade (
            ConcreteProducer * concreteProd)
```

Constructs a ConcreteProducerLevelOneUpgrade from a ConcreteProducer instance.

Parameters

concreteProd	Pointer to the ConcreteProducer to be upgraded.

4.32.2.2 ConcreteProducerLevelOneUpgrade() [2/2]

```
ConcreteProducerLevelOneUpgrade::ConcreteProducerLevelOneUpgrade (
            ConcreteProducerLevelOneUpgrade * concreteProd)
```

Copy constructor for the ConcreteProducerLevelOneUpgrade class.

Creates a new ConcreteProducerLevelOneUpgrade by copying an existing instance.

Parameters

concreteProd	Pointer to the ConcreteProducerLevelOneUpgrade object to be copied.

4.32.3 Member Function Documentation

4.32.3.1 clone()

```
Entity * ConcreteProducerLevelOneUpgrade::clone ()  [override], [virtual]
```

Creates a clone of the ConcreteProducerLevelOneUpgrade entity.

Returns

A pointer to the cloned ConcreteProducerLevelOneUpgrade instance.

Implements ConcreteProducerUpgrade.

4.32.3.2 getCost()

```
Cost ConcreteProducerLevelOneUpgrade::getCost ()  [override], [virtual]
```

Gets the cost of upgrading to the next level.

Returns

The cost associated with the level one upgrade.

Implements ConcreteProducerUpgrade.

4.32.3.3 getLevel()

```
int ConcreteProducerLevelOneUpgrade::getLevel ()  [override], [virtual]
```

Gets the upgrade level of the concrete producer.

Returns

The level of the upgrade, which is 1 for this class.

Reimplemented from Industry.

4.32.3.4 getOutput()

```
int ConcreteProducerLevelOneUpgrade::getOutput ()  [override], [virtual]
```

Gets the output production of the upgraded concrete producer.

Returns

The production output value after applying the upgrade multiplier.

Implements ConcreteProducerUpgrade.

4.32.3.5 update()

```
void ConcreteProducerLevelOneUpgrade::update ()  [override], [virtual]
```

Updates the state of the concrete producer.

This method invokes the update on the associated concrete producer.

Implements ConcreteProducerUpgrade.

4.32.3.6 upgrade()

```
Entity * ConcreteProducerLevelOneUpgrade::upgrade ()  [override], [virtual]
```

Upgrades the current concrete producer to the next level.

Returns

A pointer to the upgraded entity, specifically a ConcreteProducerLevelTwoUpgrade.

Implements ConcreteProducerUpgrade.

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

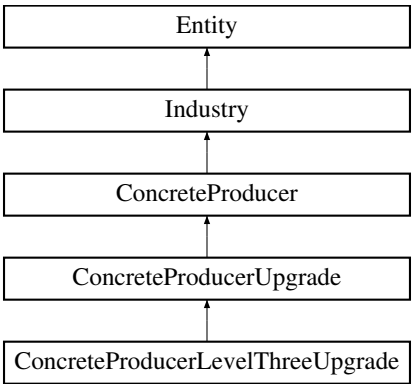
- src/entities/industry/concreteproducer/ConcreteProducerLevelOneUpgrade.h
- src/entities/industry/concreteproducer/ConcreteProducerLevelOneUpgrade.cpp

4.33 ConcreteProducerLevelThreeUpgrade Class Reference

Represents the level three upgrade of a ConcreteProducer entity.

```
#include <ConcreteProducerLevelThreeUpgrade.h>
```

Inheritance diagram for ConcreteProducerLevelThreeUpgrade:



Public Member Functions

- ConcreteProducerLevelThreeUpgrade (ConcreteProducer *concreteProd)

  Constructs a ConcreteProducerLevelThreeUpgrade from a ConcreteProducer instance.
- ConcreteProducerLevelThreeUpgrade (ConcreteProducerLevelThreeUpgrade *concreteProd)

  Copy constructor for the ConcreteProducerLevelThreeUpgrade class.
- ~**ConcreteProducerLevelThreeUpgrade** ()

  Destructor for the ConcreteProducerLevelThreeUpgrade class.
- void update () override

  Updates the state of the concrete producer.
- int getOutput () override

  Gets the output production of the upgraded concrete producer.
- int getLevel () override

  Gets the upgrade level of the concrete producer.
- Entity * clone () override

  Creates a clone of the ConcreteProducerLevelThreeUpgrade entity.
- Cost getCost () override

  Gets the cost of upgrading to the next level.
- Entity * upgrade () override

  Upgrades the current concrete producer to the next level.

Public Member Functions inherited from ConcreteProducerUpgrade

- ConcreteProducerUpgrade (ConcreteProducer *concreteProd)

  Constructs a ConcreteProducerUpgrade from a ConcreteProducer instance.
- ConcreteProducerUpgrade (ConcreteProducerUpgrade *concreteProd)

  Copy constructor for the ConcreteProducerUpgrade class.
- virtual ~**ConcreteProducerUpgrade** ()

  Virtual destructor for the ConcreteProducerUpgrade class.

Public Member Functions inherited from ConcreteProducer

- ConcreteProducer (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a ConcreteProducer entity with specified attributes.
- ConcreteProducer (ConcreteProducer *concreteProducer)

  Copy constructor for the ConcreteProducer class.
- virtual ~**ConcreteProducer** ()

  Virtual destructor for the ConcreteProducer class.

Public Member Functions inherited from Industry

- Industry (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Industry entity with specified attributes.
- Industry (Industry *industry)

  Copy constructor for the Industry class.
- virtual ~**Industry** ()

  Virtual destructor for the Industry class.
- void setOutput (int output)

  Sets the production output of the industry.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from ConcreteProducerUpgrade

- ConcreteProducer * **concreteProducer**

  Pointer to the associated ConcreteProducer instance.

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).
- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.33.1 Detailed Description

Represents the level three upgrade of a ConcreteProducer entity.

This class extends ConcreteProducerUpgrade to provide specific behavior and properties for the third level upgrade of the concrete producer.

4.33.2 Constructor & Destructor Documentation

4.33.2.1 ConcreteProducerLevelThreeUpgrade() [1/2]

```
ConcreteProducerLevelThreeUpgrade::ConcreteProducerLevelThreeUpgrade (
            ConcreteProducer * concreteProd)
```

Constructs a ConcreteProducerLevelThreeUpgrade from a ConcreteProducer instance.

Parameters

concreteProd	Pointer to the ConcreteProducer to be upgraded.

4.33.2.2 ConcreteProducerLevelThreeUpgrade() [2/2]

```
ConcreteProducerLevelThreeUpgrade::ConcreteProducerLevelThreeUpgrade (
            ConcreteProducerLevelThreeUpgrade * concreteProd)
```

Copy constructor for the ConcreteProducerLevelThreeUpgrade class.

Creates a new ConcreteProducerLevelThreeUpgrade by copying an existing instance.

Parameters

concreteProd	Pointer to the ConcreteProducerLevelThreeUpgrade object to be copied.

4.33.3 Member Function Documentation

4.33.3.1 clone()

```
Entity * ConcreteProducerLevelThreeUpgrade::clone ()  [override], [virtual]
```

Creates a clone of the ConcreteProducerLevelThreeUpgrade entity.

Returns

A pointer to the cloned ConcreteProducerLevelThreeUpgrade instance.

Implements ConcreteProducerUpgrade.

4.33.3.2 getCost()

```
Cost ConcreteProducerLevelThreeUpgrade::getCost ()  [override], [virtual]
```

Gets the cost of upgrading to the next level.

Returns

The cost associated with the level three upgrade.

Implements ConcreteProducerUpgrade.

4.33.3.3 getLevel()

```
int ConcreteProducerLevelThreeUpgrade::getLevel ()  [override], [virtual]
```

Gets the upgrade level of the concrete producer.

Returns

The level of the upgrade, which is 3 for this class.

Reimplemented from Industry.

4.33.3.4 getOutput()

```
int ConcreteProducerLevelThreeUpgrade::getOutput ()  [override], [virtual]
```

Gets the output production of the upgraded concrete producer.

Returns

The production output value after applying the upgrade multiplier.

Implements ConcreteProducerUpgrade.

4.33.3.5 update()

```
void ConcreteProducerLevelThreeUpgrade::update ()  [override], [virtual]
```

Updates the state of the concrete producer.

This method invokes the update on the associated concrete producer.

Implements ConcreteProducerUpgrade.

4.33.3.6 upgrade()

```
Entity * ConcreteProducerLevelThreeUpgrade::upgrade ()  [override], [virtual]
```

Upgrades the current concrete producer to the next level.

Returns

nullptr, as this is the final upgrade level.

Implements ConcreteProducerUpgrade.

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

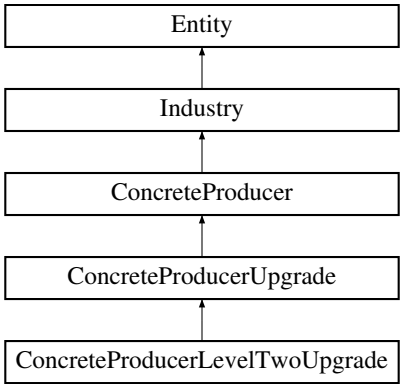
- src/entities/industry/concreteproducer/ConcreteProducerLevelThreeUpgrade.h
- src/entities/industry/concreteproducer/ConcreteProducerLevelThreeUpgrade.cpp

4.34 ConcreteProducerLevelTwoUpgrade Class Reference

Represents the level two upgrade of a ConcreteProducer entity.

`#include <ConcreteProducerLevelTwoUpgrade.h>`

Inheritance diagram for ConcreteProducerLevelTwoUpgrade:



Public Member Functions

- ConcreteProducerLevelTwoUpgrade (ConcreteProducer *concreteProd)

  Constructs a ConcreteProducerLevelTwoUpgrade from a ConcreteProducer instance.
- ConcreteProducerLevelTwoUpgrade (ConcreteProducerLevelTwoUpgrade *concreteProd)

  Copy constructor for the ConcreteProducerLevelTwoUpgrade class.
- ~**ConcreteProducerLevelTwoUpgrade** ()

  Destructor for the ConcreteProducerLevelTwoUpgrade class.
- void update () override

  Updates the state of the concrete producer.
- int getOutput () override

  Gets the output production of the upgraded concrete producer.
- int getLevel () override

  Gets the upgrade level of the concrete producer.
- Entity * clone () override

  Creates a clone of the ConcreteProducerLevelTwoUpgrade entity.
- Entity * upgrade () override

  Upgrades the current concrete producer to the next level.
- Cost getCost () override

  Gets the cost of upgrading to the next level.

Public Member Functions inherited from ConcreteProducerUpgrade

- ConcreteProducerUpgrade (ConcreteProducer *concreteProd)

  Constructs a ConcreteProducerUpgrade from a ConcreteProducer instance.
- ConcreteProducerUpgrade (ConcreteProducerUpgrade *concreteProd)

  Copy constructor for the ConcreteProducerUpgrade class.
- virtual ~**ConcreteProducerUpgrade** ()

  Virtual destructor for the ConcreteProducerUpgrade class.

Public Member Functions inherited from ConcreteProducer

- ConcreteProducer (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a ConcreteProducer entity with specified attributes.
- ConcreteProducer (ConcreteProducer *concreteProducer)

  Copy constructor for the ConcreteProducer class.
- virtual ~**ConcreteProducer** ()

  Virtual destructor for the ConcreteProducer class.

Public Member Functions inherited from Industry

- Industry (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Industry entity with specified attributes.
- Industry (Industry *industry)

  Copy constructor for the Industry class.
- virtual ~**Industry** ()

  Virtual destructor for the Industry class.
- void setOutput (int output)

  Sets the production output of the industry.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

Sets the symbol of the entity.

- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from ConcreteProducerUpgrade

- ConcreteProducer * **concreteProducer**

  Pointer to the associated ConcreteProducer instance.

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).

- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.34.1 Detailed Description

Represents the level two upgrade of a ConcreteProducer entity.

This class extends ConcreteProducerUpgrade to provide specific behavior and properties for the second level upgrade of the concrete producer.

4.34.2 Constructor & Destructor Documentation

4.34.2.1 ConcreteProducerLevelTwoUpgrade() [1/2]

```
ConcreteProducerLevelTwoUpgrade::ConcreteProducerLevelTwoUpgrade (
            ConcreteProducer * concreteProd)
```

Constructs a ConcreteProducerLevelTwoUpgrade from a ConcreteProducer instance.

Parameters

concreteProd	Pointer to the ConcreteProducer to be upgraded.

4.34.2.2 ConcreteProducerLevelTwoUpgrade() [2/2]

```
ConcreteProducerLevelTwoUpgrade::ConcreteProducerLevelTwoUpgrade (
            ConcreteProducerLevelTwoUpgrade * concreteProd)
```

Copy constructor for the ConcreteProducerLevelTwoUpgrade class.

Creates a new ConcreteProducerLevelTwoUpgrade by copying an existing instance.

Parameters

concreteProd	Pointer to the ConcreteProducerLevelTwoUpgrade object to be copied.

4.34.3 Member Function Documentation

4.34.3.1 clone()

```
Entity * ConcreteProducerLevelTwoUpgrade::clone ()  [override], [virtual]
```

Creates a clone of the ConcreteProducerLevelTwoUpgrade entity.

Returns

A pointer to the cloned ConcreteProducerLevelTwoUpgrade instance.

Implements ConcreteProducerUpgrade.

4.34.3.2 getCost()

```
Cost ConcreteProducerLevelTwoUpgrade::getCost ()  [override], [virtual]
```

Gets the cost of upgrading to the next level.

Returns

The cost associated with the level two upgrade.

Implements ConcreteProducerUpgrade.

4.34.3.3 getLevel()

```
int ConcreteProducerLevelTwoUpgrade::getLevel ()  [override], [virtual]
```

Gets the upgrade level of the concrete producer.

Returns

The level of the upgrade, which is 2 for this class.

Reimplemented from Industry.

4.34.3.4 getOutput()

```
int ConcreteProducerLevelTwoUpgrade::getOutput ()  [override], [virtual]
```

Gets the output production of the upgraded concrete producer.

Returns

The production output value after applying the upgrade multiplier.

Implements ConcreteProducerUpgrade.

4.34.3.5 update()

```
void ConcreteProducerLevelTwoUpgrade::update ()  [override], [virtual]
```

Updates the state of the concrete producer.

This method invokes the update on the associated concrete producer.

Implements ConcreteProducerUpgrade.

4.34.3.6 upgrade()

```
Entity * ConcreteProducerLevelTwoUpgrade::upgrade ()  [override], [virtual]
```

Upgrades the current concrete producer to the next level.

Returns

A pointer to the upgraded entity, specifically a ConcreteProducerLevelThreeUpgrade.

Implements ConcreteProducerUpgrade.

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

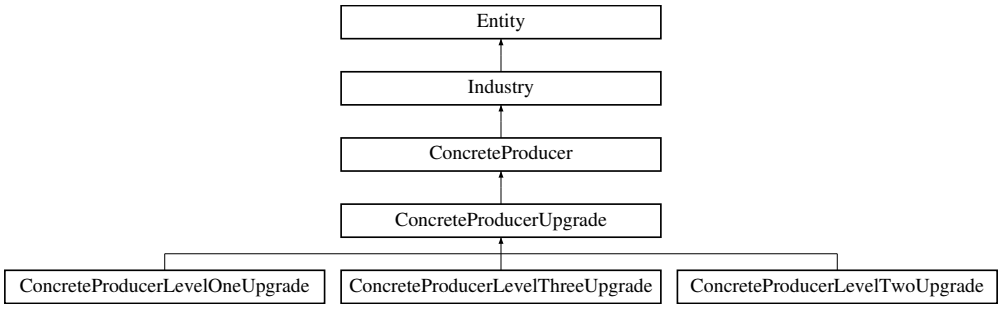
- src/entities/industry/concreteproducer/ConcreteProducerLevelTwoUpgrade.h
- src/entities/industry/concreteproducer/ConcreteProducerLevelTwoUpgrade.cpp

4.35 ConcreteProducerUpgrade Class Reference

Base class for upgrades of the ConcreteProducer entity.

```
#include <ConcreteProducerUpgrade.h>
```

Inheritance diagram for ConcreteProducerUpgrade:



Public Member Functions

- ConcreteProducerUpgrade (ConcreteProducer *concreteProd)

  Constructs a ConcreteProducerUpgrade from a ConcreteProducer instance.
- ConcreteProducerUpgrade (ConcreteProducerUpgrade *concreteProd)

  Copy constructor for the ConcreteProducerUpgrade class.
- virtual ~**ConcreteProducerUpgrade** ()

  Virtual destructor for the ConcreteProducerUpgrade class.
- virtual int getOutput ()=0

  Gets the production output of the upgraded concrete producer.
- virtual Entity * clone ()=0

  Creates a clone of the upgraded concrete producer.
- virtual void update ()=0

  Updates the state of the upgraded concrete producer.
- virtual Cost getCost ()=0

  Gets the cost of upgrading the concrete producer.
- virtual Entity * upgrade ()=0

  Upgrades the current concrete producer upgrade to the next level.

Public Member Functions inherited from ConcreteProducer

- ConcreteProducer (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a ConcreteProducer entity with specified attributes.
- ConcreteProducer (ConcreteProducer *concreteProducer)

  Copy constructor for the ConcreteProducer class.
- virtual ~**ConcreteProducer** ()

  Virtual destructor for the ConcreteProducer class.

Public Member Functions inherited from Industry

- Industry (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Industry entity with specified attributes.
- Industry (Industry *industry)

  Copy constructor for the Industry class.
- virtual ~**Industry** ()

  Virtual destructor for the Industry class.
- void setOutput (int output)

  Sets the production output of the industry.
- virtual int getLevel ()

  Gets the current level of the industry.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Protected Attributes

- ConcreteProducer * **concreteProducer**

  Pointer to the associated ConcreteProducer instance.

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).
- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.35.1 Detailed Description

Base class for upgrades of the ConcreteProducer entity.

This class serves as a base for different levels of upgrades for the ConcreteProducer, managing shared properties and methods.

4.35.2 Constructor & Destructor Documentation

4.35.2.1 ConcreteProducerUpgrade() [1/2]

```
ConcreteProducerUpgrade::ConcreteProducerUpgrade (
            ConcreteProducer * concreteProd)
```

Constructs a ConcreteProducerUpgrade from a ConcreteProducer instance.

Parameters

concreteProd	Pointer to the ConcreteProducer to be upgraded.

4.35.2.2 ConcreteProducerUpgrade() [2/2]

```
ConcreteProducerUpgrade::ConcreteProducerUpgrade (
            ConcreteProducerUpgrade * concreteProd)
```

Copy constructor for the ConcreteProducerUpgrade class.

Creates a new ConcreteProducerUpgrade by copying an existing instance.

Parameters

concreteProd	Pointer to the ConcreteProducerUpgrade object to be copied.

4.35.3 Member Function Documentation

4.35.3.1 clone()

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

Creates a clone of the upgraded concrete producer.

Returns

A pointer to the cloned ConcreteProducerUpgrade entity.

Reimplemented from ConcreteProducer.

Implemented in ConcreteProducerLevelOneUpgrade, ConcreteProducerLevelThreeUpgrade, and ConcreteProducerLevelTwoUpgrade

4.35.3.2 getCost()

```
virtual Cost ConcreteProducerUpgrade::getCost ()  [pure virtual]
```

Gets the cost of upgrading the concrete producer.

Returns

The cost associated with the upgrade.

Reimplemented from Industry.

Implemented in ConcreteProducerLevelOneUpgrade, ConcreteProducerLevelThreeUpgrade, and ConcreteProducerLevelTwoUpgrade

4.35.3.3 getOutput()

```
virtual int ConcreteProducerUpgrade::getOutput ()  [pure virtual]
```

Gets the production output of the upgraded concrete producer.

Returns

The production output value.

Reimplemented from Industry.

Implemented in ConcreteProducerLevelOneUpgrade, ConcreteProducerLevelThreeUpgrade, and ConcreteProducerLevelTwoUpgrad

4.35.3.4 update()

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

Updates the state of the upgraded concrete producer.

This method must be implemented in derived classes.

Reimplemented from ConcreteProducer.

Implemented in ConcreteProducerLevelOneUpgrade, ConcreteProducerLevelThreeUpgrade, and ConcreteProducerLevelTwoUpgrad

4.35.3.5 upgrade()

```
virtual Entity * ConcreteProducerUpgrade::upgrade ()  [pure virtual]
```

Upgrades the current concrete producer upgrade to the next level.

Returns

A pointer to the upgraded entity, or nullptr if at maximum level.

Reimplemented from ConcreteProducer.

Implemented in ConcreteProducerLevelOneUpgrade, ConcreteProducerLevelThreeUpgrade, and ConcreteProducerLevelTwoUpgrad

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

- src/entities/industry/concreteproducer/ConcreteProducerUpgrade.h
- src/entities/industry/concreteproducer/ConcreteProducerUpgrade.cpp

4.36 ConfigManager Class Reference

Singleton class to manage configurations for various entities.

```
#include <ConfigManager.h>
```

Static Public Member Functions

- static EntityConfig getEntityConfig (EntityType entityType, Size size)

  Retrieves the configuration for a specified entity type and size.
- static SatisfactionConfig getSatisfactionConfig (EntityType entityType)

  Retrieves the satisfaction configuration for a specified entity type.

4.36.1 Detailed Description

Singleton class to manage configurations for various entities.

4.36.2 Member Function Documentation

4.36.2.1 getEntityConfig()

```
static EntityConfig ConfigManager::getEntityConfig (
            EntityType entityType,
            Size size)  [inline], [static]
```

Retrieves the configuration for a specified entity type and size.

Parameters

entityType	The type of the entity.
size	The size of the entity.

Returns

The configuration for the specified entity.

4.36.2.2 getSatisfactionConfig()

```
static SatisfactionConfig ConfigManager::getSatisfactionConfig (
            EntityType entityType)  [inline], [static]
```

Retrieves the satisfaction configuration for a specified entity type.

Parameters

entityType	The type of the entity.

Returns

The satisfaction configuration for the specified entity.

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

- src/utils/ConfigManager.h

4.37 Cost Struct Reference

Represents the cost of resources for building or upgrading an entity.

`#include <Cost.h>`

Public Member Functions

- Cost (int money=0, int wood=0, int stone=0, int concrete=0)
  Constructs a Cost object with optional initial values.
- bool operator== (const Cost &other) const
  Overloaded equality operator to compare two Cost objects.

Public Attributes

- int moneyCost
- int woodCost
- int stoneCost
- int concreteCost

4.37.1 Detailed Description

Represents the cost of resources for building or upgrading an entity.

This structure holds the resource requirements in terms of money, wood, stone, and concrete. It provides a constructor for initializing costs and an overloaded equality operator for comparing costs.

4.37.2 Constructor & Destructor Documentation

4.37.2.1 Cost()

```
Cost::Cost (
            int money = 0,
            int wood = 0,
            int stone = 0,
            int concrete = 0)  [inline]
```

Constructs a Cost object with optional initial values.

Parameters

money	Initial money cost (default is 0)
wood	Initial wood cost (default is 0)
stone	Initial stone cost (default is 0)
concrete	Initial concrete cost (default is 0)

4.37.3 Member Function Documentation

4.37.3.1 operator==()

```
bool Cost::operator== (
            const Cost & other) const  [inline]
```

Overloaded equality operator to compare two Cost objects.

Parameters

other	The other Cost object to compare with

Returns

true if all cost components are equal, false otherwise

4.37.4 Member Data Documentation

4.37.4.1 concreteCost

```
int Cost::concreteCost
```

The amount of concrete required

4.37.4.2 moneyCost

```
int Cost::moneyCost
```

The amount of money required

4.37.4.3 stoneCost

```
int Cost::stoneCost
```

The amount of stone required

4.37.4.4 woodCost

```
int Cost::woodCost
```

The amount of wood required

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

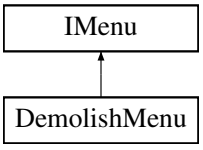
- src/utils/Cost.h

4.38 DemolishMenu Class Reference

Provides a menu interface for demolishing buildings in the game.

```
#include <DemolishMenu.h>
```

Inheritance diagram for DemolishMenu:



Public Member Functions

- DemolishMenu ()

  Constructor for DemolishMenu.
- ~**DemolishMenu** ()

  Destructor for DemolishMenu.
- void display () const override

  Displays the demolish menu.
- void handleInput () override

  Handles user input for the demolish menu.

Public Member Functions inherited from IMenu

- **IMenu** ()=default

  Default constructor for IMenu.
- IMenu (std::string heading)

  Constructor to initialize a menu with a specified heading.
- virtual ~**IMenu** ()=default

  Virtual destructor for IMenu.
- void setHeading (const std::string &heading)

  Sets the heading of the menu.

Additional Inherited Members

Protected Member Functions inherited from IMenu

- std::string repeat (const std::string &str, int times) const

  Utility function to repeat a string multiple times.
- int calculateMaxWidth (const std::string &menuHeading, const std::vector< Section > &sections) const

  Calculates the maximum width required for the menu.
- void printTopBorder (int width) const

  Prints the top border of the menu using box-drawing characters.
- void printBottomBorder (int width) const

  Prints the bottom border of the menu using box-drawing characters.
- void printSectionDivider (int width) const

  Prints a section divider in the menu using box-drawing characters.
- void printDoubleLineDivider (int width) const

  Prints a double-line divider for the main heading of the menu.
- std::string centerText (const std::string &text, int width) const

  Centers text within a specified width using space padding.
- std::string centerTextWithChar (const std::string &text, int width, const std::string &padChar) const

  Centers text within a specified width using a custom character for padding.
- void displayMenu () const

  Displays the formatted menu, including sections and options.
- void **displayChoicePrompt** () const

  Displays the choice prompt for user input.
- void displayChoiceMessagePrompt (const std::string &message) const

  Displays a custom message prompt for user input.
- void **displayInvalidChoice** () const

Displays an error message when the user makes an invalid choice.

- void displayErrorMessage (const std::string &message) const

  Displays a general error message.
- void displaySuccessMessage (const std::string &message) const

  Displays a success message in green color.
- void **displayPressEnterToContinue** () const

  Displays a message asking the user to press Enter to continue.
- void **clearScreen** () const

  Clears the terminal screen.
- std::string stripColorCodes (const std::string &input) const

  Strips ANSI color codes from a string.
- virtual void displayAvailablePositions (const std::vector< std::vector< int > > &positions) const

  Displays available positions on the city grid for an entity.

Static Protected Member Functions inherited from IMenu

- static char indexToExtendedChar (int index)

  Converts a numeric index (0-99) to a single character in an extended set.
- static std::string coordinatesToLabel (int x, int y)

  Converts x and y coordinates to a labeled string (e.g., "A, 1").

Protected Attributes inherited from IMenu

- std::vector< Section > **sections**

  List of sections contained in the menu.
- std::string **menuHeading**

  The heading/title of the menu.
- bool **hasExited**

  Flag indicating if the menu has been exited.
- CityManager **cityManager**

  Manager for city-related operations.
- bool **displayResources**

  Flag indicating whether to display resources in the menu.
- bool **isInfoMenu**

  Flag indicating whether to display option numbers.

Static Protected Attributes inherited from IMenu

- static const char * **RESET** = "\033[0m"

  ANSI color codes and styles for use in all menus.
- static const char * **BOLD_WHITE** = "\033[1;37m"
- static const char * **NORMAL_WHITE** = "\033[0;37m"
- static const char * **DARK_GRAY** = "\033[1;30m"
- static const char * **BOLD_YELLOW** = "\033[1;33m"
- static const char * **BOLD_GREEN** = "\033[1;32m"
- static const char * **BOLD_RED** = "\033[1;31m"
- static const char * **BOLD_CYAN** = "\033[1;36m"
- static const char * **BLUE** = "\033[34m"

4.38.1 Detailed Description

Provides a menu interface for demolishing buildings in the game.

The DemolishMenu class allows users to choose options for demolishing specific buildings or all buildings of a particular type. It provides functionality for confirming and processing the demolition of selected buildings.

4.38.2 Constructor & Destructor Documentation

4.38.2.1 DemolishMenu()

```
DemolishMenu::DemolishMenu ()
```

Constructor for DemolishMenu.

Constructor for DemolishMenu. Initializes the menu with options to demolish specific or all buildings of a type.

Initializes the menu with options for selecting buildings to demolish.

4.38.3 Member Function Documentation

4.38.3.1 display()

```
void DemolishMenu::display () const  [override], [virtual]
```

Displays the demolish menu.

Clears the screen and shows the available demolition options.

Implements IMenu.

4.38.3.2 handleInput()

```
void DemolishMenu::handleInput ()  [override], [virtual]
```

Handles user input for the demolish menu.

Handles user input for demolishing buildings.

Processes user choices and navigates to specific demolition operations based on input.

Implements IMenu.

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

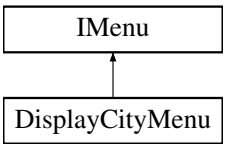
- src/menus/buildings/demolish/DemolishMenu.h
- src/menus/buildings/demolish/DemolishMenu.cpp

4.39 DisplayCityMenu Class Reference

Provides functionality to display the city and filter views by entity type.

`#include <DisplayCityMenu.h>`

Inheritance diagram for DisplayCityMenu:



Public Member Functions

- DisplayCityMenu ()

  Constructs the DisplayCityMenu.
- ~DisplayCityMenu ()

  Destructor for DisplayCityMenu.
- void display () const override

  Displays the "Display City" menu and the current city view.
- void handleInput () override

  Handles user input for the "Display City" menu.

Public Member Functions inherited from IMenu

- **IMenu** ()=default

  Default constructor for IMenu.
- IMenu (std::string heading)

  Constructor to initialize a menu with a specified heading.
- virtual ~**IMenu** ()=default

  Virtual destructor for IMenu.
- void setHeading (const std::string &heading)

  Sets the heading of the menu.

Additional Inherited Members

Protected Member Functions inherited from IMenu

- std::string repeat (const std::string &str, int times) const

  Utility function to repeat a string multiple times.
- int calculateMaxWidth (const std::string &menuHeading, const std::vector< Section > &sections) const

  Calculates the maximum width required for the menu.
- void printTopBorder (int width) const

  Prints the top border of the menu using box-drawing characters.
- void printBottomBorder (int width) const

  Prints the bottom border of the menu using box-drawing characters.
- void printSectionDivider (int width) const

Prints a section divider in the menu using box-drawing characters.

- void printDoubleLineDivider (int width) const

  Prints a double-line divider for the main heading of the menu.
- std::string centerText (const std::string &text, int width) const

  Centers text within a specified width using space padding.
- std::string centerTextWithChar (const std::string &text, int width, const std::string &padChar) const

  Centers text within a specified width using a custom character for padding.
- void displayMenu () const

  Displays the formatted menu, including sections and options.
- void **displayChoicePrompt** () const

  Displays the choice prompt for user input.
- void displayChoiceMessagePrompt (const std::string &message) const

  Displays a custom message prompt for user input.
- void **displayInvalidChoice** () const

  Displays an error message when the user makes an invalid choice.
- void displayErrorMessage (const std::string &message) const

  Displays a general error message.
- void displaySuccessMessage (const std::string &message) const

  Displays a success message in green color.
- void **displayPressEnterToContinue** () const

  Displays a message asking the user to press Enter to continue.
- void **clearScreen** () const

  Clears the terminal screen.
- std::string stripColorCodes (const std::string &input) const

  Strips ANSI color codes from a string.
- virtual void displayAvailablePositions (const std::vector< std::vector< int > > &positions) const

  Displays available positions on the city grid for an entity.

Static Protected Member Functions inherited from IMenu

- static char indexToExtendedChar (int index)

  Converts a numeric index (0-99) to a single character in an extended set.
- static std::string coordinatesToLabel (int x, int y)

  Converts x and y coordinates to a labeled string (e.g., "A, 1").

Protected Attributes inherited from IMenu

- std::vector< Section > **sections**

  List of sections contained in the menu.
- std::string **menuHeading**

  The heading/title of the menu.
- bool **hasExited**

  Flag indicating if the menu has been exited.
- CityManager **cityManager**

  Manager for city-related operations.
- bool **displayResources**

  Flag indicating whether to display resources in the menu.
- bool **isInfoMenu**

  Flag indicating whether to display option numbers.

Static Protected Attributes inherited from IMenu

- static const char * **RESET** = "\033[0m"

  ANSI color codes and styles for use in all menus.
- static const char * **BOLD_WHITE** = "\033[1;37m"
- static const char * **NORMAL_WHITE** = "\033[0;37m"
- static const char * **DARK_GRAY** = "\033[1;30m"
- static const char * **BOLD_YELLOW** = "\033[1;33m"
- static const char * **BOLD_GREEN** = "\033[1;32m"
- static const char * **BOLD_RED** = "\033[1;31m"
- static const char * **BOLD_CYAN** = "\033[1;36m"
- static const char * **BLUE** = "\033[34m"

4.39.1 Detailed Description

Provides functionality to display the city and filter views by entity type.

The DisplayCityMenu class allows players to view the entire city or filter the view to show specific types of entities such as residential buildings, economic buildings, services, utilities, industries, and transport systems.

4.39.2 Constructor & Destructor Documentation

4.39.2.1 DisplayCityMenu()

```
DisplayCityMenu::DisplayCityMenu ()
```

Constructs the DisplayCityMenu.

Constructor for DisplayCityMenu. Initializes the menu with options for different display types and a back option.

Initializes the menu with various display options and navigation.

4.39.2.2 ~DisplayCityMenu()

```
DisplayCityMenu::~DisplayCityMenu ()
```

Destructor for DisplayCityMenu.

Cleans up any resources used by the DisplayCityMenu.

4.39.3 Member Function Documentation

4.39.3.1 display()

```
void DisplayCityMenu::display () const  [override], [virtual]
```

Displays the "Display City" menu and the current city view.

Displays the menu and the city grid based on the selected display mode.

This method overrides the display method of IMenu to render the menu and show the city grid based on the selected display mode.

Implements IMenu.

4.39.3.2 handleInput()

```
void DisplayCityMenu::handleInput () [override], [virtual]
```

Handles user input for the "Display City" menu.

Handles user input in the "Display City" menu. Allows the user to select a display mode or return to the main menu.

Allows the player to choose display modes or navigate back to the main menu. Processes the user's selection and updates the current display mode accordingly.

Implements IMenu.

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

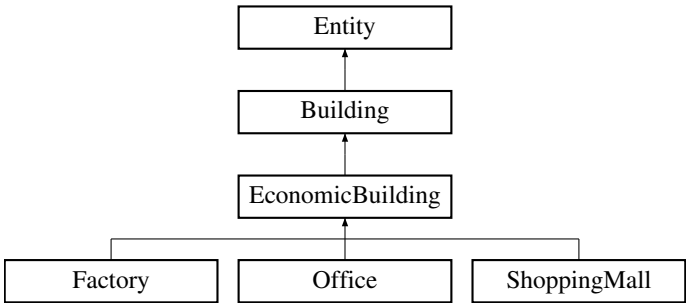
- src/menus/main/DisplayCityMenu.h
- src/menus/main/DisplayCityMenu.cpp

4.40 EconomicBuilding Class Reference

Abstract class representing economic buildings in the city builder/manager game.

```
#include <EconomicBuilding.h>
```

Inheritance diagram for EconomicBuilding:



Public Member Functions

- EconomicBuilding (EntityConfig ec, Size size, int xPos, int yPos)

  Parameterized constructor for the EconomicBuilding class.
- EconomicBuilding (EconomicBuilding ∗economic)

  Copy constructor for the EconomicBuilding class.
- virtual ∼EconomicBuilding ()

  Destructor for the EconomicBuilding class.
- virtual void update ()=0

  Updates the state of the economic building entity.
- virtual Entity ∗ clone ()=0

  Clones the economic building entity.

Public Member Functions inherited from Building

- Building (EntityConfig ec, Size size, int xPos, int yPos)

  Parameterized constructor for the Building class.
- Building (Building *building)

  Copy constructor for the Building class.
- virtual ~Building ()

  Destructor for the Building class.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

Gets the list of entities observing this entity.

- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).
- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.40.1 Detailed Description

Abstract class representing economic buildings in the city builder/manager game.

EconomicBuilding is a type of Building that generates revenue or supports the city's economy. It inherits from both the Building and Subject classes.

4.40.2 Constructor & Destructor Documentation

4.40.2.1 EconomicBuilding() [1/2]

```
EconomicBuilding::EconomicBuilding (
            EntityConfig ec,
            Size size,
            int xPos,
            int yPos)
```

Parameterized constructor for the EconomicBuilding class.

Parameters

ec	The configuration object containing general entity properties.
size	The size of the economic building entity.
xPos	The x-coordinate position of the economic building on the map.
yPos	The y-coordinate position of the economic building on the map.

Initializes a new instance of the EconomicBuilding class with specific values.

4.40.2.2 EconomicBuilding() [2/2]

```
EconomicBuilding::EconomicBuilding (
            EconomicBuilding * economic)
```

Copy constructor for the EconomicBuilding class.

Parameters

economic	A pointer to an existing EconomicBuilding object to copy from.

Creates a new EconomicBuilding instance as a copy of the provided object.

4.40.2.3 ~EconomicBuilding()

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

Destructor for the EconomicBuilding class.

Ensures proper cleanup of resources when an EconomicBuilding object is destroyed.

4.40.3 Member Function Documentation

4.40.3.1 clone()

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

Clones the economic building entity.

Returns a deep copy of the current EconomicBuilding object.

Returns

A pointer to the newly cloned EconomicBuilding entity.

Implements Building.

Implemented in Factory, Office, and ShoppingMall.

4.40.3.2 update()

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

Updates the state of the economic building entity.

A pure virtual function that must be implemented by derived classes to handle changes in the economic building's state.

Implements Building.

Implemented in Factory, Office, and ShoppingMall.

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

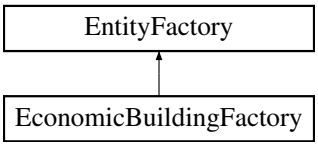
- src/entities/building/economic/EconomicBuilding.h
- src/entities/building/economic/EconomicBuilding.cpp

4.41 EconomicBuildingFactory Class Reference

Factory class for creating economic buildings, including factories, shopping malls, and offices.

```
#include <EconomicBuildingFactory.h>
```

Inheritance diagram for EconomicBuildingFactory:



Public Member Functions

- **EconomicBuildingFactory** ()

  Default constructor for EconomicBuildingFactory.
- ~**EconomicBuildingFactory** ()

  Destructor for EconomicBuildingFactory.
- virtual Entity * createEntity (EntityType type, Size size, int xPos, int yPos)

  Creates an economic building of the specified type and size at the given position.

Public Member Functions inherited from EntityFactory

- **EntityFactory** ()

  Default constructor for EntityFactory.
- virtual ~**EntityFactory** ()

  Virtual destructor for EntityFactory.

4.41.1 Detailed Description

Factory class for creating economic buildings, including factories, shopping malls, and offices.

Inherits from EntityFactory and provides methods to create different-sized economic buildings (small, medium, and large) at specified coordinates.

4.41.2 Member Function Documentation

4.41.2.1 createEntity()

```
Entity * EconomicBuildingFactory::createEntity (
            EntityType type,
            Size size,
            int xPos,
            int yPos)  [virtual]
```

Creates an economic building of the specified type and size at the given position.

Parameters

type	The type of economic building to create (e.g., Factory, ShoppingMall, Office).
size	The size of the building (small, medium, or large).
xPos	The x-coordinate of the building's position.
yPos	The y-coordinate of the building's position.

Returns

A pointer to the created Entity.

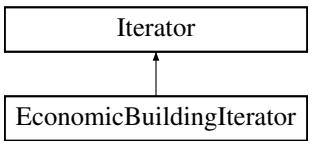
Implements EntityFactory.

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

- src/factory/building/EconomicBuildingFactory.h
- src/factory/building/EconomicBuildingFactory.cpp

4.42 EconomicBuildingIterator Class Reference

Inheritance diagram for EconomicBuildingIterator:



Public Member Functions

- **EconomicBuildingIterator** ()

  Construct a new Economic Building Iterator:: Economic Building Iterator object.
- ~**EconomicBuildingIterator** ()

  Destroy the Economic Building Iterator:: Economic Building Iterator object.
- EconomicBuildingIterator (std::vector< std::vector< Entity * > > &grid)

  Construct a new Economic Building Iterator:: Economic Building Iterator object.
- void first ()

  Sets the iterator to the first unvisited EconomicBuilding.
- void next ()

  Advances to the next unvisited EconomicBuilding.
- bool hasNext ()

  Checks if there is another unvisited EconomicBuilding.
- Entity * current ()

  Returns the current EconomicBuilding.

Public Member Functions inherited from Iterator

- **Iterator** ()

  Construct a new Iterator object, initializing row and column to zero.
- virtual ~**Iterator** ()

  Destroy the Iterator object.
- **Iterator** (std::vector< std::vector< Entity * > > &grid)
- virtual int getRow ()

  Get the current row index of the iterator.
- virtual int getCol ()

  Get the current column index of the iterator.

Additional Inherited Members

Protected Member Functions inherited from Iterator

- bool isVisited (Entity *entity)

  Check if the specified entity has been visited.
- void markVisited (Entity *entity)

  Mark the specified entity as visited.

Protected Attributes inherited from Iterator

- std::vector< std::vector< Entity * > > **grid**
- std::vector< std::vector< Entity * > >::iterator **currRow**
- std::vector< Entity * >::iterator **curr**
- int **row**
- int **col**
- std::unordered_set< Entity * > **visitedEntities**

4.42.1 Constructor & Destructor Documentation

4.42.1.1 EconomicBuildingIterator()

```
EconomicBuildingIterator::EconomicBuildingIterator (
            std::vector< std::vector< Entity * > > & grid)
```

Construct a new Economic Building Iterator:: Economic Building Iterator object.

Parameters

grid	

4.42.2 Member Function Documentation

4.42.2.1 current()

```
Entity * EconomicBuildingIterator::current ()  [virtual]
```

Returns the current EconomicBuilding.

Returns

Entity*

Implements Iterator.

4.42.2.2 first()

```
void EconomicBuildingIterator::first ()  [virtual]
```

Sets the iterator to the first unvisited EconomicBuilding.

Implements Iterator.

4.42.2.3 hasNext()

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

Checks if there is another unvisited EconomicBuilding.

Returns

true if there is another unvisited EconomicBuilding, false otherwise

Implements Iterator.

4.42.2.4 next()

```
void EconomicBuildingIterator::next () [virtual]
```

Advances to the next unvisited EconomicBuilding.

Implements Iterator.

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

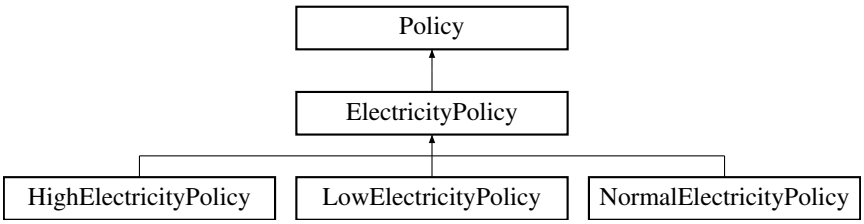
- src/iterators/building/economic/EconomicBuildingIterator.h
- src/iterators/building/economic/EconomicBuildingIterator.cpp

4.43 ElectricityPolicy Class Reference

Abstract class for ElectricityPolicy.

```
#include <ElectricityPolicy.h>
```

Inheritance diagram for ElectricityPolicy:



Public Member Functions

- ElectricityPolicy (const std::string &name, const std::string &detail)

  Constructor for ElectricityPolicy.
- virtual int calculateElectricityUsage (int electricityUsage)=0

  Pure virtual function to calculate electricity usage.
- virtual ~**ElectricityPolicy** ()

  Virtual destructor for ElectricityPolicy.

Public Member Functions inherited from Policy

- Policy (const std::string &name, const std::string &detail)

  Constructor for Policy.
- Memento * createMemento () const

  Creates a memento to store the current state of the policy.
- void setMemento (const Memento *memento)

  Sets the policy state from a memento.
- std::string getName () const

  Gets the name of the policy.
- std::string getDetail () const

  Gets the detail of the policy.

4.43.1 Detailed Description

Abstract class for ElectricityPolicy.

Defines the interface for calculating electricity usage based on different policy strategies.

4.43.2 Constructor & Destructor Documentation

4.43.2.1 ElectricityPolicy()

```
ElectricityPolicy::ElectricityPolicy (
            const std::string & name,
            const std::string & detail)  [inline]
```

Constructor for ElectricityPolicy.

Parameters

name	Name of the policy.
detail	Details describing the policy.

4.43.3 Member Function Documentation

4.43.3.1 calculateElectricityUsage()

```
virtual int ElectricityPolicy::calculateElectricityUsage (
            int electricityUsage)  [pure virtual]
```

Pure virtual function to calculate electricity usage.

Parameters

electricityUsage	Initial electricity usage.

Returns

int Modified electricity usage based on policy.

Implemented in HighElectricityPolicy, LowElectricityPolicy, and NormalElectricityPolicy.

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

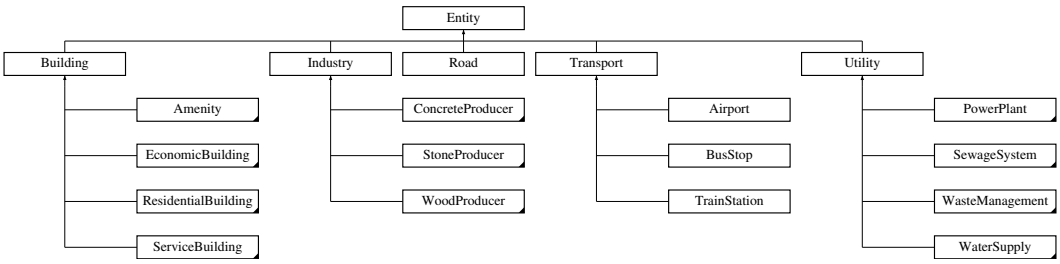
- src/policies/electricity/ElectricityPolicy.h

4.44 Entity Class Reference

Represents a game entity with properties such as position, size, and state.

`#include <Entity.h>`

Inheritance diagram for Entity:



Public Member Functions

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity ∗entity)

  Copy constructor for the Entity class.
- virtual ∼**Entity** ()

  Virtual destructor for the Entity class.
- virtual void update ()=0

  Updates the entity's state. Needs to be implemented in derived classes.
- bool isWithinEffectRadius (Entity ∗entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- virtual Entity ∗ clone ()=0

  Clones the entity. Needs to be implemented in derived classes.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.

- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Protected Attributes

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).
- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

Electricity consumption of the entity.

- float **waterConsumption**

  Water consumption of the entity.

- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.44.1 Detailed Description

Represents a game entity with properties such as position, size, and state.

The Entity class is responsible for managing the state of the entity, including its position, dimensions, and resource consumption.

4.44.2 Constructor & Destructor Documentation

4.44.2.1 Entity() [1/2]

```
Entity::Entity (
            EntityConfig ec,
            Size size,
            int xPos,
            int yPos)
```

Constructs an Entity with specified attributes.

Parameters

ec	Entity configuration containing resource consumption and effects.
size	Size of the entity.
xPos	X-coordinate position of the entity.
yPos	Y-coordinate position of the entity.

4.44.2.2 Entity() [2/2]

```
Entity::Entity (
            Entity * entity)
```

Copy constructor for the Entity class.

Creates a new Entity by copying the attributes of an existing Entity. This performs a deep copy of all properties, ensuring that the new entity is independent of the original.

Parameters

entity	Pointer to the Entity object to be copied.

4.44.3 Member Function Documentation

4.44.3.1 clone()

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

Clones the entity. Needs to be implemented in derived classes.

Returns

A pointer to the cloned entity, or nullptr if not implemented.

Implemented in Airport, Amenity, Apartment, Building, BusStop, ConcreteProducer, ConcreteProducerLevelOneUpgrade, ConcreteProducerLevelThreeUpgrade, ConcreteProducerLevelTwoUpgrade, ConcreteProducerUpgrade, EconomicBuilding, Factory, Hospital, House, Industry, Monument, Office, Park, PoliceStation, PowerPlant, PowerPlantLevelOneUpgrade, PowerPlantLevelThreeUpgrade, PowerPlantLevelTwoUpgrade, PowerPlantUpgrade, ResidentialBuilding, Road, School, ServiceBuilding, SewageSystem, SewageSystemLevelOneUpgrade, SewageSystemLevelThreeUpgrade, SewageSystemLevelTwoUpgrade, SewageSystemUpgrade, ShoppingMall, StoneProducer, StoneProducerLevelOneUpgrade, StoneProducerLevelThreeUpgrade, StoneProducerLevelTwoUpgrade, StoneProducerUpgrade, Theater, TrainStation, Transport, Utility, WasteManagement, WasteManagementLevelOneUpgrade, WasteManagementLevelThreeUpgrade, WasteManagementLevelTwoUpgrade, WasteManagementUpgrade, WaterSupply, WaterSupplyLevelOneUpgrade, WaterSupplyLevelThreeUpgrade, WaterSupplyLevelTwoUpgrade, WaterSupplyUpgrade, WoodProducer, WoodProducerLevelOneUpgrade, WoodProducerLevelThreeUpgrade, WoodProducerLevelTwoUpgrade, and WoodProducerUpgrade.

4.44.3.2 getElectricityConsumption()

```
float Entity::getElectricityConsumption ()
```

Gets the electricity consumption of the entity.

Returns

Electricity consumption value.

4.44.3.3 getHeight()

```
int Entity::getHeight ()
```

Gets the height of the entity.

Returns

The height value.

4.44.3.4 getObservers()

```
const std::vector< Entity * > Entity::getObservers ()
```

Gets the list of entities observing this entity.

Returns

A constant reference to the vector of observer entities.

4.44.3.5 getRevenue()

```
int Entity::getRevenue ()
```

Gets the revenue generated by the entity.

Returns

The revenue value.

4.44.3.6 getSize()

```
Size Entity::getSize () const  [inline]
```

Gets the size of this entity.

Returns

Size The size of the entity.

4.44.3.7 getSymbol()

```
std::string Entity::getSymbol ()
```

Gets the symbol of the entity.

Returns

The symbol representing the entity.

4.44.3.8 getType()

```
EntityType Entity::getType () const  [inline]
```

Gets the entity type of this entity.

Returns

EntityType The entity type.

4.44.3.9 getWaterConsumption()

```
float Entity::getWaterConsumption ()
```

Gets the water consumption of the entity.

Returns

Water consumption value.

4.44.3.10 getWidth()

```
int Entity::getWidth ()
```

Gets the width of the entity.

Returns

The width value.

4.44.3.11 getXPosition()

```
int Entity::getXPosition ()
```

Gets the X-coordinate position of the entity.

Returns

The X-coordinate position.

4.44.3.12 getYPosition()

```
int Entity::getYPosition ()
```

Gets the Y-coordinate position of the entity.

Returns

The Y-coordinate position.

4.44.3.13 isBuilt()

```
bool Entity::isBuilt ()
```

Checks if the entity is built (i.e., not under construction).

Returns

True if the entity is built, false otherwise.

4.44.3.14 isWithinEffectRadius()

```
bool Entity::isWithinEffectRadius (
            Entity * entity)
```

Checks if another entity is within the effect radius of this entity.

Parameters

entity	Pointer to the entity to check.

Returns

True if the entity is within the effect radius, false otherwise.

4.44.3.15 setSymbol()

```
void Entity::setSymbol (
            std::string symbol)
```

Sets the symbol of the entity.

Parameters

symbol	The new symbol for the entity.

4.44.3.16 setXPosition()

```
void Entity::setXPosition (
            int x)
```

Sets the X-coordinate position of the entity.

Parameters

x	The new X-coordinate position.

4.44.3.17 setYPosition()

```
void Entity::setYPosition (
            int y)
```

Sets the Y-coordinate position of the entity.

Parameters

y	The new Y-coordinate position.

4.44.3.18 subscribe()

```
void Entity::subscribe (
            Entity * entity)
```

Subscribes this entity as an observer of another entity.

Parameters

entity	The entity to subscribe to.

4.44.3.19 unsubscribe()

```
void Entity::unsubscribe (
            Entity * entity)
```

Unsubscribes this entity from observing another entity.

Parameters

entity	The entity to unsubscribe from.

4.44.3.20 update()

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

Updates the entity's state. Needs to be implemented in derived classes.

Implemented in Airport, Amenity, Building, BusStop, ConcreteProducer, ConcreteProducerLevelOneUpgrade, ConcreteProducerLevelThreeUpgrade, ConcreteProducerLevelTwoUpgrade, ConcreteProducerUpgrade, EconomicBuilding, Factory, Hospital, Industry, Monument, Office, Park, PoliceStation, PowerPlant, PowerPlantLevelOneUpgrade, PowerPlantLevelThreeUpgrade, PowerPlantLevelTwoUpgrade, PowerPlantUpgrade, ResidentialBuilding, Road, School, ServiceBuilding, SewageSystem, SewageSystemLevelOneUpgrade, SewageSystemLevelThreeUpgrade, SewageSystemLevelTwoUpgrade, SewageSystemUpgrade, ShoppingMall, StoneProducer, StoneProducerLevelOneUpgrade, StoneProducerLevelThreeUpgrade, StoneProducerLevelTwoUpgrade, StoneProducerUpgrade, Theater, TrainStation, Transport, Utility, WasteManagement, WasteManagementLevelOneUpgrade, WasteManagementLevelThreeUpgrade, WasteManagementLevelTwoUpgrade, WasteManagementUpgrade, WaterSupply, WaterSupplyLevelOneUpgrade, WaterSupplyLevelThreeUpgrade, WaterSupplyLevelTwoUpgrade, WaterSupplyUpgrade, WoodProducer, WoodProducerLevelOneUpgrade, WoodProducerLevelThreeUpgrade, WoodProducerLevelTwoUpgrade, and WoodProducerUpgrade.

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

- src/entities/base/Entity.h
- src/entities/base/Entity.cpp

4.45 EntityConfig Struct Reference

Configuration struct for an entity.

```
#include <EntityConfig.h>
```

Public Member Functions

- **EntityConfig** ()

  Default constructor initializing default values.
- EntityConfig (const Cost &cost, int electricity, int water, const std::string &symbol, int radius, int localEffect, int globalEffect, int width, int height, int revenue, int buildTime, EntityType entityType, Size size)

  Constructor initializing all properties.

Public Attributes

- Cost **cost**

  Cost of the entity.
- int **electricityConsumption**

  Electricity consumption level.
- int **waterConsumption**

  Water consumption level.
- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of the entity's effect.
- int **localEffectStrength**

  Strength of the local effect.
- int **globalEffectStrength**

  Strength of the global effect.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **revenue**

  Revenue generated by the entity.
- int **buildTime**

  Time required to build the entity.
- EntityType **entityType**

  Type of the entity.
- Size **size**

  Size category of the entity.

4.45.1 Detailed Description

Configuration struct for an entity.

4.45.2 Constructor & Destructor Documentation

4.45.2.1 EntityConfig()

```
EntityConfig::EntityConfig (
            const Cost & cost,
            int electricity,
            int water,
            const std::string & symbol,
            int radius,
            int localEffect,
            int globalEffect,
            int width,
            int height,
            int revenue,
            int buildTime,
            EntityType entityType,
            Size size)  [inline]
```

Constructor initializing all properties.

Parameters

cost	Cost of the entity.
electricity	Electricity consumption.
water	Water consumption.
symbol	Symbol representing the entity.
radius	Radius of the entity's effect.
localEffect	Local effect strength.
globalEffect	Global effect strength.
width	Width of the entity.
height	Height of the entity.
revenue	Revenue generated by the entity.
buildTime	Time required to build the entity.
entityType	Type of the entity.
size	Size category of the entity.

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

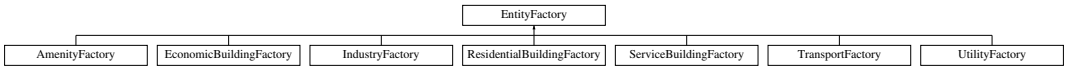
- src/utils/EntityConfig.h

4.46 EntityFactory Class Reference

Abstract factory class for creating entities of various types and sizes.

`#include <EntityFactory.h>`

Inheritance diagram for EntityFactory:



Public Member Functions

- **EntityFactory** ()

  Default constructor for EntityFactory.
- virtual ~**EntityFactory** ()

  Virtual destructor for EntityFactory.
- virtual Entity * createEntity (EntityType type, Size size, int xPos, int yPos)=0

  Pure virtual function to create an entity of a specified type and size at a given position.

4.46.1 Detailed Description

Abstract factory class for creating entities of various types and sizes.

The EntityFactory class serves as a base class to define the interface for creating different types and sizes of entities, including small, medium, and large variants.

Note

EntityFactory is an abstract class and cannot be instantiated directly. It requires subclassing, where the subclass provides concrete implementations for the `createEntity` function.

4.46.2 Member Function Documentation

4.46.2.1 createEntity()

```
virtual Entity * EntityFactory::createEntity (
            EntityType type,
            Size size,
            int xPos,
            int yPos)  [pure virtual]
```

Pure virtual function to create an entity of a specified type and size at a given position.

Derived classes must implement this function to create a specific type of entity. This allows for flexible creation of entities, where the exact class of entity created can vary based on the type and size.

Parameters

type	The type of entity to create (e.g., Residential, Industrial).
size	The size of the entity to create (small, medium, or large).
xPos	The x-coordinate for the entity's position.
yPos	The y-coordinate for the entity's position.

Returns

A pointer to the created Entity.

Implemented in AmenityFactory, EconomicBuildingFactory, IndustryFactory, ResidentialBuildingFactory, ServiceBuildingFactory, TransportFactory, and UtilityFactory.

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

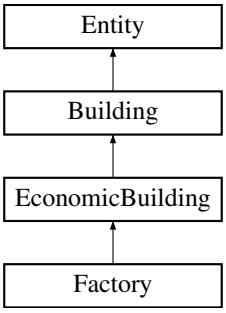
- src/factory/base/EntityFactory.h
- src/factory/base/EntityFactory.cpp

4.47 Factory Class Reference

Concrete class representing a factory in the city builder/manager game.

`#include <Factory.h>`

Inheritance diagram for Factory:



Public Member Functions

- Factory (EntityConfig ec, Size size, int xPos, int yPos)
  Parameterized constructor for the Factory class.
- Factory (Factory *factory)
  Copy constructor for the Factory class.
- ~Factory ()
  Destructor for the Factory class.
- void update ()
  Updates the state of the factory entity.
- Entity * clone ()
  Clones the factory entity.

Public Member Functions inherited from EconomicBuilding

- EconomicBuilding (EntityConfig ec, Size size, int xPos, int yPos)
  Parameterized constructor for the EconomicBuilding class.
- EconomicBuilding (EconomicBuilding *economic)
  Copy constructor for the EconomicBuilding class.
- virtual ~EconomicBuilding ()
  Destructor for the EconomicBuilding class.

Public Member Functions inherited from Building

- Building (EntityConfig ec, Size size, int xPos, int yPos)
  Parameterized constructor for the Building class.
- Building (Building *building)
  Copy constructor for the Building class.
- virtual ~Building ()
  Destructor for the Building class.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).
- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.47.1 Detailed Description

Concrete class representing a factory in the city builder/manager game.

Factory is a type of EconomicBuilding that produces goods and supports industrial activities.

4.47.2 Constructor & Destructor Documentation

4.47.2.1 Factory() [1/2]

```
Factory::Factory (
            EntityConfig ec,
            Size size,
            int xPos,
            int yPos)
```

Parameterized constructor for the Factory class.

Parameters

ec	The configuration object containing general entity properties.
size	The size of the factory entity.
xPos	The x-coordinate position of the factory on the map.
yPos	The y-coordinate position of the factory on the map.

Initializes a new instance of the Factory class with specific values.

4.47.2.2 Factory() [2/2]

```
Factory::Factory (
            Factory * factory)
```

Copy constructor for the Factory class.

Parameters

factory	A pointer to an existing Factory object to copy from.

Creates a new Factory instance as a copy of the provided object.

4.47.2.3 ~Factory()

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

Destructor for the Factory class.

Ensures proper cleanup of resources when a Factory object is destroyed.

4.47.3 Member Function Documentation

4.47.3.1 clone()

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

Clones the factory entity.

Returns a deep copy of the current Factory object.

Returns

A pointer to the newly cloned Factory entity.

Implements EconomicBuilding.

4.47.3.2 update()

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

Updates the state of the factory entity.

This function handles changes in the factory's state.

Implements EconomicBuilding.

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

- src/entities/building/economic/Factory.h
- src/entities/building/economic/Factory.cpp

4.48 GovernmentManager Class Reference

Manages government policies and taxation within the city.

```
#include <GovernmentManager.h>
```

Public Member Functions

- **GovernmentManager** ()
  Constructor for GovernmentManager.
- ~**GovernmentManager** ()
  Destructor for GovernmentManager.
- void setResidentialTaxRate (float rate)
  Sets the residential tax rate in the city.
- void setEconomicTaxRate (float rate)
  Sets the economic tax rate in the city.
- int getResidentialTax ()
  Gets the total residential tax collected.
- int getEconomicTax ()
  Gets the total economic tax collected.
- int getResidentialTaxRate ()
  Gets the current residential tax rate.
- int getEconomicTaxRate ()
  Gets the current economic tax rate.
- void enactWaterUsagePolicy (PolicyType policy)
  Enacts a specified water usage policy in the city.
- void enactElectricityPolicy (PolicyType policy)
  Enacts a specified electricity policy in the city.
- std::vector< Memento * > getAllPastPolicies ()
  Retrieves all past policies stored by the caretaker.

4.48.1 Detailed Description

Manages government policies and taxation within the city.

The GovernmentManager class interacts with the City and Visitor classes to manage residential and economic taxation and to enact water and electricity usage policies.

4.48.2 Member Function Documentation

4.48.2.1 enactElectricityPolicy()

```
void GovernmentManager::enactElectricityPolicy (
            PolicyType policy)
```

Enacts a specified electricity policy in the city.

Parameters

policy	The type of electricity policy to enact.

4.48.2.2 enactWaterUsagePolicy()

```
void GovernmentManager::enactWaterUsagePolicy (
            PolicyType policy)
```

Enacts a specified water usage policy in the city.

Parameters

policy	The type of water policy to enact.

4.48.2.3 getAllPastPolicies()

```
std::vector< Memento * > GovernmentManager::getAllPastPolicies ()
```

Retrieves all past policies stored by the caretaker.

Returns

Vector of pointers to Memento objects representing past policies.

4.48.2.4 getEconomicTax()

```
int GovernmentManager::getEconomicTax ()
```

Gets the total economic tax collected.

Returns

The total economic tax amount.

4.48.2.5 getEconomicTaxRate()

```
int GovernmentManager::getEconomicTaxRate ()
```

Gets the current economic tax rate.

Returns

The economic tax rate.

4.48.2.6 getResidentialTax()

```
int GovernmentManager::getResidentialTax ()
```

Gets the total residential tax collected.

Returns

The total residential tax amount.

4.48.2.7 getResidentialTaxRate()

```
int GovernmentManager::getResidentialTaxRate ()
```

Gets the current residential tax rate.

Returns

The residential tax rate.

4.48.2.8 setEconomicTaxRate()

```
void GovernmentManager::setEconomicTaxRate (
            float rate)
```

Sets the economic tax rate in the city.

Parameters

rate	The economic tax rate.

4.48.2.9 setResidentialTaxRate()

```
void GovernmentManager::setResidentialTaxRate (
            float rate)
```

Sets the residential tax rate in the city.

Parameters

rate	The residential tax rate.

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

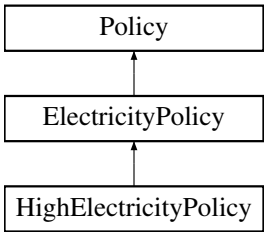
- src/managers/GovernmentManager.h
- src/managers/GovernmentManager.cpp

4.49 HighElectricityPolicy Class Reference

Concrete strategy for high electricity policy.

`#include <HighElectricityPolicy.h>`

Inheritance diagram for HighElectricityPolicy:



Public Member Functions

- int calculateElectricityUsage (int electricityUsage) override

  Overrides calculateElectricityUsage to increase electricity usage.
- **HighElectricityPolicy** ()

  Constructor for HighElectricityPolicy. Initializes the policy with specific name and detail.

Public Member Functions inherited from ElectricityPolicy

- ElectricityPolicy (const std::string &name, const std::string &detail)

  Constructor for ElectricityPolicy.
- virtual ~**ElectricityPolicy** ()

  Virtual destructor for ElectricityPolicy.

Public Member Functions inherited from Policy

- Policy (const std::string &name, const std::string &detail)

  Constructor for Policy.
- Memento * createMemento () const

  Creates a memento to store the current state of the policy.
- void setMemento (const Memento *memento)

  Sets the policy state from a memento.
- std::string getName () const

  Gets the name of the policy.
- std::string getDetail () const

  Gets the detail of the policy.

4.49.1 Detailed Description

Concrete strategy for high electricity policy.

Increases electricity usage by applying a high usage factor.

4.49.2 Member Function Documentation

4.49.2.1 calculateElectricityUsage()

```
int HighElectricityPolicy::calculateElectricityUsage (
            int electricityUsage)  [override], [virtual]
```

Overrides calculateElectricityUsage to increase electricity usage.

Implementation of HighElectricityPolicy to increase electricity usage by 25%.

Parameters

electricityUsage	Initial electricity usage.

Returns

int Increased electricity usage (e.g., 125% of the original).

Parameters

electricityUsage	Initial electricity usage.

Returns

int Increased electricity usage.

Implements ElectricityPolicy.

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

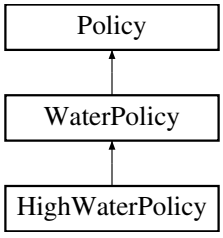
- src/policies/electricity/HighElectricityPolicy.h
- src/policies/electricity/HighElectricityPolicy.cpp

4.50 HighWaterPolicy Class Reference

Concrete strategy for high water policy.

```
#include <HighWaterPolicy.h>
```

Inheritance diagram for HighWaterPolicy:



Public Member Functions

- int calculateWaterUsage (int waterUsage) override

  Overrides calculateWaterUsage to increase water usage.
- **HighWaterPolicy** ()

  Constructor for HighWaterPolicy. Initializes the policy with specific name and detail.

Public Member Functions inherited from WaterPolicy

- WaterPolicy (const std::string &name, const std::string &detail)

  Constructor for WaterPolicy.
- virtual ~**WaterPolicy** ()

  Virtual destructor for WaterPolicy.

Public Member Functions inherited from Policy

- Policy (const std::string &name, const std::string &detail)

  Constructor for Policy.
- Memento * createMemento () const

  Creates a memento to store the current state of the policy.
- void setMemento (const Memento *memento)

  Sets the policy state from a memento.
- std::string getName () const

  Gets the name of the policy.
- std::string getDetail () const

  Gets the detail of the policy.

4.50.1 Detailed Description

Concrete strategy for high water policy.

Increases water usage by applying a high usage factor.

4.50.2 Member Function Documentation

4.50.2.1 calculateWaterUsage()

```
int HighWaterPolicy::calculateWaterUsage (
            int waterUsage)  [override], [virtual]
```

Overrides calculateWaterUsage to increase water usage.

Implementation of HighWaterPolicy to increase water usage by 20%.

Parameters

waterUsage	Initial water usage.

Returns

int Increased water usage (e.g., 120% of the original).

Parameters

waterUsage	Initial water usage.

Returns

int Increased water usage.

Implements WaterPolicy.

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

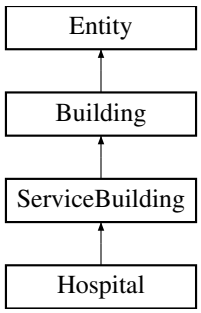
- src/policies/water/HighWaterPolicy.h
- src/policies/water/HighWaterPolicy.cpp

4.51 Hospital Class Reference

Class representing a hospital in the city.

`#include <Hospital.h>`

Inheritance diagram for Hospital:



Public Member Functions

- Hospital (EntityConfig ec, Size size, int xPos, int yPos)

  Parameterized constructor for the Hospital class.
- Hospital (Hospital *hospital)

  Copy constructor for the Hospital class.
- ~Hospital ()

  Destructor for the Hospital class.
- void update ()

  Updates the state of the hospital entity.
- Entity * clone ()

  Clones the hospital entity.

Public Member Functions inherited from ServiceBuilding

- ServiceBuilding (EntityConfig ec, Size size, int xPos, int yPos)

  Parameterized constructor for the ServiceBuilding class.
- ServiceBuilding (ServiceBuilding ∗service)

  Copy constructor for the ServiceBuilding class.
- virtual ∼ServiceBuilding ()

  Destructor for the ServiceBuilding class.

Public Member Functions inherited from Building

- Building (EntityConfig ec, Size size, int xPos, int yPos)

  Parameterized constructor for the Building class.
- Building (Building ∗building)

  Copy constructor for the Building class.
- virtual ∼Building ()

  Destructor for the Building class.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity ∗entity)

  Copy constructor for the Entity class.
- virtual ∼**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity ∗entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

Subscribes the entity to all residential entities within its effect radius.

- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).
- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.

- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.51.1 Detailed Description

Class representing a hospital in the city.

The Hospital class provides healthcare services to the population. It inherits from the ServiceBuilding class.

4.51.2 Constructor & Destructor Documentation

4.51.2.1 Hospital() [1/2]

```
Hospital::Hospital (
            EntityConfig ec,
            Size size,
            int xPos,
            int yPos)
```

Parameterized constructor for the Hospital class.

Parameters

ec	Entity configuration for initializing the hospital.
size	The size of the hospital.
xPos	The x-coordinate of the hospital's location.
yPos	The y-coordinate of the hospital's location.

4.51.2.2 Hospital() [2/2]

```
Hospital::Hospital (
            Hospital * hospital)
```

Copy constructor for the Hospital class.

Parameters

hospital	A pointer to an existing Hospital object to copy from.

4.51.2.3 ~Hospital()

```
Hospital::~Hospital ()
```

Destructor for the Hospital class.

Cleans up resources used by the Hospital object.

4.51.3 Member Function Documentation

4.51.3.1 clone()

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

Clones the hospital entity.

Returns

A pointer to a deep copy of the Hospital object.

Implements ServiceBuilding.

4.51.3.2 update()

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

Updates the state of the hospital entity.

Handles changes in the hospital's state.

Implements ServiceBuilding.

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

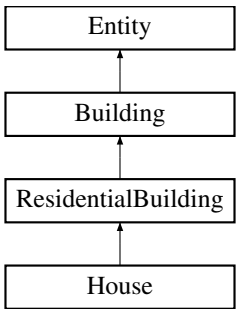
- src/entities/building/service/Hospital.h
- src/entities/building/service/Hospital.cpp

4.52 House Class Reference

Represents a house building within the game.

```
#include <House.h>
```

Inheritance diagram for House:



Public Member Functions

- House (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a House with specified attributes.
- House (House *entity)

  Copy constructor for the House class.
- virtual ~**House** ()

  Destructor for the House class.
- Entity * clone ()

  Creates a clone of the house.

Public Member Functions inherited from ResidentialBuilding

- ResidentialBuilding (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a ResidentialBuilding with specified attributes.
- ResidentialBuilding (ResidentialBuilding *entity)

  Copy constructor for the ResidentialBuilding class.
- virtual ~**ResidentialBuilding** ()

  Destructor for the ResidentialBuilding class.
- void update ()

  Updates the residential building's state.
- void **reset** ()

  Resets the satisfaction factors for the building.
- void **calculateSatisfaction** ()

  Calculates the satisfaction level based on nearby entities.
- float getSatisfaction ()

  Gets the satisfaction level of the building.
- void updateAirport (Entity *entity)

  Updates the effect of a nearby airport.
- void updateBusStop (Entity *entity)

  Updates the effect of a nearby bus stop.
- void updateTrainStation (Entity *entity)

  Updates the effect of a nearby train station.
- void updateFactory (Entity *entity)

  Updates the effect of a nearby factory.
- void updateShoppingMall (Entity *entity)

  Updates the effect of a nearby shopping mall.
- void updateOffice (Entity *entity)

  Updates the effect of a nearby office.
- void updateHospital (Entity *entity)

  Updates the effect of a nearby hospital.
- void updatePoliceStation (Entity *entity)

  Updates the effect of a nearby police station.
- void updateSchool (Entity *entity)

  Updates the effect of a nearby school.
- void updateAmenity (Entity *entity)

  Updates the effect of a nearby amenity.
- void updateUtility (Entity *entity)

  Updates the effect of a nearby utility.
- void updateIndustry (Entity *entity)

Updates the effect of a nearby industry.

- int getCapacity ()

  Gets the capacity of the residential building.
- void setCapacity (int capacity)

  Sets the capacity of the residential building.

Public Member Functions inherited from Building

- Building (EntityConfig ec, Size size, int xPos, int yPos)

  Parameterized constructor for the Building class.
- Building (Building *building)

  Copy constructor for the Building class.
- virtual ~Building ()

  Destructor for the Building class.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.

- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).
- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.52.1 Detailed Description

Represents a house building within the game.

The House class is a type of ResidentialBuilding, specifically representing standalone housing. It provides constructors, a destructor, and an implementation of the clone function.

4.52.2 Constructor & Destructor Documentation

4.52.2.1 House() [1/2]

```
House::House (
            EntityConfig ec,
            Size size,
            int xPos,
            int yPos)
```

Constructs a House with specified attributes.

Parameters

ec	Configuration containing resource consumption and other properties.
size	Size of the house.
xPos	X-coordinate position of the house.
yPos	Y-coordinate position of the house.

4.52.2.2 House() [2/2]

```
House::House (
            House * entity)
```

Copy constructor for the House class.

Creates a new House by copying the attributes of an existing House.

Parameters

entity	Pointer to the House object to be copied.

4.52.3 Member Function Documentation

4.52.3.1 clone()

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

Creates a clone of the house.

Returns

A pointer to the cloned House.

Implements ResidentialBuilding.

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

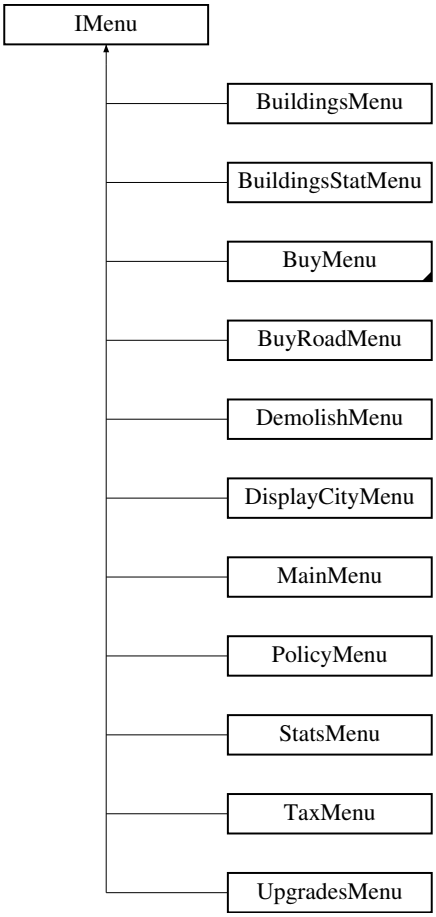
- src/entities/building/residential/House.h
- src/entities/building/residential/House.cpp

4.53 IMenu Class Reference

Abstract base class for creating menus.

`#include <IMenu.h>`

Inheritance diagram for IMenu:



Public Member Functions

- **IMenu** ()=default

  Default constructor for IMenu.
- IMenu (std::string heading)

  Constructor to initialize a menu with a specified heading.
- virtual ~**IMenu** ()=default

  Virtual destructor for IMenu.
- virtual void display () const =0

  Pure virtual function to display the menu.
- virtual void handleInput ()=0

  Pure virtual function to handle user input in the menu.
- void setHeading (const std::string &heading)

  Sets the heading of the menu.

Protected Member Functions

- std::string repeat (const std::string &str, int times) const

  Utility function to repeat a string multiple times.
- int calculateMaxWidth (const std::string &menuHeading, const std::vector< Section > &sections) const

  Calculates the maximum width required for the menu.
- void printTopBorder (int width) const

  Prints the top border of the menu using box-drawing characters.
- void printBottomBorder (int width) const

  Prints the bottom border of the menu using box-drawing characters.
- void printSectionDivider (int width) const

  Prints a section divider in the menu using box-drawing characters.
- void printDoubleLineDivider (int width) const

  Prints a double-line divider for the main heading of the menu.
- std::string centerText (const std::string &text, int width) const

  Centers text within a specified width using space padding.
- std::string centerTextWithChar (const std::string &text, int width, const std::string &padChar) const

  Centers text within a specified width using a custom character for padding.
- void displayMenu () const

  Displays the formatted menu, including sections and options.
- void **displayChoicePrompt** () const

  Displays the choice prompt for user input.
- void displayChoiceMessagePrompt (const std::string &message) const

  Displays a custom message prompt for user input.
- void **displayInvalidChoice** () const

  Displays an error message when the user makes an invalid choice.
- void displayErrorMessage (const std::string &message) const

  Displays a general error message.
- void displaySuccessMessage (const std::string &message) const

  Displays a success message in green color.
- void **displayPressEnterToContinue** () const

  Displays a message asking the user to press Enter to continue.
- void **clearScreen** () const

  Clears the terminal screen.
- std::string stripColorCodes (const std::string &input) const

  Strips ANSI color codes from a string.
- virtual void displayAvailablePositions (const std::vector< std::vector< int > > &positions) const

  Displays available positions on the city grid for an entity.

Static Protected Member Functions

- static char indexToExtendedChar (int index)

  Converts a numeric index (0-99) to a single character in an extended set.
- static std::string coordinatesToLabel (int x, int y)

  Converts x and y coordinates to a labeled string (e.g., "A, 1").

Protected Attributes

- std::vector< Section > **sections**

  List of sections contained in the menu.
- std::string **menuHeading**

  The heading/title of the menu.
- bool **hasExited**

  Flag indicating if the menu has been exited.
- CityManager **cityManager**

  Manager for city-related operations.
- bool **displayResources**

  Flag indicating whether to display resources in the menu.
- bool **isInfoMenu**

  Flag indicating whether to display option numbers.

Static Protected Attributes

- static const char * **RESET** = "\033[0m"

  ANSI color codes and styles for use in all menus.
- static const char * **BOLD_WHITE** = "\033[1;37m"
- static const char * **NORMAL_WHITE** = "\033[0;37m"
- static const char * **DARK_GRAY** = "\033[1;30m"
- static const char * **BOLD_YELLOW** = "\033[1;33m"
- static const char * **BOLD_GREEN** = "\033[1;32m"
- static const char * **BOLD_RED** = "\033[1;31m"
- static const char * **BOLD_CYAN** = "\033[1;36m"
- static const char * **BLUE** = "\033[34m"

4.53.1 Detailed Description

Abstract base class for creating menus.

This class provides the common functionality and interface for all menus. Derived classes must implement the display and input handling methods.

4.53.2 Constructor & Destructor Documentation

4.53.2.1 IMenu()

```
IMenu::IMenu (
            std::string heading)
```

Constructor to initialize a menu with a specified heading.

Constructs a menu with the specified heading.

Parameters

heading	The heading/title of the menu.
heading	The heading of the menu.

4.53.3 Member Function Documentation

4.53.3.1 calculateMaxWidth()

```
int IMenu::calculateMaxWidth (
            const std::string & menuHeading,
            const std::vector< Section > & sections) const  [protected]
```

Calculates the maximum width required for the menu.

Calculates the maximum width of the menu based on its heading and sections.

This function ensures the menu is wide enough to fit the heading and options.

Parameters

menuHeading	The heading of the menu.
sections	The sections of the menu.

Returns

The maximum width needed to display the menu.

This function calculates the maximum width of the menu by comparing the heading and the text of the options in the sections to ensure the menu is wide enough to fit all content.

Parameters

menuHeading	The heading of the menu.
sections	The sections of the menu.

Returns

The maximum width of the menu.

4.53.3.2 centerText()

```
std::string IMenu::centerText (
            const std::string & text,
            int width) const  [protected]
```

Centers text within a specified width using space padding.

Centers text with space padding on both sides.

Parameters

text	The text to be centered.
width	The total width to center the text within.

Returns

A string containing the centered text with space padding.

Parameters

text	The text to be centered.
width	The total width to center the text within.

Returns

A string containing the centered text.

4.53.3.3 centerTextWithChar()

```
std::string IMenu::centerTextWithChar (
            const std::string & text,
            int width,
            const std::string & padChar) const  [protected]
```

Centers text within a specified width using a custom character for padding.

Centers text with custom character padding on both sides.

Parameters

text	The text to be centered.
width	The total width to center the text within.
padChar	The character used for padding.

Returns

A string containing the centered text with custom character padding.

Parameters

text	The text to be centered.
width	The total width to center the text within.
padChar	The character used to pad the text.

Returns

A string containing the centered text with padding.

4.53.3.4 coordinatesToLabel()

```
std::string IMenu::coordinatesToLabel (
            int x,
            int y)  [static], [protected]
```

Converts x and y coordinates to a labeled string (e.g., "A, 1").

Parameters

x	The x-coordinate.
y	The y-coordinate.

Returns

A formatted string representing the labeled coordinates.

4.53.3.5 display()

```
virtual void IMenu::display () const  [pure virtual]
```

Pure virtual function to display the menu.

Must be implemented by derived classes to handle the rendering of the menu.

Implemented in BuildingsMenu, BuildingsStatMenu, BuyMenu, BuyRoadMenu, DemolishMenu, DisplayCityMenu, MainMenu, PolicyMenu, StatsMenu, TaxMenu, and UpgradesMenu.

4.53.3.6 displayAvailablePositions()

```
void IMenu::displayAvailablePositions (
            const std::vector< std::vector< int > > & positions) const  [protected], [virtual]
```

Displays available positions on the city grid for an entity.

Marks positions based on availability for a given type and size.

Parameters

positions	A vector of available positions on the grid.

Reimplemented in BuyMenu.

4.53.3.7 displayChoiceMessagePrompt()

```
void IMenu::displayChoiceMessagePrompt (
            const std::string & message) const  [protected]
```

Displays a custom message prompt for user input.

Displays a message prompt with a custom message.

Parameters

message	The custom message to display.
message	The message to display.

4.53.3.8 displayErrorMessage()

```
void IMenu::displayErrorMessage (
            const std::string & message) const  [protected]
```

Displays a general error message.

Displays an error message with the provided message text.

Parameters

message	The error message to display.

4.53.3.9 displayMenu()

```
void IMenu::displayMenu () const  [protected]
```

Displays the formatted menu, including sections and options.

Displays the menu by printing its sections, options, and borders.

4.53.3.10 displaySuccessMessage()

```
void IMenu::displaySuccessMessage (
            const std::string & message) const  [protected]
```

Displays a success message in green color.

Displays a success message with green color.

Parameters

message	The success message to display.

4.53.3.11 handleInput()

```
virtual void IMenu::handleInput ()  [pure virtual]
```

Pure virtual function to handle user input in the menu.

Must be implemented by derived classes to process user interaction.

Implemented in BuildingsMenu, BuildingsStatMenu, BuyMenu, BuyRoadMenu, DemolishMenu, DisplayCityMenu, MainMenu, PolicyMenu, StatsMenu, TaxMenu, and UpgradesMenu.

4.53.3.12 indexToExtendedChar()

```
char IMenu::indexToExtendedChar (
            int index)  [static], [protected]
```

Converts a numeric index (0-99) to a single character in an extended set.

Parameters

index	Numeric index to convert (0-99).

Returns

The corresponding character for the given index.

Exceptions

std::out_of_range	If the index is outside the allowed range.

4.53.3.13 printBottomBorder()

```
void IMenu::printBottomBorder (
            int width) const  [protected]
```

Prints the bottom border of the menu using box-drawing characters.

Parameters

width	The width of the menu.

4.53.3.14 printDoubleLineDivider()

```
void IMenu::printDoubleLineDivider (
            int width) const  [protected]
```

Prints a double-line divider for the main heading of the menu.

Parameters

width	The width of the menu.

4.53.3.15 printSectionDivider()

```
void IMenu::printSectionDivider (
            int width) const  [protected]
```

Prints a section divider in the menu using box-drawing characters.

Parameters

width	The width of the menu.

4.53.3.16 printTopBorder()

```
void IMenu::printTopBorder (
            int width) const  [protected]
```

Prints the top border of the menu using box-drawing characters.

Parameters

width	The width of the menu.

4.53.3.17 repeat()

```
std::string IMenu::repeat (
            const std::string & str,
            int times) const  [protected]
```

Utility function to repeat a string multiple times.

Repeats a given string for a specified number of times.

Parameters

str	The string to repeat.
times	The number of times to repeat the string.

Returns

A concatenated string repeated the specified number of times.

Parameters

str	The string to be repeated.
times	The number of times to repeat the string.

Returns

The repeated string.

4.53.3.18 setHeading()

```
void IMenu::setHeading (
            const std::string & heading)
```

Sets the heading of the menu.

Parameters

heading	The new heading for the menu.

4.53.3.19 stripColorCodes()

```
std::string IMenu::stripColorCodes (
            const std::string & input) const  [protected]
```

Strips ANSI color codes from a string.

Parameters

input	The string potentially containing color codes.

Returns

The string with color codes removed.

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

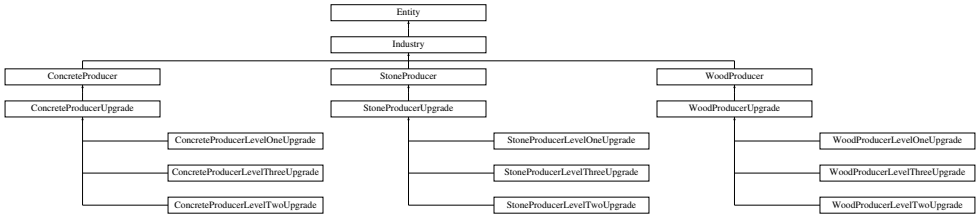
- src/menus/base/IMenu.h
- src/menus/base/IMenu.cpp

4.54 Industry Class Reference

Represents an industrial entity within the game.

`#include <Industry.h>`

Inheritance diagram for Industry:



Public Member Functions

- Industry (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Industry entity with specified attributes.
- Industry (Industry ∗industry)

  Copy constructor for the Industry class.
- virtual ∼**Industry** ()

  Virtual destructor for the Industry class.
- virtual void update ()=0

  Updates the state of the industry entity.
- virtual Entity ∗ clone ()=0

  Creates a clone of the industry entity.
- virtual int getOutput ()

  Gets the production output of the industry.
- void setOutput (int output)

  Sets the production output of the industry.
- virtual int getLevel ()

  Gets the current level of the industry.
- virtual Cost getCost ()

  Gets the cost of an upgrade.
- virtual Entity ∗ upgrade ()=0

  Upgrades the current industry to the next level.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).
- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.54.1 Detailed Description

Represents an industrial entity within the game.

The Industry class is responsible for managing properties specific to industrial entities, such as production output. Derived classes are expected to implement the `update` and `clone` methods for specific industrial behavior.

4.54.2 Constructor & Destructor Documentation

4.54.2.1 Industry() [1/2]

```
Industry::Industry (
            EntityConfig ec,
            Size size,
            int xPos,
            int yPos)
```

Constructs an Industry entity with specified attributes.

Parameters

ec	Configuration settings for the entity.
size	Size of the industrial entity.
xPos	X-coordinate position of the entity.
yPos	Y-coordinate position of the entity.

4.54.2.2 Industry() [2/2]

```
Industry::Industry (
            Industry * industry)
```

Copy constructor for the Industry class.

Creates a new Industry entity by copying the attributes of an existing Industry.

Parameters

industry	Pointer to the Industry object to be copied.

4.54.3 Member Function Documentation

4.54.3.1 clone()

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

Creates a clone of the industry entity.

This function must be implemented in derived classes.

Returns

A pointer to the cloned Industry entity.

Implements Entity.

Implemented in ConcreteProducer, ConcreteProducerLevelOneUpgrade, ConcreteProducerLevelThreeUpgrade, ConcreteProducerLevelTwoUpgrade, ConcreteProducerUpgrade, StoneProducer, StoneProducerLevelOneUpgrade, StoneProducerLevelThreeUpgrade, StoneProducerLevelTwoUpgrade, StoneProducerUpgrade, WoodProducer, WoodProducerLevelOneUpgrade, WoodProducerLevelThreeUpgrade, WoodProducerLevelTwoUpgrade, and WoodProducerUpgrade.

4.54.3.2 getCost()

```
Cost Industry::getCost ()  [virtual]
```

Gets the cost of an upgrade.

Returns

Cost struct of various costs for upgrade.

Reimplemented in ConcreteProducerLevelOneUpgrade, ConcreteProducerLevelThreeUpgrade, ConcreteProducerLevelTwoUpgrade, ConcreteProducerUpgrade, StoneProducerLevelOneUpgrade, StoneProducerLevelThreeUpgrade, StoneProducerLevelTwoUpgrade, StoneProducerUpgrade, WoodProducerLevelOneUpgrade, WoodProducerLevelThreeUpgrade, WoodProducerLevelTwoUpgrade, and WoodProducerUpgrade.

4.54.3.3 getLevel()

```
int Industry::getLevel ()  [virtual]
```

Gets the current level of the industry.

Returns

The level of the industry (default implementation returns 0).

Reimplemented in ConcreteProducerLevelOneUpgrade, ConcreteProducerLevelThreeUpgrade, ConcreteProducerLevelTwoUpgrade, StoneProducerLevelOneUpgrade, StoneProducerLevelThreeUpgrade, StoneProducerLevelTwoUpgrade, WoodProducerLevelOneUpg WoodProducerLevelThreeUpgrade, and WoodProducerLevelTwoUpgrade.

4.54.3.4 getOutput()

```
int Industry::getOutput ()  [virtual]
```

Gets the production output of the industry.

Returns

The production output value.

Reimplemented in ConcreteProducerLevelOneUpgrade, ConcreteProducerLevelThreeUpgrade, ConcreteProducerLevelTwoUpgrade, ConcreteProducerUpgrade, StoneProducerLevelOneUpgrade, StoneProducerLevelThreeUpgrade, StoneProducerLevelTwoUpgrade, StoneProducerUpgrade, WoodProducerLevelOneUpgrade, WoodProducerLevelThreeUpgrade, WoodProducerLevelTwoUpgrade, and WoodProducerUpgrade.

4.54.3.5 setOutput()

```
void Industry::setOutput (
            int output)
```

Sets the production output of the industry.

Parameters

output	The new output value.

4.54.3.6 update()

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

Updates the state of the industry entity.

This function must be implemented in derived classes.

Implements Entity.

Implemented in ConcreteProducer, ConcreteProducerLevelOneUpgrade, ConcreteProducerLevelThreeUpgrade, ConcreteProducerLevelTwoUpgrade, ConcreteProducerUpgrade, StoneProducer, StoneProducerLevelOneUpgrade, StoneProducerLevelThreeUpgrade, StoneProducerLevelTwoUpgrade, StoneProducerUpgrade, WoodProducer, WoodProducerLevelOneUpgrade, WoodProducerLevelThreeUpgrade, WoodProducerLevelTwoUpgrade, and WoodProducerUpgrade.

4.54.3.7 upgrade()

```
virtual Entity * Industry::upgrade ()  [pure virtual]
```

Upgrades the current industry to the next level.

Returns

A pointer to the upgraded industry instance, or nullptr if already at maximum level.

Implemented in ConcreteProducer, ConcreteProducerLevelOneUpgrade, ConcreteProducerLevelThreeUpgrade, ConcreteProducerLevelTwoUpgrade, ConcreteProducerUpgrade, StoneProducer, StoneProducerLevelOneUpgrade, StoneProducerLevelThreeUpgrade, StoneProducerLevelTwoUpgrade, StoneProducerUpgrade, WoodProducer, WoodProducerLevelOneUpgrade, WoodProducerLevelThreeUpgrade, WoodProducerLevelTwoUpgrade, and WoodProducerUpgrade.

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

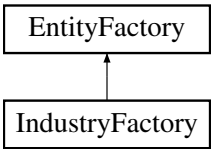
- src/entities/industry/base/Industry.h
- src/entities/industry/base/Industry.cpp

4.55 IndustryFactory Class Reference

Factory class for creating industrial entities such as concrete, stone, and wood producers.

```
#include <IndustryFactory.h>
```

Inheritance diagram for IndustryFactory:



Public Member Functions

- **IndustryFactory** ()
  Default constructor for IndustryFactory.
- ~**IndustryFactory** ()
  Destructor for IndustryFactory.
- virtual Entity * createEntity (EntityType type, Size size, int xPos, int yPos)
  Creates an industrial entity of a specified type and size at the given coordinates.

Public Member Functions inherited from EntityFactory

- **EntityFactory** ()
  Default constructor for EntityFactory.
- virtual ~**EntityFactory** ()
  Virtual destructor for EntityFactory.

4.55.1 Detailed Description

Factory class for creating industrial entities such as concrete, stone, and wood producers.

This class inherits from EntityFactory and provides methods to create various-sized industrial entities (small, medium, and large) at specific positions in the environment.

4.55.2 Member Function Documentation

4.55.2.1 createEntity()

```
Entity * IndustryFactory::createEntity (
            EntityType type,
            Size size,
            int xPos,
            int yPos)  [virtual]
```

Creates an industrial entity of a specified type and size at the given coordinates.

Parameters

type	The type of industrial entity to create (e.g., ConcreteProducer, StoneProducer, WoodProducer).
size	The size of the entity (small, medium, or large).
xPos	The x-coordinate for the entity's position.
yPos	The y-coordinate for the entity's position.

Returns

A pointer to the created Entity.

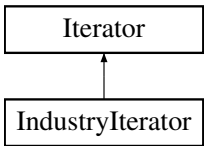
Implements EntityFactory.

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

- src/factory/industry/IndustryFactory.h
- src/factory/industry/IndustryFactory.cpp

4.56 IndustryIterator Class Reference

Inheritance diagram for IndustryIterator:



Public Member Functions

- **IndustryIterator** ()

  Construct a new Industry Iterator:: Industry Iterator object.
- ~**IndustryIterator** ()

  Destroy the Industry Iterator:: Industry Iterator object.
- IndustryIterator (std::vector< std::vector< Entity * > > &grid)

  Construct a new Industry Iterator:: Industry Iterator object.
- void first ()

  Sets the iterator to the first unvisited Industry.
- void next ()

  Advances to the next unvisited Industry.
- bool hasNext ()

  Checks if there is another unvisited Industry.
- Entity * current ()

  Returns the current Industry.

Public Member Functions inherited from Iterator

- **Iterator** ()

  Construct a new Iterator object, initializing row and column to zero.
- virtual ~**Iterator** ()

  Destroy the Iterator object.
- **Iterator** (std::vector< std::vector< Entity * > > &grid)
- virtual int getRow ()

  Get the current row index of the iterator.
- virtual int getCol ()

  Get the current column index of the iterator.

Additional Inherited Members

Protected Member Functions inherited from Iterator

- bool isVisited (Entity *entity)

  Check if the specified entity has been visited.
- void markVisited (Entity *entity)

  Mark the specified entity as visited.

Protected Attributes inherited from Iterator

- std::vector< std::vector< Entity * > > **grid**
- std::vector< std::vector< Entity * > >::iterator **currRow**
- std::vector< Entity * >::iterator **curr**
- int **row**
- int **col**
- std::unordered_set< Entity * > **visitedEntities**

4.56.1 Constructor & Destructor Documentation

4.56.1.1 IndustryIterator()

```
IndustryIterator::IndustryIterator (
            std::vector< std::vector< Entity * > > & grid)
```

Construct a new Industry Iterator:: Industry Iterator object.

Parameters

grid	

4.56.2 Member Function Documentation

4.56.2.1 current()

```
Entity * IndustryIterator::current ()  [virtual]
```

Returns the current Industry.

Returns

Entity*

Implements Iterator.

4.56.2.2 first()

```
void IndustryIterator::first ()  [virtual]
```

Sets the iterator to the first unvisited Industry.

Implements Iterator.

4.56.2.3 hasNext()

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

Checks if there is another unvisited Industry.

Returns

true if there is another unvisited Industry, false otherwise

Implements Iterator.

4.56.2.4 next()

```
void IndustryIterator::next ()  [virtual]
```

Advances to the next unvisited Industry.

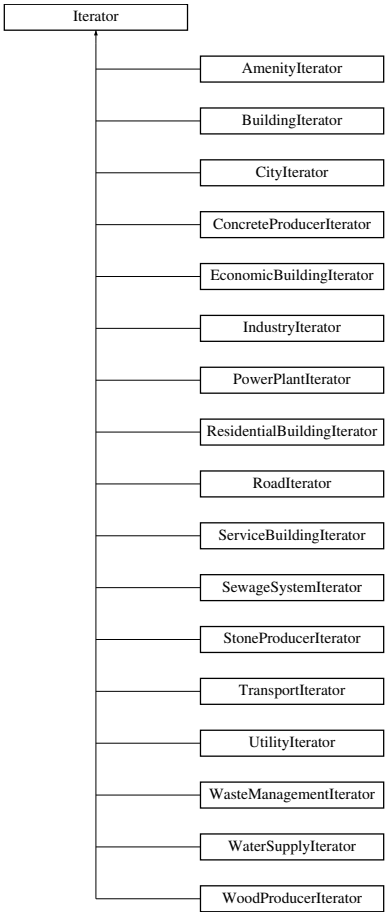
Implements Iterator.

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

- src/iterators/industry/IndustryIterator.h
- src/iterators/industry/IndustryIterator.cpp

4.57 Iterator Class Reference

Inheritance diagram for Iterator:



Public Member Functions

- **Iterator** ()

  Construct a new Iterator object, initializing row and column to zero.
- virtual ~**Iterator** ()

  Destroy the Iterator object.
- **Iterator** (std::vector< std::vector< Entity * > > &grid)
- virtual void first ()=0
- virtual void next ()=0
- virtual bool hasNext ()=0
- virtual Entity * current ()=0
- virtual int getRow ()

  Get the current row index of the iterator.
- virtual int getCol ()

  Get the current column index of the iterator.

Protected Member Functions

- bool isVisited (Entity *entity)

  Check if the specified entity has been visited.
- void markVisited (Entity *entity)

  Mark the specified entity as visited.

Protected Attributes

- std::vector< std::vector< Entity * > > **grid**
- std::vector< std::vector< Entity * > >::iterator **currRow**
- std::vector< Entity * >::iterator **curr**
- int **row**
- int **col**
- std::unordered_set< Entity * > **visitedEntities**

4.57.1 Member Function Documentation

4.57.1.1 current()

```
virtual Entity * Iterator::current ()  [pure virtual]
```

Implemented in AmenityIterator, BuildingIterator, CityIterator, ConcreteProducerIterator, EconomicBuildingIterator, IndustryIterator, PowerPlantIterator, ResidentialBuildingIterator, RoadIterator, ServiceBuildingIterator, SewageSystemIterator, StoneProducerIterator, TransportIterator, UtilityIterator, WasteManagementIterator, WaterSupplyIterator, and WoodProducerIterator.

4.57.1.2 first()

```
virtual void Iterator::first ()  [pure virtual]
```

Implemented in AmenityIterator, BuildingIterator, CityIterator, ConcreteProducerIterator, EconomicBuildingIterator, IndustryIterator, PowerPlantIterator, ResidentialBuildingIterator, RoadIterator, ServiceBuildingIterator, SewageSystemIterator, StoneProducerIterator, TransportIterator, UtilityIterator, WasteManagementIterator, WaterSupplyIterator, and WoodProducerIterator.

4.57.1.3 getCol()

```
int Iterator::getCol ()  [virtual]
```

Get the current column index of the iterator.

Returns

int The current column index.

4.57.1.4 getRow()

```
int Iterator::getRow ()  [virtual]
```

Get the current row index of the iterator.

Returns

int The current row index.

4.57.1.5 hasNext()

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

Implemented in AmenityIterator, BuildingIterator, CityIterator, ConcreteProducerIterator, EconomicBuildingIterator, IndustryIterator, PowerPlantIterator, ResidentialBuildingIterator, RoadIterator, ServiceBuildingIterator, SewageSystemIterator, StoneProducerIterator, TransportIterator, UtilityIterator, WasteManagementIterator, WaterSupplyIterator, and WoodProducerIterator.

4.57.1.6 isVisited()

```
bool Iterator::isVisited (
            Entity * entity)  [protected]
```

Check if the specified entity has been visited.

Parameters

entity	Pointer to the entity to check.

Returns

true if the entity has been visited; false otherwise.

4.57.1.7 markVisited()

```
void Iterator::markVisited (
            Entity * entity)  [protected]
```

Mark the specified entity as visited.

Parameters

entity	Pointer to the entity to mark as visited.

4.57.1.8 next()

```
virtual void Iterator::next ()  [pure virtual]
```

Implemented in AmenityIterator, BuildingIterator, CityIterator, ConcreteProducerIterator, EconomicBuildingIterator, IndustryIterator, PowerPlantIterator, ResidentialBuildingIterator, RoadIterator, ServiceBuildingIterator, SewageSystemIterator, StoneProducerIterator, TransportIterator, UtilityIterator, WasteManagementIterator, WaterSupplyIterator, and WoodProducerIterator.

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

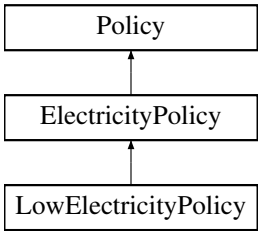
- src/iterators/base/Iterator.h
- src/iterators/base/Iterator.cpp

4.58 LowElectricityPolicy Class Reference

Concrete strategy for low electricity policy.

`#include <LowElectricityPolicy.h>`

Inheritance diagram for LowElectricityPolicy:



Public Member Functions

- int calculateElectricityUsage (int electricityUsage) override

  Overrides calculateElectricityUsage to reduce electricity usage.
- **LowElectricityPolicy** ()

  Constructor for LowElectricityPolicy. Initializes the policy with specific name and detail.

Public Member Functions inherited from ElectricityPolicy

- ElectricityPolicy (const std::string &name, const std::string &detail)

  Constructor for ElectricityPolicy.
- virtual ~**ElectricityPolicy** ()

  Virtual destructor for ElectricityPolicy.

Public Member Functions inherited from Policy

- Policy (const std::string &name, const std::string &detail)

  Constructor for Policy.
- Memento * createMemento () const

  Creates a memento to store the current state of the policy.
- void setMemento (const Memento *memento)

  Sets the policy state from a memento.
- std::string getName () const

  Gets the name of the policy.
- std::string getDetail () const

  Gets the detail of the policy.

4.58.1 Detailed Description

Concrete strategy for low electricity policy.

Reduces electricity usage by applying a low usage factor.

4.58.2 Member Function Documentation

4.58.2.1 calculateElectricityUsage()

```
int LowElectricityPolicy::calculateElectricityUsage (
            int electricityUsage)  [override], [virtual]
```

Overrides calculateElectricityUsage to reduce electricity usage.

Implementation of LowElectricityPolicy to reduce electricity usage by 25%.

Parameters

electricityUsage	Initial electricity usage.

Returns

int Reduced electricity usage (e.g., 75% of the original).

Parameters

electricityUsage	Initial electricity usage.

Returns

int Reduced electricity usage.

Implements ElectricityPolicy.

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

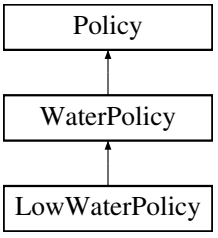
- src/policies/electricity/LowElectricityPolicy.h
- src/policies/electricity/LowElectricityPolicy.cpp

4.59 LowWaterPolicy Class Reference

Concrete strategy for low water policy.

```
#include <LowWaterPolicy.h>
```

Inheritance diagram for LowWaterPolicy:



Public Member Functions

- int calculateWaterUsage (int waterUsage) override

  Overrides calculateWaterUsage to reduce water usage.
- **LowWaterPolicy** ()

  Constructor for LowWaterPolicy. Initializes the policy with specific name and detail.

Public Member Functions inherited from WaterPolicy

- WaterPolicy (const std::string &name, const std::string &detail)

  Constructor for WaterPolicy.
- virtual ~**WaterPolicy** ()

  Virtual destructor for WaterPolicy.

Public Member Functions inherited from Policy

- Policy (const std::string &name, const std::string &detail)

  Constructor for Policy.
- Memento * createMemento () const

  Creates a memento to store the current state of the policy.
- void setMemento (const Memento *memento)

  Sets the policy state from a memento.
- std::string getName () const

  Gets the name of the policy.
- std::string getDetail () const

  Gets the detail of the policy.

4.59.1 Detailed Description

Concrete strategy for low water policy.

Reduces water usage by applying a low usage factor.

4.59.2 Member Function Documentation

4.59.2.1 calculateWaterUsage()

```
int LowWaterPolicy::calculateWaterUsage (
            int waterUsage)  [override], [virtual]
```

Overrides calculateWaterUsage to reduce water usage.

Implementation of LowWaterPolicy to reduce water usage by 20%.

Parameters

waterUsage	Initial water usage.

Returns

int Reduced water usage (e.g., 80% of the original).

Parameters

waterUsage	Initial water usage.

Returns

int Reduced water usage.

Implements WaterPolicy.

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

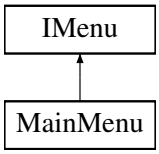
- src/policies/water/LowWaterPolicy.h
- src/policies/water/LowWaterPolicy.cpp

4.60 MainMenu Class Reference

Represents the main menu of the game, providing primary navigation options.

`#include <MainMenu.h>`

Inheritance diagram for MainMenu:



Public Member Functions

- MainMenu ()

  Constructs the MainMenu.
- ~MainMenu ()

  Destructor for MainMenu.
- void display () const override

  Displays the MainMenu.
- void handleInput () override

  Handles user input in the MainMenu.

Public Member Functions inherited from IMenu

- **IMenu** ()=default

  Default constructor for IMenu.
- IMenu (std::string heading)

  Constructor to initialize a menu with a specified heading.
- virtual ~**IMenu** ()=default

  Virtual destructor for IMenu.
- void setHeading (const std::string &heading)

  Sets the heading of the menu.

Additional Inherited Members

Protected Member Functions inherited from IMenu

- std::string repeat (const std::string &str, int times) const

  Utility function to repeat a string multiple times.
- int calculateMaxWidth (const std::string &menuHeading, const std::vector< Section > &sections) const

  Calculates the maximum width required for the menu.
- void printTopBorder (int width) const

  Prints the top border of the menu using box-drawing characters.
- void printBottomBorder (int width) const

  Prints the bottom border of the menu using box-drawing characters.
- void printSectionDivider (int width) const

  Prints a section divider in the menu using box-drawing characters.
- void printDoubleLineDivider (int width) const

  Prints a double-line divider for the main heading of the menu.
- std::string centerText (const std::string &text, int width) const

  Centers text within a specified width using space padding.
- std::string centerTextWithChar (const std::string &text, int width, const std::string &padChar) const

  Centers text within a specified width using a custom character for padding.
- void displayMenu () const

  Displays the formatted menu, including sections and options.
- void **displayChoicePrompt** () const

  Displays the choice prompt for user input.
- void displayChoiceMessagePrompt (const std::string &message) const

  Displays a custom message prompt for user input.
- void **displayInvalidChoice** () const

  Displays an error message when the user makes an invalid choice.
- void displayErrorMessage (const std::string &message) const

  Displays a general error message.
- void displaySuccessMessage (const std::string &message) const

  Displays a success message in green color.
- void **displayPressEnterToContinue** () const

  Displays a message asking the user to press Enter to continue.
- void **clearScreen** () const

  Clears the terminal screen.
- std::string stripColorCodes (const std::string &input) const

  Strips ANSI color codes from a string.
- virtual void displayAvailablePositions (const std::vector< std::vector< int > > &positions) const

  Displays available positions on the city grid for an entity.

Static Protected Member Functions inherited from IMenu

- static char indexToExtendedChar (int index)

  Converts a numeric index (0-99) to a single character in an extended set.
- static std::string coordinatesToLabel (int x, int y)

  Converts x and y coordinates to a labeled string (e.g., "A, 1").

Protected Attributes inherited from IMenu

- std::vector< Section > **sections**

  List of sections contained in the menu.
- std::string **menuHeading**

  The heading/title of the menu.
- bool **hasExited**

  Flag indicating if the menu has been exited.
- CityManager **cityManager**

  Manager for city-related operations.
- bool **displayResources**

  Flag indicating whether to display resources in the menu.
- bool **isInfoMenu**

  Flag indicating whether to display option numbers.

Static Protected Attributes inherited from IMenu

- static const char * **RESET** = "\033[0m"

  ANSI color codes and styles for use in all menus.
- static const char * **BOLD_WHITE** = "\033[1;37m"
- static const char * **NORMAL_WHITE** = "\033[0;37m"
- static const char * **DARK_GRAY** = "\033[1;30m"
- static const char * **BOLD_YELLOW** = "\033[1;33m"
- static const char * **BOLD_GREEN** = "\033[1;32m"
- static const char * **BOLD_RED** = "\033[1;31m"
- static const char * **BOLD_CYAN** = "\033[1;36m"
- static const char * **BLUE** = "\033[34m"

4.60.1 Detailed Description

Represents the main menu of the game, providing primary navigation options.

The MainMenu class offers the player key interactions, such as accessing sub-menus for buildings, upgrades, policies, taxes, city stats, and the option to quit the game.

4.60.2 Constructor & Destructor Documentation

4.60.2.1 MainMenu()

```
MainMenu::MainMenu ()
```

Constructs the MainMenu.

Constructor for MainMenu. Initializes the menu options and navigation for the Main Menu.

Initializes the menu with various sections and options for player interaction.

4.60.2.2 ~MainMenu()

```
MainMenu::~MainMenu ()
```

Destructor for MainMenu.

Destructor for MainMenu. Cleans up any resources used by the Main Menu.

Cleans up resources used by the MainMenu instance.

4.60.3 Member Function Documentation

4.60.3.1 display()

```
void MainMenu::display () const  [override], [virtual]
```

Displays the MainMenu.

Displays the Main Menu. Uses the inherited displayMenu() method to render the menu with all the options.

Overrides the display method from IMenu to render the main menu interface.

Implements IMenu.

4.60.3.2 handleInput()

```
void MainMenu::handleInput ()  [override], [virtual]
```

Handles user input in the MainMenu.

Handles user input in the Main Menu.

Processes user selections to navigate to other menus or exit the game. This method listens for input, validates it, and triggers corresponding actions.

This function manages the logic for navigating between the various menus or exiting the game based on the player's input.

Implements IMenu.

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

- src/menus/main/MainMenu.h
- src/menus/main/MainMenu.cpp

4.61 Memento Class Reference

Class representing a Memento for saving and restoring the state of a Policy.

```
#include <Memento.h>
```

Public Member Functions

- Memento (const std::string &name, const std::string &detail)

  Constructor for Memento.
- std::string getName () const

  Gets the name of the policy.
- std::string getDetail () const

  Gets the detail of the policy.
- void setName (const std::string &name)

  Sets the name of the policy.
- void setDetail (const std::string &detail)

  Sets the detail of the policy.

4.61.1 Detailed Description

Class representing a Memento for saving and restoring the state of a Policy.

4.61.2 Constructor & Destructor Documentation

4.61.2.1 Memento()

```
Memento::Memento (
            const std::string & name,
            const std::string & detail)
```

Constructor for Memento.

Constructor implementation for Memento.

Parameters

name	Name of the policy.
detail	Detail of the policy.

4.61.3 Member Function Documentation

4.61.3.1 getDetail()

```
std::string Memento::getDetail () const
```

Gets the detail of the policy.

Returns

std::string Detail of the policy.

4.61.3.2 getName()

```
std::string Memento::getName () const
```

Gets the name of the policy.

Returns

std::string Name of the policy.

4.61.3.3 setDetail()

```
void Memento::setDetail (
            const std::string & detail)
```

Sets the detail of the policy.

Parameters

detail	Detail of the policy.

4.61.3.4 setName()

```
void Memento::setName (
            const std::string & name)
```

Sets the name of the policy.

Parameters

name	Name of the policy.

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

- src/utils/Memento.h
- src/utils/Memento.cpp

4.62 MenuManager Class Reference

Manages the different menus in the game and allows switching between them.

```
#include <MenuManager.h>
```

Public Member Functions

- void setCurrentMenu (Menu menuType)

  Sets the current menu to be displayed, using an enum value to select the menu.
- void setCurrentMenu (std::shared_ptr< IMenu > menu)

  Sets the current menu using a dynamic menu object.
- void displayCurrentMenu ()

  Displays the currently active menu.
- void handleCurrentMenuInput ()

  Handles user input for the currently active menu.
- void setCity (City *city)

  Sets the City object reference for use in menus.
- City * getCity () const

  Retrieves the City object reference.
- void clearScreen () const

  Clears the terminal screen.

Static Public Member Functions

- static MenuManager & instance ()

  Provides access to the single instance of MenuManager.

4.62.1 Detailed Description

Manages the different menus in the game and allows switching between them.

MenuManager implements the Singleton design pattern, ensuring that only one instance of the class exists and provides a global point of access to it.

4.62.2 Member Function Documentation

4.62.2.1 clearScreen()

```
void MenuManager::clearScreen () const
```

Clears the terminal screen.

Utility method to clear the screen for cleaner menu displays.

4.62.2.2 displayCurrentMenu()

```
void MenuManager::displayCurrentMenu ()
```

Displays the currently active menu.

Calls the display method of the active menu to render its content to the terminal.

4.62.2.3 getCity()

```
City * MenuManager::getCity () const
```

Retrieves the City object reference.

Gets the reference to the City object.

Provides access to the City object used by the MenuManager.

Returns

Pointer to the City object.

Pointer to the City object.

4.62.2.4 handleCurrentMenuInput()

```
void MenuManager::handleCurrentMenuInput ()
```

Handles user input for the currently active menu.

Handles the input for the current menu.

Delegates input handling to the active menu's handleInput method.

4.62.2.5 instance()

```
MenuManager & MenuManager::instance ()  [static]
```

Provides access to the single instance of MenuManager.

Singleton instance method.

Ensures that only one instance of MenuManager is created.

Returns

The singleton instance of MenuManager.

The singleton instance of MenuManager.

4.62.2.6 setCity()

```
void MenuManager::setCity (
            City * city)
```

Sets the City object reference for use in menus.

Sets the reference to the City object to be used by menus.

Links the City object to the MenuManager, allowing menus to access and interact with city data.

Parameters

city	Pointer to the City object.
city	Pointer to the City object.

4.62.2.7 setCurrentMenu() [1/2]

```
void MenuManager::setCurrentMenu (
            Menu menuType)
```

Sets the current menu to be displayed, using an enum value to select the menu.

Sets the current menu by enum key.

Changes the active menu based on the provided menu type, allowing seamless transitions between menus.

Parameters

menuType	The type of the menu to switch to, defined by the Menu enum.
menuType	The type of the menu to switch to.

4.62.2.8 setCurrentMenu() [2/2]

```
void MenuManager::setCurrentMenu (
            std::shared_ptr< IMenu > menu)
```

Sets the current menu using a dynamic menu object.

Sets the current menu using a dynamic menu passed as a shared pointer.

Allows setting the current menu by passing a shared pointer to a custom menu object.

Parameters

menu	The shared pointer to the IMenu object to set as current.
menu	The shared pointer to the menu to set as current.

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

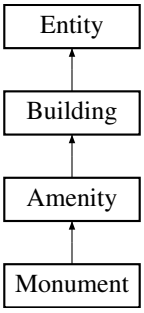
- src/menus/base/MenuManager.h
- src/menus/base/MenuManager.cpp

4.63 Monument Class Reference

Represents a monument entity within the game.

`#include <Monument.h>`

Inheritance diagram for Monument:



Public Member Functions

- Monument (EntityConfig ec, Size size, int xPos, int yPos)
  Constructs a Monument with specified attributes.
- Monument (Monument ∗monument)
  Copy constructor for the Monument class.
- virtual ∼**Monument** ()
  Destructor for the Monument class.
- void update ()
  Updates the monument's state.
- Entity ∗ clone ()
  Creates a clone of the monument.

Public Member Functions inherited from Amenity

- Amenity (EntityConfig ec, Size size, int xPos, int yPos)
  Constructs an Amenity with specified attributes.
- Amenity (Amenity ∗amenity)
  Copy constructor for the Amenity class.
- virtual ∼**Amenity** ()
  Virtual destructor for the Amenity class.

Public Member Functions inherited from Building

- Building (EntityConfig ec, Size size, int xPos, int yPos)
  Parameterized constructor for the Building class.
- Building (Building ∗building)
  Copy constructor for the Building class.
- virtual ∼Building ()
  Destructor for the Building class.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).
- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.63.1 Detailed Description

Represents a monument entity within the game.

The Monument class is a specific type of Amenity with its own unique characteristics and behaviors. This class provides implementations for updating the monument's state and cloning itself.

4.63.2 Constructor & Destructor Documentation

4.63.2.1 Monument() [1/2]

```
Monument::Monument (
            EntityConfig ec,
            Size size,
            int xPos,
            int yPos)
```

Constructs a Monument with specified attributes.

Parameters

ec	Configuration containing resource consumption and properties.
size	Size of the monument.
xPos	X-coordinate position of the monument.
yPos	Y-coordinate position of the monument.

4.63.2.2 Monument() [2/2]

```
Monument::Monument (
            Monument * monument)
```

Copy constructor for the Monument class.

Creates a new Monument by copying the attributes of an existing Monument.

Parameters

monument	Pointer to the Monument object to be copied.

4.63.3 Member Function Documentation

4.63.3.1 clone()

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

Creates a clone of the monument.

Returns

A pointer to the cloned Monument.

Implements Amenity.

4.63.3.2 update()

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

Updates the monument's state.

Implements Amenity.

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

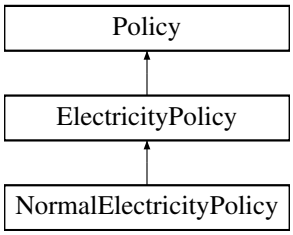
- src/entities/building/amenity/Monument.h
- src/entities/building/amenity/Monument.cpp

4.64 NormalElectricityPolicy Class Reference

Concrete strategy for normal electricity policy.

`#include <NormalElectricityPolicy.h>`

Inheritance diagram for NormalElectricityPolicy:



Public Member Functions

- int calculateElectricityUsage (int electricityUsage) override

  Overrides calculateElectricityUsage to keep electricity usage unchanged.
- **NormalElectricityPolicy** ()

  Constructor for NormalElectricityPolicy. Initializes the policy with specific name and detail.

Public Member Functions inherited from ElectricityPolicy

- ElectricityPolicy (const std::string &name, const std::string &detail)

  Constructor for ElectricityPolicy.
- virtual ~**ElectricityPolicy** ()

  Virtual destructor for ElectricityPolicy.

Public Member Functions inherited from Policy

- Policy (const std::string &name, const std::string &detail)

  Constructor for Policy.
- Memento * createMemento () const

  Creates a memento to store the current state of the policy.
- void setMemento (const Memento *memento)

  Sets the policy state from a memento.
- std::string getName () const

  Gets the name of the policy.
- std::string getDetail () const

  Gets the detail of the policy.

4.64.1 Detailed Description

Concrete strategy for normal electricity policy.

Keeps electricity usage unchanged.

4.64.2 Member Function Documentation

4.64.2.1 calculateElectricityUsage()

```
int NormalElectricityPolicy::calculateElectricityUsage (
            int electricityUsage)  [override], [virtual]
```

Overrides calculateElectricityUsage to keep electricity usage unchanged.

Implementation of NormalElectricityPolicy to keep electricity usage unchanged.

Parameters

electricityUsage	Initial electricity usage.

Returns

int Unchanged electricity usage.

Implements ElectricityPolicy.

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

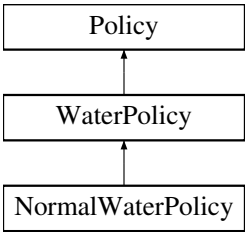
- src/policies/electricity/NormalElectricityPolicy.h
- src/policies/electricity/NormalElectricityPolicy.cpp

4.65 NormalWaterPolicy Class Reference

Concrete strategy for normal water policy.

```
#include <NormalWaterPolicy.h>
```

Inheritance diagram for NormalWaterPolicy:



Public Member Functions

- int calculateWaterUsage (int waterUsage) override

  Overrides calculateWaterUsage to keep water usage unchanged.
- **NormalWaterPolicy** ()

  Constructor for NormalWaterPolicy. Initializes the policy with specific name and detail.

Public Member Functions inherited from WaterPolicy

- WaterPolicy (const std::string &name, const std::string &detail)
  Constructor for WaterPolicy.
- virtual ~**WaterPolicy** ()
  Virtual destructor for WaterPolicy.

Public Member Functions inherited from Policy

- Policy (const std::string &name, const std::string &detail)
  Constructor for Policy.
- Memento * createMemento () const
  Creates a memento to store the current state of the policy.
- void setMemento (const Memento *memento)
  Sets the policy state from a memento.
- std::string getName () const
  Gets the name of the policy.
- std::string getDetail () const
  Gets the detail of the policy.

4.65.1 Detailed Description

Concrete strategy for normal water policy.

Keeps the water usage unchanged.

4.65.2 Member Function Documentation

4.65.2.1 calculateWaterUsage()

```
int NormalWaterPolicy::calculateWaterUsage (
            int waterUsage)  [override], [virtual]
```

Overrides calculateWaterUsage to keep water usage unchanged.

Implementation of NormalWaterPolicy to keep water usage unchanged.

Parameters

waterUsage	Initial water usage.

Returns

int The same water usage as provided.

Parameters

waterUsage	Initial water usage.

Returns

int Unchanged water usage.

Implements WaterPolicy.

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

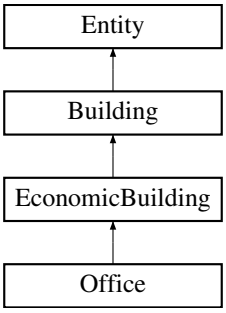
- src/policies/water/NormalWaterPolicy.h
- src/policies/water/NormalWaterPolicy.cpp

4.66 Office Class Reference

Concrete class representing an office building in the city builder/manager game.

`#include <Office.h>`

Inheritance diagram for Office:



Public Member Functions

- Office (EntityConfig ec, Size size, int xPos, int yPos)
  Parameterized constructor for the Office class.
- Office (Office ∗office)
  Copy constructor for the Office class.
- ∼Office ()
  Destructor for the Office class.
- void update ()
  Updates the state of the office building entity.
- Entity ∗ clone ()
  Clones the office entity.

Public Member Functions inherited from EconomicBuilding

- EconomicBuilding (EntityConfig ec, Size size, int xPos, int yPos)
  Parameterized constructor for the EconomicBuilding class.
- EconomicBuilding (EconomicBuilding ∗economic)
  Copy constructor for the EconomicBuilding class.
- virtual ∼EconomicBuilding ()
  Destructor for the EconomicBuilding class.

Public Member Functions inherited from Building

- Building (EntityConfig ec, Size size, int xPos, int yPos)
  Parameterized constructor for the Building class.
- Building (Building ∗building)
  Copy constructor for the Building class.
- virtual ∼Building ()
  Destructor for the Building class.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).
- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.66.1 Detailed Description

Concrete class representing an office building in the city builder/manager game.

Office is a type of EconomicBuilding that houses businesses and supports administrative activities.

4.66.2 Constructor & Destructor Documentation

4.66.2.1 Office() [1/2]

```
Office::Office (
            EntityConfig ec,
            Size size,
            int xPos,
            int yPos)
```

Parameterized constructor for the Office class.

Parameters

ec	The configuration object containing general entity properties.
size	The size of the office entity.
xPos	The x-coordinate position of the office on the map.
yPos	The y-coordinate position of the office on the map.

Initializes a new instance of the Office class with specific values.

4.66.2.2 Office() [2/2]

```
Office::Office (
            Office * office)
```

Copy constructor for the Office class.

Parameters

office	A pointer to an existing Office object to copy from.

Creates a new Office instance as a copy of the provided object.

4.66.2.3 ~Office()

```
Office::~Office ()
```

Destructor for the Office class.

Ensures proper cleanup of resources when an Office object is destroyed.

4.66.3 Member Function Documentation

4.66.3.1 clone()

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

Clones the office entity.

Returns a deep copy of the current Office object.

Returns

A pointer to the newly cloned Office entity.

Implements EconomicBuilding.

4.66.3.2 update()

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

Updates the state of the office building entity.

This function handles changes in the office's state.

Implements EconomicBuilding.

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

- src/entities/building/economic/Office.h
- src/entities/building/economic/Office.cpp

4.67 Option Struct Reference

Represents a menu option with a custom key (char or int), icon, and text.

```
#include <IMenu.h>
```

Public Attributes

- std::variant< char, int > **key**

  Custom key can be either char or int.
- std::string **icon**

  Icon representing this option.
- std::string **text**

  The display text for the option.

4.67.1 Detailed Description

Represents a menu option with a custom key (char or int), icon, and text.

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

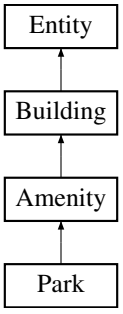
- src/menus/base/IMenu.h

4.68 Park Class Reference

Represents a park entity within the game.

```
#include <Park.h>
```

Inheritance diagram for Park:



Public Member Functions

- Park (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a Park with specified attributes.
- Park (Park *park)

  Copy constructor for the Park class.
- virtual ~**Park** ()

  Destructor for the Park class.
- void update ()

  Updates the park's state.
- Entity * clone ()

  Creates a clone of the park.

Public Member Functions inherited from Amenity

- Amenity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Amenity with specified attributes.
- Amenity (Amenity *amenity)

  Copy constructor for the Amenity class.
- virtual ~**Amenity** ()

  Virtual destructor for the Amenity class.

Public Member Functions inherited from Building

- Building (EntityConfig ec, Size size, int xPos, int yPos)

  Parameterized constructor for the Building class.
- Building (Building *building)

  Copy constructor for the Building class.
- virtual ~Building ()

  Destructor for the Building class.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.

- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

Height of the entity.

- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).
- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.68.1 Detailed Description

Represents a park entity within the game.

The Park class is a specific type of Amenity that provides unique attributes and behaviors specific to parks, such as improving nearby area aesthetics. This class implements the methods to update its state and clone itself.

4.68.2 Constructor & Destructor Documentation

4.68.2.1 Park() [1/2]

```
Park::Park (
            EntityConfig ec,
            Size size,
            int xPos,
            int yPos)
```

Constructs a Park with specified attributes.

Parameters

ec	Configuration containing resource consumption and properties.
size	Size of the park.
xPos	X-coordinate position of the park.
yPos	Y-coordinate position of the park.

4.68.2.2 Park() [2/2]

```
Park::Park (
            Park * park)
```

Copy constructor for the Park class.

Creates a new Park by copying the attributes of an existing Park.

Parameters

park	Pointer to the Park object to be copied.

4.68.3 Member Function Documentation

4.68.3.1 clone()

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

Creates a clone of the park.

Returns

A pointer to the cloned Park.

Implements Amenity.

4.68.3.2 update()

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

Updates the park's state.

Implements Amenity.

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

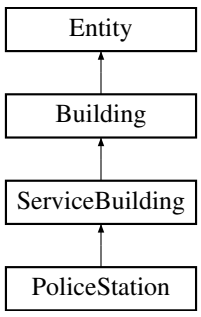
- src/entities/building/amenity/Park.h
- src/entities/building/amenity/Park.cpp

4.69 PoliceStation Class Reference

Class representing a police station in the city.

```
#include <PoliceStation.h>
```

Inheritance diagram for PoliceStation:



Public Member Functions

- PoliceStation (EntityConfig ec, Size size, int xPos, int yPos)

  Parameterized constructor for the PoliceStation class.
- PoliceStation (PoliceStation *police)

  Copy constructor for the PoliceStation class.
- ~PoliceStation ()

  Destructor for the PoliceStation class.
- void update ()

  Updates the state of the police station entity.
- Entity * clone ()

  Clones the police station entity.

Public Member Functions inherited from ServiceBuilding

- ServiceBuilding (EntityConfig ec, Size size, int xPos, int yPos)

  Parameterized constructor for the ServiceBuilding class.
- ServiceBuilding (ServiceBuilding *service)

  Copy constructor for the ServiceBuilding class.
- virtual ~ServiceBuilding ()

  Destructor for the ServiceBuilding class.

Public Member Functions inherited from Building

- Building (EntityConfig ec, Size size, int xPos, int yPos)

  Parameterized constructor for the Building class.
- Building (Building *building)

  Copy constructor for the Building class.
- virtual ~Building ()

  Destructor for the Building class.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.

- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

Height of the entity.

- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).
- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.69.1 Detailed Description

Class representing a police station in the city.

The PoliceStation class provides law enforcement services. It inherits from the ServiceBuilding class.

4.69.2 Constructor & Destructor Documentation

4.69.2.1 PoliceStation() [1/2]

```
PoliceStation::PoliceStation (
            EntityConfig ec,
            Size size,
            int xPos,
            int yPos)
```

Parameterized constructor for the PoliceStation class.

Parameters

ec	Entity configuration for initializing the police station.
size	The size of the police station.
xPos	The x-coordinate of the police station's location.
yPos	The y-coordinate of the police station's location.

4.69.2.2 PoliceStation() [2/2]

```
PoliceStation::PoliceStation (
            PoliceStation * police)
```

Copy constructor for the PoliceStation class.

Parameters

police	A pointer to an existing PoliceStation object to copy from.

4.69.2.3 ~PoliceStation()

```
PoliceStation::~PoliceStation ()
```

Destructor for the PoliceStation class.

Cleans up resources used by the PoliceStation object.

4.69.3 Member Function Documentation

4.69.3.1 clone()

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

Clones the police station entity.

Returns

A pointer to a deep copy of the PoliceStation object.

Implements ServiceBuilding.

4.69.3.2 update()

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

Updates the state of the police station entity.

Handles changes in the police station's state.

Implements ServiceBuilding.

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

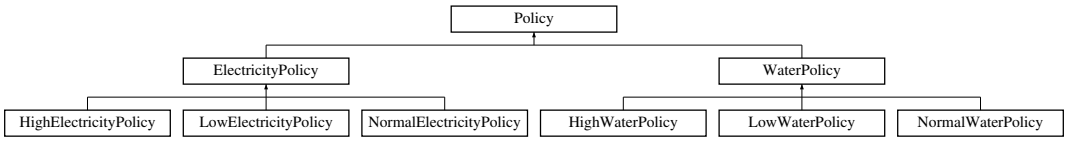
- src/entities/building/service/PoliceStation.h
- src/entities/building/service/PoliceStation.cpp

4.70 Policy Class Reference

Class representing a Policy.

`#include <Policy.h>`

Inheritance diagram for Policy:



Public Member Functions

- Policy (const std::string &name, const std::string &detail)

  Constructor for Policy.
- Memento * createMemento () const

  Creates a memento to store the current state of the policy.
- void setMemento (const Memento *memento)

  Sets the policy state from a memento.
- std::string getName () const

  Gets the name of the policy.
- std::string getDetail () const

  Gets the detail of the policy.

4.70.1 Detailed Description

Class representing a Policy.

Can create a Memento to save its state and restore from it.

4.70.2 Constructor & Destructor Documentation

4.70.2.1 Policy()

```
Policy::Policy (
            const std::string & name,
            const std::string & detail)
```

Constructor for Policy.

Parameters

name	Name of the policy.
detail	Detail of the policy.

4.70.3 Member Function Documentation

4.70.3.1 createMemento()

```
Memento * Policy::createMemento () const
```

Creates a memento to store the current state of the policy.

Returns

Memento* Pointer to a new Memento object representing the current state.

This method captures the current state (name and detail) of the policy and returns a Memento object containing this state.

Returns

Memento* A pointer to a new Memento object containing the current state.

4.70.3.2 getDetail()

```
std::string Policy::getDetail () const
```

Gets the detail of the policy.

Returns

std::string Detail of the policy.

4.70.3.3 getName()

```
std::string Policy::getName () const
```

Gets the name of the policy.

Returns

std::string Name of the policy.

4.70.3.4 setMemento()

```
void Policy::setMemento (
            const Memento * memento)
```

Sets the policy state from a memento.

Restores the policy state from a provided memento.

Parameters

memento	Pointer to the memento from which to restore state.

This method sets the policy's name and detail based on the memento's state.

Parameters

memento	A pointer to the Memento object containing the saved state.

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

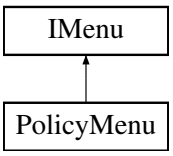
- src/policies/base/Policy.h
- src/policies/base/Policy.cpp

4.71 PolicyMenu Class Reference

Provides functionality for players to apply and review city policies.

`#include <PolicyMenu.h>`

Inheritance diagram for PolicyMenu:



Public Member Functions

- PolicyMenu ()
  Constructs the PolicyMenu.
- ~PolicyMenu ()
  Destructor for PolicyMenu.
- void display () const override
  Displays the Policy Menu.
- void handleInput () override
  Handles user input in the Policy Menu.

Public Member Functions inherited from IMenu

- **IMenu** ()=default
  Default constructor for IMenu.
- IMenu (std::string heading)
  Constructor to initialize a menu with a specified heading.
- virtual ~**IMenu** ()=default
  Virtual destructor for IMenu.
- void setHeading (const std::string &heading)
  Sets the heading of the menu.

Additional Inherited Members

Protected Member Functions inherited from IMenu

- std::string repeat (const std::string &str, int times) const

  Utility function to repeat a string multiple times.
- int calculateMaxWidth (const std::string &menuHeading, const std::vector< Section > &sections) const

  Calculates the maximum width required for the menu.
- void printTopBorder (int width) const

  Prints the top border of the menu using box-drawing characters.
- void printBottomBorder (int width) const

  Prints the bottom border of the menu using box-drawing characters.
- void printSectionDivider (int width) const

  Prints a section divider in the menu using box-drawing characters.
- void printDoubleLineDivider (int width) const

  Prints a double-line divider for the main heading of the menu.
- std::string centerText (const std::string &text, int width) const

  Centers text within a specified width using space padding.
- std::string centerTextWithChar (const std::string &text, int width, const std::string &padChar) const

  Centers text within a specified width using a custom character for padding.
- void displayMenu () const

  Displays the formatted menu, including sections and options.
- void **displayChoicePrompt** () const

  Displays the choice prompt for user input.
- void displayChoiceMessagePrompt (const std::string &message) const

  Displays a custom message prompt for user input.
- void **displayInvalidChoice** () const

  Displays an error message when the user makes an invalid choice.
- void displayErrorMessage (const std::string &message) const

  Displays a general error message.
- void displaySuccessMessage (const std::string &message) const

  Displays a success message in green color.
- void **displayPressEnterToContinue** () const

  Displays a message asking the user to press Enter to continue.
- void **clearScreen** () const

  Clears the terminal screen.
- std::string stripColorCodes (const std::string &input) const

  Strips ANSI color codes from a string.
- virtual void displayAvailablePositions (const std::vector< std::vector< int > > &positions) const

  Displays available positions on the city grid for an entity.

Static Protected Member Functions inherited from IMenu

- static char indexToExtendedChar (int index)

  Converts a numeric index (0-99) to a single character in an extended set.
- static std::string coordinatesToLabel (int x, int y)

  Converts x and y coordinates to a labeled string (e.g., "A, 1").

Protected Attributes inherited from IMenu

- std::vector< Section > **sections**

  List of sections contained in the menu.
- std::string **menuHeading**

  The heading/title of the menu.
- bool **hasExited**

  Flag indicating if the menu has been exited.
- CityManager **cityManager**

  Manager for city-related operations.
- bool **displayResources**

  Flag indicating whether to display resources in the menu.
- bool **isInfoMenu**

  Flag indicating whether to display option numbers.

Static Protected Attributes inherited from IMenu

- static const char * **RESET** = "\033[0m"

  ANSI color codes and styles for use in all menus.
- static const char * **BOLD_WHITE** = "\033[1;37m"
- static const char * **NORMAL_WHITE** = "\033[0;37m"
- static const char * **DARK_GRAY** = "\033[1;30m"
- static const char * **BOLD_YELLOW** = "\033[1;33m"
- static const char * **BOLD_GREEN** = "\033[1;32m"
- static const char * **BOLD_RED** = "\033[1;31m"
- static const char * **BOLD_CYAN** = "\033[1;36m"
- static const char * **BLUE** = "\033[34m"

4.71.1 Detailed Description

Provides functionality for players to apply and review city policies.

The PolicyMenu class allows players to choose new water and electricity policies, and view the history of all previously applied policies.

4.71.2 Constructor & Destructor Documentation

4.71.2.1 PolicyMenu()

```
PolicyMenu::PolicyMenu ()
```

Constructs the PolicyMenu.

Initializes the PolicyMenu instance with options for applying new policies and viewing history.

4.71.2.2 ~PolicyMenu()

```
PolicyMenu::~PolicyMenu ()
```

Destructor for PolicyMenu.

Cleans up resources used by the PolicyMenu.

4.71.3 Member Function Documentation

4.71.3.1 display()

```
void PolicyMenu::display () const  [override], [virtual]
```

Displays the Policy Menu.

Overrides the display method of IMenu to render the policy options.

Implements IMenu.

4.71.3.2 handleInput()

```
void PolicyMenu::handleInput ()  [override], [virtual]
```

Handles user input in the Policy Menu.

Processes input to navigate between policy options and apply new policies.

Implements IMenu.

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

- src/menus/policy/PolicyMenu.h
- src/menus/policy/PolicyMenu.cpp

4.72 PopulationManager Class Reference

Responsible for managing the population growth, decrease, capacity calculations, and satisfaction levels within a City.

```
#include <PopulationManager.h>
```

Public Member Functions

- PopulationManager (int minimumIncrease, int maximumIncrease)

  Constructs a PopulationManager with specified growth parameters.
- ~**PopulationManager** ()

  Destructor for PopulationManager.
- void **calculatePopulationCapacity** ()

  Calculates the population capacity of the City and sets it.
- void **growPopulation** ()

  Increases the City's population by a random value within the specified minimum and maximum range.
- void **decreasePopulation** ()

  Decreases the City's population by a random value within the specified minimum and maximum range.
- void **calculateSatisfaction** ()

  Calculates the satisfaction level of the City based on available resources and utilities, updating the City's satisfaction.

4.72.1 Detailed Description

Responsible for managing the population growth, decrease, capacity calculations, and satisfaction levels within a City.

4.72.2 Constructor & Destructor Documentation

4.72.2.1 PopulationManager()

```
PopulationManager::PopulationManager (
            int minimumIncrease,
            int maximumIncrease)
```

Constructs a PopulationManager with specified growth parameters.

Parameters

minimumIncrease	Minimum population increase per cycle.
maximumIncrease	Maximum population increase per cycle.

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

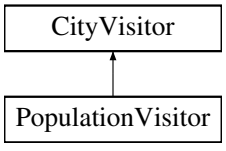
- src/managers/PopulationManager.h
- src/managers/PopulationManager.cpp

4.73 PopulationVisitor Class Reference

Visitor that calculates population and resource capacities in a city.

```
#include <PopulationVisitor.h>
```

Inheritance diagram for PopulationVisitor:



Public Member Functions

- **PopulationVisitor** ()
  Constructs a PopulationVisitor with zeroed capacity and consumption values.
- ~**PopulationVisitor** ()
  Default destructor.
- void visit (City *city) override
  Visits a city to calculate population and resource data.
- int getTotalPopulationCapacity () const
  Gets the total population capacity.
- int getHousePopulationCapacity () const
  Gets the total population capacity for houses.
- int getTotalWaterConsumption () const
  Gets the total water consumption.
- int getTotalElectricityConsumption () const
  Gets the total electricity consumption.
- int getApartmentPopulationCapacity () const
  Gets the total population capacity for apartments.

Public Member Functions inherited from CityVisitor

- **CityVisitor** ()=default
  Default constructor.
- virtual ~**CityVisitor** ()=default
  Default destructor.

4.73.1 Detailed Description

Visitor that calculates population and resource capacities in a city.

4.73.2 Member Function Documentation

4.73.2.1 getApartmentPopulationCapacity()

```
int PopulationVisitor::getApartmentPopulationCapacity () const
```

Gets the total population capacity for apartments.

Returns

Population capacity specific to apartments.

4.73.2.2 getHousePopulationCapacity()

```
int PopulationVisitor::getHousePopulationCapacity () const
```

Gets the total population capacity for houses.

Returns

Population capacity specific to houses.

4.73.2.3 getTotalElectricityConsumption()

```
int PopulationVisitor::getTotalElectricityConsumption () const
```

Gets the total electricity consumption.

Returns

Total electricity consumption in the city.

4.73.2.4 getTotalPopulationCapacity()

```
int PopulationVisitor::getTotalPopulationCapacity () const
```

Gets the total population capacity.

Returns

Total population capacity of all buildings.

4.73.2.5 getTotalWaterConsumption()

```
int PopulationVisitor::getTotalWaterConsumption () const
```

Gets the total water consumption.

Returns

Total water consumption in the city.

4.73.2.6 visit()

```
void PopulationVisitor::visit (
            City * city)  [override], [virtual]
```

Visits a city to calculate population and resource data.

Parameters

city	Pointer to the City object being visited.

Implements CityVisitor.

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

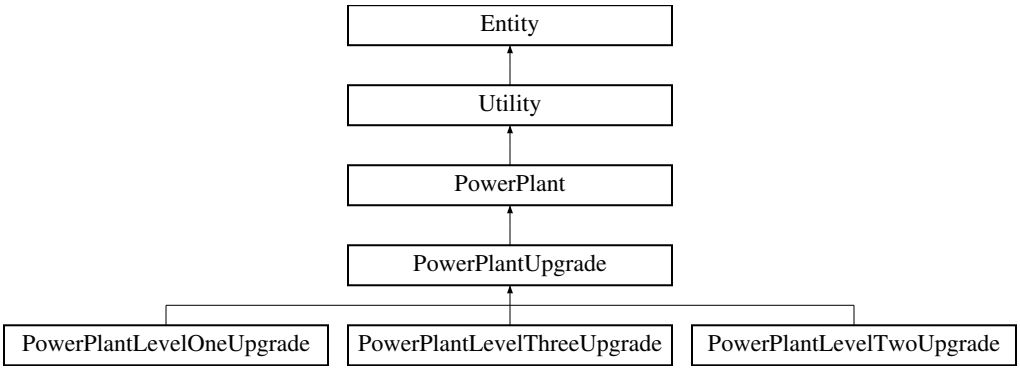
- src/visitors/population/PopulationVisitor.h
- src/visitors/population/PopulationVisitor.cpp

4.74 PowerPlant Class Reference

Represents a power plant in the city builder simulation.

`#include <PowerPlant.h>`

Inheritance diagram for PowerPlant:



Public Member Functions

- PowerPlant (EntityConfig ec, Size size, int xPos, int yPos)
  Constructs a PowerPlant object with specified attributes.
- PowerPlant (PowerPlant *power)
  Copy constructor for the PowerPlant class.
- virtual ~**PowerPlant** ()
  Destructor for the PowerPlant object.
- void update () override
  Updates the state of the power plant.
- Entity * clone () override
  Clones the current PowerPlant object.
- Entity * upgrade () override
  Upgrades the current utility to the next level.

Public Member Functions inherited from Utility

- Utility (EntityConfig ec, Size size, int xPos, int yPos)
  Constructs a Utility object with the specified parameters.
- Utility (Utility *utility)
  Copy constructor for the Utility class.
- virtual ~**Utility** ()
  Destructor for the Utility object.
- virtual int getOutput ()
  Retrieves the output of the utility.
- void setOutput (int output)
  Sets the output value of the utility.
- virtual Cost getCost ()
  Retrieves the cost of the utility or its upgraded version.
- virtual int getLevel ()
  Gets the level of the utility.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from Entity

- std::string **symbol**
  Symbol representing the entity.
- int **effectRadius**
  Radius of effect for this entity.
- int **localEffectStrength**
  Local effect strength of the entity.
- int **globalEffectStrength**
  Global effect strength of the entity.
- int **width**
  Width of the entity.
- int **height**
  Height of the entity.
- int **xPosition**
  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**
  Y-coordinate of the entity's position (bottom left corner).
- Size **size**
  Size object representing the entity's dimensions.
- EntityType **type**
  The type of entity.
- State * **state**
  Pointer to the current state of the entity.
- int **revenue**
  Revenue generated by the entity.
- float **electricityConsumption**
  Electricity consumption of the entity.
- float **waterConsumption**
  Water consumption of the entity.
- std::vector< Entity * > **observers**
  List of other entities observing this entity.

4.74.1 Detailed Description

Represents a power plant in the city builder simulation.

The PowerPlant class is a specialized type of Utility that produces electricity.

4.74.2 Constructor & Destructor Documentation

4.74.2.1 PowerPlant() [1/2]

```
PowerPlant::PowerPlant (
            EntityConfig ec,
            Size size,
            int xPos,
            int yPos)
```

Constructs a PowerPlant object with specified attributes.

Initializes a PowerPlant with detailed parameters.

Parameters

ec	EntityConfig.
size	Size.
xPos	xPosition
yPos	yPosition

4.74.2.2 PowerPlant() [2/2]

```
PowerPlant::PowerPlant (
            PowerPlant * power)
```

Copy constructor for the PowerPlant class.

Creates a new PowerPlant object by copying the attributes of an existing PowerPlant.

Parameters

power	Pointer to the existing PowerPlant object to be copied.

4.74.3 Member Function Documentation

4.74.3.1 clone()

```
Entity * PowerPlant::clone ()  [override], [virtual]
```

Clones the current PowerPlant object.

Creates and returns a copy of the current PowerPlant instance.

Returns

A pointer to the newly cloned PowerPlant object.

Implements Utility.

Reimplemented in PowerPlantLevelOneUpgrade, PowerPlantLevelThreeUpgrade, PowerPlantLevelTwoUpgrade, and PowerPlantUpgrade.

4.74.3.2 update()

```
void PowerPlant::update ()  [override], [virtual]
```

Updates the state of the power plant.

Defines the specific behavior of the PowerPlant when it is updated in the simulation.

Implements Utility.

Reimplemented in PowerPlantLevelOneUpgrade, PowerPlantLevelThreeUpgrade, PowerPlantLevelTwoUpgrade, and PowerPlantUpgrade.

4.74.3.3 upgrade()

```
Entity * PowerPlant::upgrade ()  [override], [virtual]
```

Upgrades the current utility to the next level.

Returns

A pointer to the upgraded utility instance, or nullptr if already at maximum level.

Implements Utility.

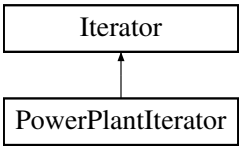
Reimplemented in PowerPlantLevelOneUpgrade, PowerPlantLevelThreeUpgrade, PowerPlantLevelTwoUpgrade, and PowerPlantUpgrade.

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

- src/entities/utility/powerplant/PowerPlant.h
- src/entities/utility/powerplant/PowerPlant.cpp

4.75 PowerPlantIterator Class Reference

Inheritance diagram for PowerPlantIterator:



Public Member Functions

- **PowerPlantIterator** ()

  Construct a new PowerPlantIterator object.
- ~**PowerPlantIterator** ()

  Destroy the PowerPlantIterator object.
- PowerPlantIterator (std::vector< std::vector< Entity * > > &grid)

  Construct a new PowerPlantIterator object with grid.
- void first ()

  Sets the iterator to the first unvisited PowerPlant.
- void next ()

  Advances to the next unvisited PowerPlant.
- bool hasNext ()

  Checks if there is another unvisited PowerPlant.
- Entity * current ()

  Returns the current PowerPlant.

Public Member Functions inherited from Iterator

- **Iterator** ()

  Construct a new Iterator object, initializing row and column to zero.
- virtual ~**Iterator** ()

  Destroy the Iterator object.
- **Iterator** (std::vector< std::vector< Entity * > > &grid)
- virtual int getRow ()

  Get the current row index of the iterator.
- virtual int getCol ()

  Get the current column index of the iterator.

Additional Inherited Members

Protected Member Functions inherited from Iterator

- bool isVisited (Entity *entity)

  Check if the specified entity has been visited.
- void markVisited (Entity *entity)

  Mark the specified entity as visited.

Protected Attributes inherited from Iterator

- std::vector< std::vector< Entity * > > **grid**
- std::vector< std::vector< Entity * > >::iterator **currRow**
- std::vector< Entity * >::iterator **curr**
- int **row**
- int **col**
- std::unordered_set< Entity * > **visitedEntities**

4.75.1 Constructor & Destructor Documentation

4.75.1.1 PowerPlantIterator()

```
PowerPlantIterator::PowerPlantIterator (
            std::vector< std::vector< Entity * > > & grid)
```

Construct a new PowerPlantIterator object with grid.

Parameters

grid	

4.75.2 Member Function Documentation

4.75.2.1 current()

```
Entity * PowerPlantIterator::current ()  [virtual]
```

Returns the current PowerPlant.

Returns

Entity*

Implements Iterator.

4.75.2.2 first()

```
void PowerPlantIterator::first ()  [virtual]
```

Sets the iterator to the first unvisited PowerPlant.

Implements Iterator.

4.75.2.3 hasNext()

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

Checks if there is another unvisited PowerPlant.

Returns

true if there is another unvisited PowerPlant, false otherwise

Implements Iterator.

4.75.2.4 next()

```
void PowerPlantIterator::next ()  [virtual]
```

Advances to the next unvisited PowerPlant.

Implements Iterator.

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

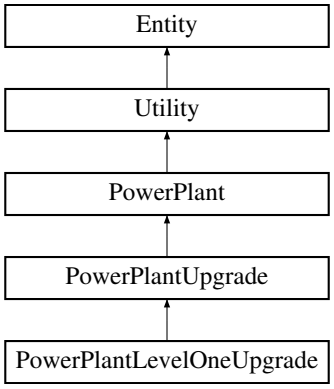
- src/iterators/utility/PowerPlantIterator.h
- src/iterators/utility/PowerPlantIterator.cpp

4.76 PowerPlantLevelOneUpgrade Class Reference

Represents the first level upgrade to a PowerPlant entity.

`#include <PowerPlantLevelOneUpgrade.h>`

Inheritance diagram for PowerPlantLevelOneUpgrade:



Public Member Functions

- PowerPlantLevelOneUpgrade (PowerPlant *power)
  Constructs a PowerPlantLevelOneUpgrade object.
- PowerPlantLevelOneUpgrade (PowerPlantLevelOneUpgrade *pPLOU)
  Copy constructor for PowerPlantLevelOneUpgrade.
- ~PowerPlantLevelOneUpgrade ()
  Destructor for PowerPlantLevelOneUpgrade.
- void update () override
  Updates the state of the upgraded power plant.
- Entity * clone () override
  Clones the current PowerPlantLevelOneUpgrade object.
- Entity * upgrade () override
  Upgrades the current utility to the next level.
- int getOutput () override
  Retrieves the upgraded power plant's output.
- Cost getCost () override
  Retrieves the cost of the utility or its upgraded version.
- int getLevel () override
  Gets the level of the power plant upgrade.

Public Member Functions inherited from PowerPlantUpgrade

- PowerPlantUpgrade (PowerPlant *power)
  Constructs a PowerPlantUpgrade object based on an existing PowerPlant.
- PowerPlantUpgrade (PowerPlantUpgrade *pPU)
  Copy constructor for the PowerPlantUpgrade class.
- virtual ~**PowerPlantUpgrade** ()
  Destructor for the PowerPlantUpgrade object.

Public Member Functions inherited from PowerPlant

- PowerPlant (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a PowerPlant object with specified attributes.
- PowerPlant (PowerPlant *power)

  Copy constructor for the PowerPlant class.
- virtual ~**PowerPlant** ()

  Destructor for the PowerPlant object.

Public Member Functions inherited from Utility

- Utility (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a Utility object with the specified parameters.
- Utility (Utility *utility)

  Copy constructor for the Utility class.
- virtual ~**Utility** ()

  Destructor for the Utility object.
- void setOutput (int output)

  Sets the output value of the utility.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

Sets the symbol of the entity.

- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from PowerPlantUpgrade

- PowerPlant * **powerPlant**

  Pointer to the original PowerPlant that is being upgraded.

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).

- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.76.1 Detailed Description

Represents the first level upgrade to a PowerPlant entity.

The PowerPlantLevelOneUpgrade class enhances the base functionality of a PowerPlant by increasing its output. This class is the first upgrade level in a series of potential power plant improvements.

4.76.2 Constructor & Destructor Documentation

4.76.2.1 PowerPlantLevelOneUpgrade() [1/2]

```
PowerPlantLevelOneUpgrade::PowerPlantLevelOneUpgrade (
            PowerPlant * power)
```

Constructs a PowerPlantLevelOneUpgrade object.

Initializes the upgrade by enhancing the specified PowerPlant with a level one upgrade.

Parameters

power	Pointer to the original PowerPlant to be upgraded.

4.76.2.2 PowerPlantLevelOneUpgrade() [2/2]

```
PowerPlantLevelOneUpgrade::PowerPlantLevelOneUpgrade (
            PowerPlantLevelOneUpgrade * pPLOU)
```

Copy constructor for PowerPlantLevelOneUpgrade.

Creates a new PowerPlantLevelOneUpgrade object by copying the attributes of an existing PowerPlantLevelOneUpgrade object.

Parameters

pPLOU	Pointer to the existing PowerPlantLevelOneUpgrade to be copied.

4.76.2.3 ~PowerPlantLevelOneUpgrade()

```
PowerPlantLevelOneUpgrade::~PowerPlantLevelOneUpgrade ()
```

Destructor for PowerPlantLevelOneUpgrade.

Cleans up any resources associated with the upgrade.

4.76.3 Member Function Documentation

4.76.3.1 clone()

```
Entity * PowerPlantLevelOneUpgrade::clone ()  [override], [virtual]
```

Clones the current PowerPlantLevelOneUpgrade object.

Creates a new instance of PowerPlantLevelOneUpgrade with the same attributes as the current object.

Returns

A pointer to the newly cloned PowerPlantLevelOneUpgrade object.

Implements PowerPlantUpgrade.

4.76.3.2 getCost()

```
Cost PowerPlantLevelOneUpgrade::getCost ()  [override], [virtual]
```

Retrieves the cost of the utility or its upgraded version.

Returns

A Cost object representing the monetary and material costs.

Implements PowerPlantUpgrade.

4.76.3.3 getLevel()

```
int PowerPlantLevelOneUpgrade::getLevel ()  [override], [virtual]
```

Gets the level of the power plant upgrade.

Returns

The level of the power plant upgrade.

Reimplemented from Utility.

4.76.3.4 getOutput()

```
int PowerPlantLevelOneUpgrade::getOutput ()  [override], [virtual]
```

Retrieves the upgraded power plant's output.

Returns the power output of the level one upgraded power plant.

Returns

The updated power output as an integer.

Implements PowerPlantUpgrade.

4.76.3.5 update()

```
void PowerPlantLevelOneUpgrade::update ()  [override], [virtual]
```

Updates the state of the upgraded power plant.

Implements specific behavior for the power plant after applying the level one upgrade.

Implements PowerPlantUpgrade.

4.76.3.6 upgrade()

```
Entity * PowerPlantLevelOneUpgrade::upgrade ()  [override], [virtual]
```

Upgrades the current utility to the next level.

Returns

A pointer to the upgraded utility instance, or nullptr if already at maximum level.

Implements PowerPlantUpgrade.

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

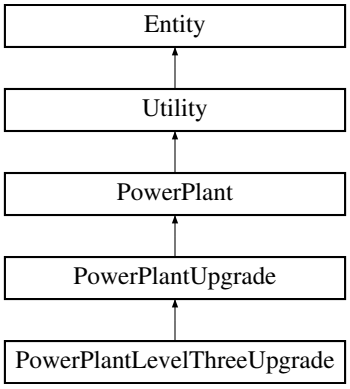
- src/entities/utility/powerplant/PowerPlantLevelOneUpgrade.h
- src/entities/utility/powerplant/PowerPlantLevelOneUpgrade.cpp

4.77 PowerPlantLevelThreeUpgrade Class Reference

Represents the third level upgrade to a PowerPlant entity.

`#include <PowerPlantLevelThreeUpgrade.h>`

Inheritance diagram for PowerPlantLevelThreeUpgrade:



Public Member Functions

- PowerPlantLevelThreeUpgrade (PowerPlant ∗power)

  Constructs a PowerPlantLevelThreeUpgrade object.
- PowerPlantLevelThreeUpgrade (PowerPlantLevelThreeUpgrade ∗pPLTU)

  Copy constructor for PowerPlantLevelThreeUpgrade.
- ∼PowerPlantLevelThreeUpgrade ()

  Destructor for PowerPlantLevelThreeUpgrade.
- void update () override

  Updates the state of the upgraded power plant.
- Entity ∗ clone () override

  Clones the current PowerPlantLevelThreeUpgrade object.
- Entity ∗ upgrade () override

  Upgrades the current utility to the next level.
- int getOutput () override

  Retrieves the upgraded power plant's output.
- Cost getCost () override

  Retrieves the cost of the utility or its upgraded version.
- int getLevel () override

  Gets the level of the power plant upgrade.

Public Member Functions inherited from PowerPlantUpgrade

- PowerPlantUpgrade (PowerPlant ∗power)

  Constructs a PowerPlantUpgrade object based on an existing PowerPlant.
- PowerPlantUpgrade (PowerPlantUpgrade ∗pPU)

  Copy constructor for the PowerPlantUpgrade class.
- virtual ∼**PowerPlantUpgrade** ()

  Destructor for the PowerPlantUpgrade object.

Public Member Functions inherited from PowerPlant

- PowerPlant (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a PowerPlant object with specified attributes.
- PowerPlant (PowerPlant *power)

  Copy constructor for the PowerPlant class.
- virtual ~**PowerPlant** ()

  Destructor for the PowerPlant object.

Public Member Functions inherited from Utility

- Utility (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a Utility object with the specified parameters.
- Utility (Utility *utility)

  Copy constructor for the Utility class.
- virtual ~**Utility** ()

  Destructor for the Utility object.
- void setOutput (int output)

  Sets the output value of the utility.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

Sets the symbol of the entity.

- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from PowerPlantUpgrade

- PowerPlant * **powerPlant**

  Pointer to the original PowerPlant that is being upgraded.

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).

- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.77.1 Detailed Description

Represents the third level upgrade to a PowerPlant entity.

The PowerPlantLevelThreeUpgrade class enhances the base functionality of a PowerPlant by increasing its output. This class is the third upgrade level in a series of potential power plant improvements.

4.77.2 Constructor & Destructor Documentation

4.77.2.1 PowerPlantLevelThreeUpgrade() [1/2]

```
PowerPlantLevelThreeUpgrade::PowerPlantLevelThreeUpgrade (
            PowerPlant * power)
```

Constructs a PowerPlantLevelThreeUpgrade object.

Initializes the upgrade by enhancing the specified PowerPlant with a level three upgrade.

Parameters

power	Pointer to the original PowerPlant to be upgraded.

4.77.2.2 PowerPlantLevelThreeUpgrade() [2/2]

```
PowerPlantLevelThreeUpgrade::PowerPlantLevelThreeUpgrade (
            PowerPlantLevelThreeUpgrade * pPLTU)
```

Copy constructor for PowerPlantLevelThreeUpgrade.

Creates a new PowerPlantLevelThreeUpgrade object by copying the attributes of an existing PowerPlantLevelThreeUpgrade object.

Parameters

pPLTU	Pointer to the existing PowerPlantLevelThreeUpgrade to be copied.

4.77.2.3 ~PowerPlantLevelThreeUpgrade()

```
PowerPlantLevelThreeUpgrade::~PowerPlantLevelThreeUpgrade ()
```

Destructor for PowerPlantLevelThreeUpgrade.

Cleans up any resources associated with the upgrade.

4.77.3 Member Function Documentation

4.77.3.1 clone()

```
Entity * PowerPlantLevelThreeUpgrade::clone ()  [override], [virtual]
```

Clones the current PowerPlantLevelThreeUpgrade object.

Creates a new instance of PowerPlantLevelThreeUpgrade with the same attributes as the current object.

Returns

A pointer to the newly cloned PowerPlantLevelThreeUpgrade object.

Implements PowerPlantUpgrade.

4.77.3.2 getCost()

```
Cost PowerPlantLevelThreeUpgrade::getCost ()  [override], [virtual]
```

Retrieves the cost of the utility or its upgraded version.

Returns

A Cost object representing the monetary and material costs.

Implements PowerPlantUpgrade.

4.77.3.3 getLevel()

```
int PowerPlantLevelThreeUpgrade::getLevel ()  [override], [virtual]
```

Gets the level of the power plant upgrade.

Returns

The level of the power plant upgrade.

Reimplemented from Utility.

4.77.3.4 getOutput()

```
int PowerPlantLevelThreeUpgrade::getOutput ()  [override], [virtual]
```

Retrieves the upgraded power plant's output.

Returns the power output of the level three upgraded power plant.

Returns

The updated power output as an integer.

Implements PowerPlantUpgrade.

4.77.3.5 update()

```
void PowerPlantLevelThreeUpgrade::update ()  [override], [virtual]
```

Updates the state of the upgraded power plant.

Implements specific behavior for the power plant after applying the level three upgrade.

Implements PowerPlantUpgrade.

4.77.3.6 upgrade()

```
Entity * PowerPlantLevelThreeUpgrade::upgrade ()  [override], [virtual]
```

Upgrades the current utility to the next level.

Returns

A pointer to the upgraded utility instance, or nullptr if already at maximum level.

Implements PowerPlantUpgrade.

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

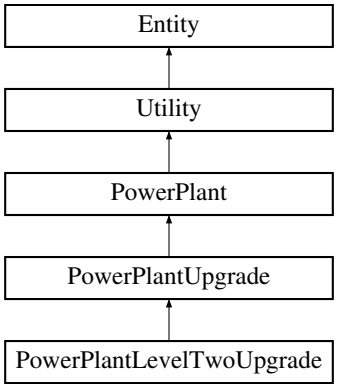
- src/entities/utility/powerplant/PowerPlantLevelThreeUpgrade.h
- src/entities/utility/powerplant/PowerPlantLevelThreeUpgrade.cpp

4.78 PowerPlantLevelTwoUpgrade Class Reference

Represents the second level upgrade to a PowerPlant entity.

`#include <PowerPlantLevelTwoUpgrade.h>`

Inheritance diagram for PowerPlantLevelTwoUpgrade:



Public Member Functions

- PowerPlantLevelTwoUpgrade (PowerPlant ∗power)

  Constructs a PowerPlantLevelTwoUpgrade object.
- PowerPlantLevelTwoUpgrade (PowerPlantLevelTwoUpgrade ∗pPLTU)

  Copy constructor for PowerPlantLevelTwoUpgrade.
- ∼PowerPlantLevelTwoUpgrade ()

  Destructor for PowerPlantLevelTwoUpgrade.
- void update () override

  Updates the state of the upgraded power plant.
- Entity ∗ clone () override

  Clones the current PowerPlantLevelTwoUpgrade object.
- Entity ∗ upgrade () override

  Upgrades the current utility to the next level.
- int getOutput () override

  Retrieves the upgraded power plant's output.
- Cost getCost () override

  Retrieves the cost of the utility or its upgraded version.
- int getLevel () override

  Gets the level of the power plant upgrade.

Public Member Functions inherited from PowerPlantUpgrade

- PowerPlantUpgrade (PowerPlant ∗power)

  Constructs a PowerPlantUpgrade object based on an existing PowerPlant.
- PowerPlantUpgrade (PowerPlantUpgrade ∗pPU)

  Copy constructor for the PowerPlantUpgrade class.
- virtual ∼**PowerPlantUpgrade** ()

  Destructor for the PowerPlantUpgrade object.

Public Member Functions inherited from PowerPlant

- PowerPlant (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a PowerPlant object with specified attributes.
- PowerPlant (PowerPlant *power)

  Copy constructor for the PowerPlant class.
- virtual ~**PowerPlant** ()

  Destructor for the PowerPlant object.

Public Member Functions inherited from Utility

- Utility (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a Utility object with the specified parameters.
- Utility (Utility *utility)

  Copy constructor for the Utility class.
- virtual ~**Utility** ()

  Destructor for the Utility object.
- void setOutput (int output)

  Sets the output value of the utility.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

Sets the symbol of the entity.

- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from PowerPlantUpgrade

- PowerPlant * **powerPlant**

  Pointer to the original PowerPlant that is being upgraded.

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).

- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.78.1 Detailed Description

Represents the second level upgrade to a PowerPlant entity.

The PowerPlantLevelTwoUpgrade class enhances the base functionality of a PowerPlant by increasing its output. This class is the second upgrade level in a series of potential power plant improvements.

4.78.2 Constructor & Destructor Documentation

4.78.2.1 PowerPlantLevelTwoUpgrade() [1/2]

```
PowerPlantLevelTwoUpgrade::PowerPlantLevelTwoUpgrade (
            PowerPlant * power)
```

Constructs a PowerPlantLevelTwoUpgrade object.

Initializes the upgrade by enhancing the specified PowerPlant with a level two upgrade.

Parameters

power	Pointer to the original PowerPlant to be upgraded.

4.78.2.2 PowerPlantLevelTwoUpgrade() [2/2]

```
PowerPlantLevelTwoUpgrade::PowerPlantLevelTwoUpgrade (
            PowerPlantLevelTwoUpgrade * pPLTU)
```

Copy constructor for PowerPlantLevelTwoUpgrade.

Creates a new PowerPlantLevelTwoUpgrade object by copying the attributes of an existing PowerPlantLevelTwoUpgrade object.

Parameters

pPLTU	Pointer to the existing PowerPlantLevelTwoUpgrade to be copied.

4.78.2.3 ~PowerPlantLevelTwoUpgrade()

```
PowerPlantLevelTwoUpgrade::~PowerPlantLevelTwoUpgrade ()
```

Destructor for PowerPlantLevelTwoUpgrade.

Cleans up any resources associated with the upgrade.

4.78.3 Member Function Documentation

4.78.3.1 clone()

```
Entity * PowerPlantLevelTwoUpgrade::clone ()  [override], [virtual]
```

Clones the current PowerPlantLevelTwoUpgrade object.

Creates a new instance of PowerPlantLevelTwoUpgrade with the same attributes as the current object.

Returns

A pointer to the newly cloned PowerPlantLevelTwoUpgrade object.

Implements PowerPlantUpgrade.

4.78.3.2 getCost()

```
Cost PowerPlantLevelTwoUpgrade::getCost ()  [override], [virtual]
```

Retrieves the cost of the utility or its upgraded version.

Returns

A Cost object representing the monetary and material costs.

Implements PowerPlantUpgrade.

4.78.3.3 getLevel()

```
int PowerPlantLevelTwoUpgrade::getLevel ()  [override], [virtual]
```

Gets the level of the power plant upgrade.

Returns

The level of the power plant upgrade.

Reimplemented from Utility.

4.78.3.4 getOutput()

```
int PowerPlantLevelTwoUpgrade::getOutput ()  [override], [virtual]
```

Retrieves the upgraded power plant's output.

Returns the power output of the level two upgraded power plant.

Returns

The updated power output as an integer.

Implements PowerPlantUpgrade.

4.78.3.5 update()

```
void PowerPlantLevelTwoUpgrade::update ()  [override], [virtual]
```

Updates the state of the upgraded power plant.

Implements specific behavior for the power plant after applying the level two upgrade.

Implements PowerPlantUpgrade.

4.78.3.6 upgrade()

```
Entity * PowerPlantLevelTwoUpgrade::upgrade ()  [override], [virtual]
```

Upgrades the current utility to the next level.

Returns

A pointer to the upgraded utility instance, or nullptr if already at maximum level.

Implements PowerPlantUpgrade.

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

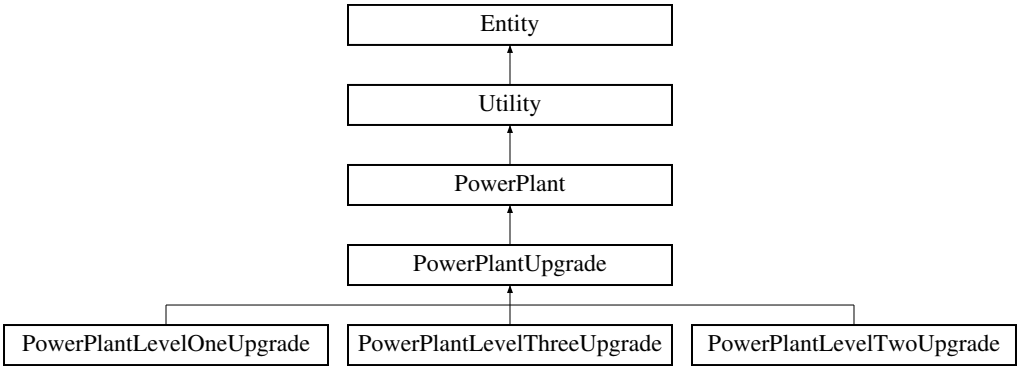
- src/entities/utility/powerplant/PowerPlantLevelTwoUpgrade.h
- src/entities/utility/powerplant/PowerPlantLevelTwoUpgrade.cpp

4.79 PowerPlantUpgrade Class Reference

Represents an upgrade to a PowerPlant entity in the city builder simulation.

`#include <PowerPlantUpgrade.h>`

Inheritance diagram for PowerPlantUpgrade:



Public Member Functions

- PowerPlantUpgrade (PowerPlant ∗power)

  Constructs a PowerPlantUpgrade object based on an existing PowerPlant.
- PowerPlantUpgrade (PowerPlantUpgrade ∗pPU)

  Copy constructor for the PowerPlantUpgrade class.
- virtual ∼**PowerPlantUpgrade** ()

  Destructor for the PowerPlantUpgrade object.
- virtual void update ()=0

  Pure virtual function to update the upgraded power plant.
- virtual Entity ∗ clone ()=0

  Pure virtual function to clone the upgraded power plant.
- virtual Entity ∗ upgrade ()=0

  Upgrades the current utility to the next level.
- virtual int getOutput ()=0

  Retrieves the output of the upgraded power plant.
- virtual Cost getCost ()=0

  Retrieves the cost of the utility or its upgraded version.

Public Member Functions inherited from PowerPlant

- PowerPlant (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a PowerPlant object with specified attributes.
- PowerPlant (PowerPlant ∗power)

  Copy constructor for the PowerPlant class.
- virtual ∼**PowerPlant** ()

  Destructor for the PowerPlant object.

Public Member Functions inherited from Utility

- Utility (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a Utility object with the specified parameters.
- Utility (Utility *utility)

  Copy constructor for the Utility class.
- virtual ~**Utility** ()

  Destructor for the Utility object.
- void setOutput (int output)

  Sets the output value of the utility.
- virtual int getLevel ()

  Gets the level of the utility.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

Unsubscribes this entity from all buildings it is observing.

- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Protected Attributes

- PowerPlant * **powerPlant**

  Pointer to the original PowerPlant that is being upgraded.

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).
- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.79.1 Detailed Description

Represents an upgrade to a PowerPlant entity in the city builder simulation.

The PowerPlantUpgrade class extends the functionality of a PowerPlant, enhancing its capabilities and acting as a wrapper around the existing PowerPlant object.

4.79.2 Constructor & Destructor Documentation

4.79.2.1 PowerPlantUpgrade() [1/2]

```
PowerPlantUpgrade::PowerPlantUpgrade (
            PowerPlant * power)
```

Constructs a PowerPlantUpgrade object based on an existing PowerPlant.

Initializes the upgrade with a reference to an existing PowerPlant, enhancing its features.

Parameters

power	Pointer to the PowerPlant being upgraded.

4.79.2.2 PowerPlantUpgrade() [2/2]

```
PowerPlantUpgrade::PowerPlantUpgrade (
            PowerPlantUpgrade * pPU)
```

Copy constructor for the PowerPlantUpgrade class.

Creates a new PowerPlantUpgrade object by copying the attributes of an existing PowerPlantUpgrade.

Parameters

pPU	Pointer to the existing PowerPlantUpgrade object to be copied.

4.79.3 Member Function Documentation

4.79.3.1 clone()

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

Pure virtual function to clone the upgraded power plant.

This method allows cloning of the upgraded power plant, creating a new instance with the same attributes.

Returns

A pointer to a new cloned PowerPlantUpgrade object.

Reimplemented from PowerPlant.

Implemented in PowerPlantLevelOneUpgrade, PowerPlantLevelThreeUpgrade, and PowerPlantLevelTwoUpgrade.

4.79.3.2 getCost()

```
virtual Cost PowerPlantUpgrade::getCost ()  [pure virtual]
```

Retrieves the cost of the utility or its upgraded version.

Returns

A Cost object representing the monetary and material costs.

Reimplemented from Utility.

Implemented in PowerPlantLevelOneUpgrade, PowerPlantLevelThreeUpgrade, and PowerPlantLevelTwoUpgrade.

4.79.3.3 getOutput()

```
virtual int PowerPlantUpgrade::getOutput ()  [pure virtual]
```

Retrieves the output of the upgraded power plant.

Returns

The power output as an integer.

Reimplemented from Utility.

Implemented in PowerPlantLevelOneUpgrade, PowerPlantLevelThreeUpgrade, and PowerPlantLevelTwoUpgrade.

4.79.3.4 update()

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

Pure virtual function to update the upgraded power plant.

This method defines the specific behavior of the power plant after the upgrade when it is updated.

Reimplemented from PowerPlant.

Implemented in PowerPlantLevelOneUpgrade, PowerPlantLevelThreeUpgrade, and PowerPlantLevelTwoUpgrade.

4.79.3.5 upgrade()

```
virtual Entity * PowerPlantUpgrade::upgrade ()  [pure virtual]
```

Upgrades the current utility to the next level.

Returns

A pointer to the upgraded utility instance, or nullptr if already at maximum level.

Reimplemented from PowerPlant.

Implemented in PowerPlantLevelOneUpgrade, PowerPlantLevelThreeUpgrade, and PowerPlantLevelTwoUpgrade.

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

- src/entities/utility/powerplant/PowerPlantUpgrade.h
- src/entities/utility/powerplant/PowerPlantUpgrade.cpp

4.80 Rectangle Class Reference

Represents a rectangular area with position and size.

`#include <BSPPartitioner.h>`

Public Member Functions

- Rectangle (int x, int y, int width, int height)

  Constructs a Rectangle with specified position and size.

Public Attributes

- int **x**

  The x-coordinate of the rectangle.
- int **y**

  The y-coordinate of the rectangle.
- int **width**

  The width of the rectangle.
- int **height**

  The height of the rectangle.

4.80.1 Detailed Description

Represents a rectangular area with position and size.

4.80.2 Constructor & Destructor Documentation

4.80.2.1 Rectangle()

```
Rectangle::Rectangle (
            int x,
            int y,
            int width,
            int height)  [inline]
```

Constructs a Rectangle with specified position and size.

Parameters

x	The x-coordinate.
y	The y-coordinate.
width	The width of the rectangle.
height	The height of the rectangle.

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

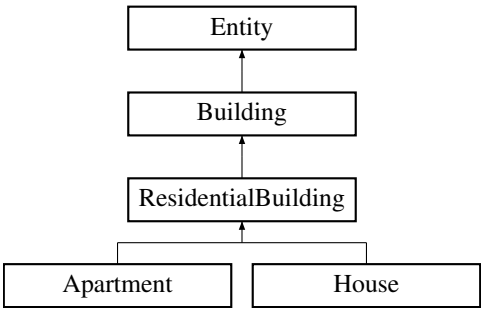
- src/utils/BSPPartitioner.h

4.81 ResidentialBuilding Class Reference

Represents a residential building entity within the game.

`#include <ResidentialBuilding.h>`

Inheritance diagram for ResidentialBuilding:



Public Member Functions

- ResidentialBuilding (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a ResidentialBuilding with specified attributes.
- ResidentialBuilding (ResidentialBuilding ∗entity)

  Copy constructor for the ResidentialBuilding class.
- virtual ∼**ResidentialBuilding** ()

  Destructor for the ResidentialBuilding class.
- void update ()

  Updates the residential building's state.
- virtual Entity ∗ clone ()=0

  Creates a clone of the residential building.
- void **reset** ()

  Resets the satisfaction factors for the building.
- void **calculateSatisfaction** ()

  Calculates the satisfaction level based on nearby entities.
- float getSatisfaction ()

  Gets the satisfaction level of the building.
- void updateAirport (Entity ∗entity)

  Updates the effect of a nearby airport.
- void updateBusStop (Entity ∗entity)

  Updates the effect of a nearby bus stop.
- void updateTrainStation (Entity ∗entity)

  Updates the effect of a nearby train station.
- void updateFactory (Entity ∗entity)

  Updates the effect of a nearby factory.
- void updateShoppingMall (Entity ∗entity)

  Updates the effect of a nearby shopping mall.
- void updateOffice (Entity ∗entity)

  Updates the effect of a nearby office.
- void updateHospital (Entity ∗entity)

  Updates the effect of a nearby hospital.

- void updatePoliceStation (Entity *entity)

  Updates the effect of a nearby police station.
- void updateSchool (Entity *entity)

  Updates the effect of a nearby school.
- void updateAmenity (Entity *entity)

  Updates the effect of a nearby amenity.
- void updateUtility (Entity *entity)

  Updates the effect of a nearby utility.
- void updateIndustry (Entity *entity)

  Updates the effect of a nearby industry.
- int getCapacity ()

  Gets the capacity of the residential building.
- void setCapacity (int capacity)

  Sets the capacity of the residential building.

Public Member Functions inherited from Building

- Building (EntityConfig ec, Size size, int xPos, int yPos)

  Parameterized constructor for the Building class.
- Building (Building *building)

  Copy constructor for the Building class.
- virtual ~Building ()

  Destructor for the Building class.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

Checks if the entity is built (i.e., not under construction).

- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity ∗entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity ∗entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity ∗ > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).
- Size **size**

  Size object representing the entity's dimensions.

- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.81.1 Detailed Description

Represents a residential building entity within the game.

The ResidentialBuilding class handles properties and behaviors specific to residential structures, such as satisfaction calculations based on proximity to various amenities, utilities, and industries. It includes methods for updating these values based on nearby entities.

4.81.2 Constructor & Destructor Documentation

4.81.2.1 ResidentialBuilding() [1/2]

```
ResidentialBuilding::ResidentialBuilding (
            EntityConfig ec,
            Size size,
            int xPos,
            int yPos)
```

Constructs a ResidentialBuilding with specified attributes.

Parameters

ec	Configuration containing resource consumption and other properties.
size	Size of the residential building.
xPos	X-coordinate position of the building.
yPos	Y-coordinate position of the building.

4.81.2.2 ResidentialBuilding() [2/2]

```
ResidentialBuilding::ResidentialBuilding (
            ResidentialBuilding * entity)
```

Copy constructor for the ResidentialBuilding class.

Creates a new ResidentialBuilding by copying the attributes of an existing ResidentialBuilding.

Parameters

entity	Pointer to the ResidentialBuilding object to be copied.

4.81.3 Member Function Documentation

4.81.3.1 clone()

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

Creates a clone of the residential building.

Returns

A pointer to the cloned ResidentialBuilding.

Implements Building.

Implemented in Apartment, and House.

4.81.3.2 getCapacity()

```
int ResidentialBuilding::getCapacity ()
```

Gets the capacity of the residential building.

Returns

The occupant capacity.

4.81.3.3 getSatisfaction()

```
float ResidentialBuilding::getSatisfaction ()
```

Gets the satisfaction level of the building.

Returns

The satisfaction level.

4.81.3.4 setCapacity()

```
void ResidentialBuilding::setCapacity (
            int capacity)
```

Sets the capacity of the residential building.

Parameters

capacity	The new capacity value.

4.81.3.5 update()

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

Updates the residential building's state.

Implements Building.

4.81.3.6 updateAirport()

```
void ResidentialBuilding::updateAirport (
            Entity * entity)
```

Updates the effect of a nearby airport.

Parameters

entity	Entity representing the airport.

4.81.3.7 updateAmenity()

```
void ResidentialBuilding::updateAmenity (
            Entity * entity)
```

Updates the effect of a nearby amenity.

Parameters

entity	Entity representing the amenity.

4.81.3.8 updateBusStop()

```
void ResidentialBuilding::updateBusStop (
            Entity * entity)
```

Updates the effect of a nearby bus stop.

Parameters

entity	Entity representing the bus stop.

4.81.3.9 updateFactory()

```
void ResidentialBuilding::updateFactory (
            Entity * entity)
```

Updates the effect of a nearby factory.

Parameters

entity	Entity representing the factory.

4.81.3.10 updateHospital()

```
void ResidentialBuilding::updateHospital (
            Entity * entity)
```

Updates the effect of a nearby hospital.

Parameters

entity	Entity representing the hospital.

4.81.3.11 updateIndustry()

```
void ResidentialBuilding::updateIndustry (
            Entity * entity)
```

Updates the effect of a nearby industry.

Parameters

entity	Entity representing the industry.

4.81.3.12 updateOffice()

```
void ResidentialBuilding::updateOffice (
            Entity * entity)
```

Updates the effect of a nearby office.

Parameters

entity	Entity representing the office.

4.81.3.13 updatePoliceStation()

```
void ResidentialBuilding::updatePoliceStation (
            Entity * entity)
```

Updates the effect of a nearby police station.

Parameters

entity	Entity representing the police station.

4.81.3.14 updateSchool()

```
void ResidentialBuilding::updateSchool (
            Entity * entity)
```

Updates the effect of a nearby school.

Parameters

entity	Entity representing the school.

4.81.3.15 updateShoppingMall()

```
void ResidentialBuilding::updateShoppingMall (
            Entity * entity)
```

Updates the effect of a nearby shopping mall.

Parameters

entity	Entity representing the shopping mall.

4.81.3.16 updateTrainStation()

```
void ResidentialBuilding::updateTrainStation (
            Entity * entity)
```

Updates the effect of a nearby train station.

Parameters

entity	Entity representing the train station.

4.81.3.17 updateUtility()

```
void ResidentialBuilding::updateUtility (
            Entity * entity)
```

Updates the effect of a nearby utility.

Parameters

entity	Entity representing the utility.

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

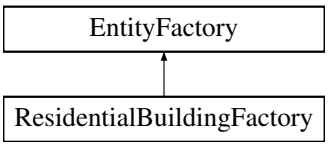
- src/entities/building/residential/ResidentialBuilding.h
- src/entities/building/residential/ResidentialBuilding.cpp

4.82 ResidentialBuildingFactory Class Reference

Factory class for creating residential buildings, including houses and apartments.

`#include <ResidentialBuildingFactory.h>`

Inheritance diagram for ResidentialBuildingFactory:



Public Member Functions

- **ResidentialBuildingFactory** ()

  Default constructor for ResidentialBuildingFactory.
- ∼**ResidentialBuildingFactory** ()

  Destructor for ResidentialBuildingFactory.
- virtual Entity ∗ createEntity (EntityType type, Size size, int xPos, int yPos)

  Creates a residential building of a specified type and size at the given coordinates.

Public Member Functions inherited from EntityFactory

- **EntityFactory** ()

  Default constructor for EntityFactory.
- virtual ∼**EntityFactory** ()

  Virtual destructor for EntityFactory.

4.82.1 Detailed Description

Factory class for creating residential buildings, including houses and apartments.

Inherits from EntityFactory and provides methods to create various-sized residential buildings (small, medium, and large) of different types at specified positions.

4.82.2 Member Function Documentation

4.82.2.1 createEntity()

```
Entity * ResidentialBuildingFactory::createEntity (
            EntityType type,
            Size size,
            int xPos,
            int yPos)  [virtual]
```

Creates a residential building of a specified type and size at the given coordinates.

Parameters

type	Type of residential building to create (e.g., House, Apartment).
size	Size of the building (small, medium, or large).
xPos	X-coordinate for the building's position.
yPos	Y-coordinate for the building's position.

Returns

A pointer to the created Entity.

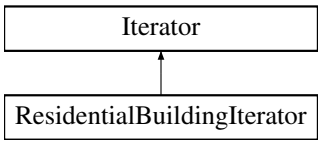
Implements EntityFactory.

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

- src/factory/building/ResidentialBuildingFactory.h
- src/factory/building/ResidentialBuildingFactory.cpp

4.83 ResidentialBuildingIterator Class Reference

Inheritance diagram for ResidentialBuildingIterator:



Public Member Functions

- **ResidentialBuildingIterator** ()

  Construct a new Residential Building Iterator:: Residential Building Iterator object.
- ~**ResidentialBuildingIterator** ()

  Destroy the Residential Building Iterator:: Residential Building Iterator object.
- ResidentialBuildingIterator (std::vector< std::vector< Entity * > > &grid)

  Construct a new Residential Building Iterator:: Residential Building Iterator object.
- void first ()

  Sets the iterator to the first unvisited ResidentialBuilding.
- void next ()

  Advances to the next unvisited ResidentialBuilding.
- bool hasNext ()

  Checks if there is another unvisited ResidentialBuilding.
- Entity * current ()

  Returns the current ResidentialBuilding.

Public Member Functions inherited from Iterator

- **Iterator** ()

  Construct a new Iterator object, initializing row and column to zero.
- virtual ~**Iterator** ()

  Destroy the Iterator object.
- **Iterator** (std::vector< std::vector< Entity * > > &grid)
- virtual int getRow ()

  Get the current row index of the iterator.
- virtual int getCol ()

  Get the current column index of the iterator.

Additional Inherited Members

Protected Member Functions inherited from Iterator

- bool isVisited (Entity *entity)

  Check if the specified entity has been visited.
- void markVisited (Entity *entity)

  Mark the specified entity as visited.

Protected Attributes inherited from Iterator

- std::vector< std::vector< Entity * > > **grid**
- std::vector< std::vector< Entity * > >::iterator **currRow**
- std::vector< Entity * >::iterator **curr**
- int **row**
- int **col**
- std::unordered_set< Entity * > **visitedEntities**

4.83.1 Constructor & Destructor Documentation

4.83.1.1 ResidentialBuildingIterator()

```
ResidentialBuildingIterator::ResidentialBuildingIterator (
            std::vector< std::vector< Entity * > > & grid)
```

Construct a new Residential Building Iterator:: Residential Building Iterator object.

Parameters

grid	

4.83.2 Member Function Documentation

4.83.2.1 current()

```
Entity * ResidentialBuildingIterator::current ()  [virtual]
```

Returns the current ResidentialBuilding.

Returns

Entity*

Implements Iterator.

4.83.2.2 first()

```
void ResidentialBuildingIterator::first ()  [virtual]
```

Sets the iterator to the first unvisited ResidentialBuilding.

Implements Iterator.

4.83.2.3 hasNext()

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

Checks if there is another unvisited ResidentialBuilding.

Returns

true if there is another unvisited ResidentialBuilding, false otherwise

Implements Iterator.

4.83.2.4 next()

```
void ResidentialBuildingIterator::next ()  [virtual]
```

Advances to the next unvisited ResidentialBuilding.

Implements Iterator.

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

- src/iterators/building/residential/ResidentialBuildingIterator.h
- src/iterators/building/residential/ResidentialBuildingIterator.cpp

4.84 ResourceManager Class Reference

Public Member Functions

- **ResourceManager** ()

  Construct a new Resource Manager object.
- ∼**ResourceManager** ()

  Destroy the Resource Manager object.
- void buildIndustry (EntityType type, Size size, int x, int y)

  Build an industry of a specified type and size at given coordinates.
- int **calculateMoneyMade** ()

  Calculate the total money made in the city.
- int **calculateWoodMade** ()

  Calculate the total wood produced in the city.
- int **calculateStoneMade** ()

  Calculate the total stone produced in the city.
- int **calculateConcreteMade** ()

  Calculate the total concrete produced in the city.
- std::vector< Industry ∗ > **getAllIndustryBuildings** ()

  Get all industry buildings in the city.
- std::vector< Industry ∗ > **getAllConcreteProducers** ()

  Get all concrete producer industries.
- std::vector< Industry ∗ > **getAllStoneProducers** ()

  Get all stone producer industries.
- std::vector< Industry ∗ > **getAllWoodProducers** ()

  Get all wood producer industries.
- bool canAffordUpgrade (Industry ∗industry)

  Check if the city can afford an upgrade for a given industry.
- bool upgrade (Industry ∗&industry)

  Upgrade the specified industry if possible.

4.84.1 Member Function Documentation

4.84.1.1 buildIndustry()

```
void ResourceManager::buildIndustry (
            EntityType type,
            Size size,
            int x,
            int y)
```

Build an industry of a specified type and size at given coordinates.

Parameters

type	Type of the industry to build.
size	Size of the industry.
x	X-coordinate for placement.
y	Y-coordinate for placement.

4.84.1.2 canAffordUpgrade()

```
bool ResourceManager::canAffordUpgrade (
            Industry * industry)
```

Check if the city can afford an upgrade for a given industry.

Parameters

industry	Pointer to the industry to check.

Returns

true if the upgrade can be afforded; false otherwise.

4.84.1.3 upgrade()

```
bool ResourceManager::upgrade (
            Industry *& industry)
```

Upgrade the specified industry if possible.

Parameters

industry	Reference to the industry to upgrade.

Returns

true if the upgrade was successful; false otherwise.

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

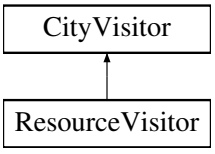
- src/managers/ResourceManager.h
- src/managers/ResourceManager.cpp

4.85 ResourceVisitor Class Reference

Visitor that calculates total resource output in a city.

`#include <ResourceVisitor.h>`

Inheritance diagram for ResourceVisitor:



Public Member Functions

- **ResourceVisitor** ()

  Constructs a ResourceVisitor with zeroed resource totals.
- ~**ResourceVisitor** ()

  Default destructor.
- void visit (City *city) override

  Visits a city to calculate resource production.
- int getTotalWood () const

  Gets the total wood produced.
- int getTotalConcrete () const

  Gets the total concrete produced.
- int getTotalStone () const

  Gets the total stone produced.

Public Member Functions inherited from CityVisitor

- **CityVisitor** ()=default

  Default constructor.
- virtual ~**CityVisitor** ()=default

  Default destructor.

4.85.1 Detailed Description

Visitor that calculates total resource output in a city.

4.85.2 Member Function Documentation

4.85.2.1 getTotalConcrete()

```
int ResourceVisitor::getTotalConcrete () const  [inline]
```

Gets the total concrete produced.

Returns

Total concrete output in the city.

4.85.2.2 getTotalStone()

```
int ResourceVisitor::getTotalStone () const  [inline]
```

Gets the total stone produced.

Returns

Total stone output in the city.

4.85.2.3 getTotalWood()

```
int ResourceVisitor::getTotalWood () const  [inline]
```

Gets the total wood produced.

Returns

Total wood output in the city.

4.85.2.4 visit()

```
void ResourceVisitor::visit (
            City * city)  [override], [virtual]
```

Visits a city to calculate resource production.

Parameters

city	Pointer to the City object being visited.

Implements CityVisitor.

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

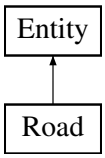
- src/visitors/resource/ResourceVisitor.h
- src/visitors/resource/ResourceVisitor.cpp

4.86 Road Class Reference

Represents a road entity within the game.

```
#include <Road.h>
```

Inheritance diagram for Road:



Public Member Functions

- Road (EntityConfig ec, Size size, int xPos, int yPos)
  Constructs a Road entity with specified attributes.
- Road (Road *road)
  Copy constructor for the Road class.
- ~**Road** ()
  Destructor for the Road class.
- void update ()
  Updates the state of the road entity.
- Entity * clone ()
  Creates a clone of the road entity.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).
- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.86.1 Detailed Description

Represents a road entity within the game.

The Road class manages the properties and behavior of road entities, including their position and size. This class provides functionality to update the road's state and to create clones of road entities.

4.86.2 Constructor & Destructor Documentation

4.86.2.1 Road() [1/2]

```
Road::Road (
            EntityConfig ec,
            Size size,
            int xPos,
            int yPos)
```

Constructs a Road entity with specified attributes.

Parameters

ec	Configuration settings for the entity.
size	Size of the road entity.
xPos	X-coordinate position of the road.
yPos	Y-coordinate position of the road.

4.86.2.2 Road() [2/2]

```
Road::Road (
            Road * road)
```

Copy constructor for the Road class.

Creates a new Road entity by copying the attributes of an existing Road.

Parameters

road	Pointer to the Road object to be copied.

4.86.3 Member Function Documentation

4.86.3.1 clone()

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

Creates a clone of the road entity.

Returns

A pointer to the cloned Road entity.

Implements Entity.

4.86.3.2 update()

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

Updates the state of the road entity.

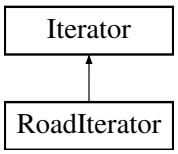
Implements Entity.

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

- src/entities/road/Road.h
- src/entities/road/Road.cpp

4.87 RoadIterator Class Reference

Inheritance diagram for RoadIterator:



Public Member Functions

- **RoadIterator** ()

  Construct a new Road Iterator:: Road Iterator object.
- ~**RoadIterator** ()

  Destroy the Road Iterator:: Road Iterator object.
- RoadIterator (std::vector< std::vector< Entity * > > &grid)

  Construct a new Road Iterator:: Road Iterator object.
- void first ()

  Sets the iterator to the first unvisited Road.
- void next ()

  Advances to the next unvisited Road.
- bool hasNext ()

  Checks if there is another unvisited Road.
- Entity * current ()

  Returns the current Road.

Public Member Functions inherited from Iterator

- **Iterator** ()

  Construct a new Iterator object, initializing row and column to zero.
- virtual ~**Iterator** ()

  Destroy the Iterator object.
- **Iterator** (std::vector< std::vector< Entity * > > &grid)
- virtual int getRow ()

  Get the current row index of the iterator.
- virtual int getCol ()

  Get the current column index of the iterator.

Additional Inherited Members

Protected Member Functions inherited from Iterator

- bool isVisited (Entity *entity)

  Check if the specified entity has been visited.
- void markVisited (Entity *entity)

  Mark the specified entity as visited.

Protected Attributes inherited from Iterator

- std::vector< std::vector< Entity * > > **grid**
- std::vector< std::vector< Entity * > >::iterator **currRow**
- std::vector< Entity * >::iterator **curr**
- int **row**
- int **col**
- std::unordered_set< Entity * > **visitedEntities**

4.87.1 Constructor & Destructor Documentation

4.87.1.1 RoadIterator()

```
RoadIterator::RoadIterator (
            std::vector< std::vector< Entity * > > & grid)
```

Construct a new Road Iterator:: Road Iterator object.

Parameters

grid	

4.87.2 Member Function Documentation

4.87.2.1 current()

```
Entity * RoadIterator::current ()  [virtual]
```

Returns the current Road.

Returns

Entity*

Implements Iterator.

4.87.2.2 first()

```
void RoadIterator::first ()  [virtual]
```

Sets the iterator to the first unvisited Road.

Implements Iterator.

4.87.2.3 hasNext()

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

Checks if there is another unvisited Road.

Returns

true if there is another unvisited Road, false otherwise

Implements Iterator.

4.87.2.4 next()

```
void RoadIterator::next () [virtual]
```

Advances to the next unvisited Road.

Implements Iterator.

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

- src/iterators/road/RoadIterator.h
- src/iterators/road/RoadIterator.cpp

4.88 SatisfactionConfig Struct Reference

Public Member Functions

- **SatisfactionConfig** (float localRate=0.0f, float globalRate=0.0f, float localExtreme=0.0f, float global↩ Extreme=0.0f)

Public Attributes

- float **localRate**
- float **globalRate**
- float **localExtreme**
- float **globalExtreme**

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

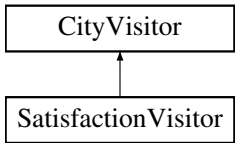
- src/utils/SatisfactionConfig.h

4.89 SatisfactionVisitor Class Reference

Visitor that calculates the average satisfaction of residential buildings in a city.

`#include <SatisfactionVisitor.h>`

Inheritance diagram for SatisfactionVisitor:



Public Member Functions

- **SatisfactionVisitor** ()
  Constructs a SatisfactionVisitor with zeroed satisfaction and count.
- ~**SatisfactionVisitor** ()
  Default destructor.
- void visit (City *city) override
  Visits a city to calculate residential satisfaction.
- float getAverageSatisfaction () const
  Gets the average satisfaction of residential buildings.
- int getResidentialCount () const
  Gets the count of residential buildings.

Public Member Functions inherited from CityVisitor

- **CityVisitor** ()=default
  Default constructor.
- virtual ~**CityVisitor** ()=default
  Default destructor.

4.89.1 Detailed Description

Visitor that calculates the average satisfaction of residential buildings in a city.

4.89.2 Member Function Documentation

4.89.2.1 getAverageSatisfaction()

```
float SatisfactionVisitor::getAverageSatisfaction () const
```

Gets the average satisfaction of residential buildings.

Returns

Average satisfaction, or 0 if no residential buildings.

4.89.2.2 getResidentialCount()

```
int SatisfactionVisitor::getResidentialCount () const
```

Gets the count of residential buildings.

Returns

Number of residential buildings in the city.

4.89.2.3 visit()

```
void SatisfactionVisitor::visit (
            City * city)  [override], [virtual]
```

Visits a city to calculate residential satisfaction.

Parameters

city	Pointer to the City object being visited.

Implements CityVisitor.

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

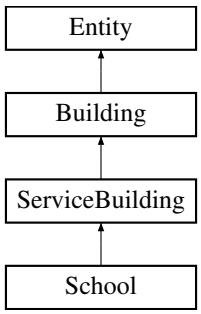
- src/visitors/satisfaction/SatisfactionVisitor.h
- src/visitors/satisfaction/SatisfactionVisitor.cpp

4.90 School Class Reference

Class representing a school in the city.

```
#include <School.h>
```

Inheritance diagram for School:



Public Member Functions

- School (EntityConfig ec, Size size, int xPos, int yPos)

  Parameterized constructor for the School class.
- School (School *school)

  Copy constructor for the School class.
- ~School ()

  Destructor for the School class.
- void update ()

  Updates the state of the school entity.
- Entity * clone ()

  Clones the school entity.

Public Member Functions inherited from ServiceBuilding

- ServiceBuilding (EntityConfig ec, Size size, int xPos, int yPos)

  Parameterized constructor for the ServiceBuilding class.
- ServiceBuilding (ServiceBuilding *service)

  Copy constructor for the ServiceBuilding class.
- virtual ~ServiceBuilding ()

  Destructor for the ServiceBuilding class.

Public Member Functions inherited from Building

- Building (EntityConfig ec, Size size, int xPos, int yPos)

  Parameterized constructor for the Building class.
- Building (Building *building)

  Copy constructor for the Building class.
- virtual ~Building ()

  Destructor for the Building class.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.

- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

Height of the entity.

- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).
- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.90.1 Detailed Description

Class representing a school in the city.

The School class provides education services to the population. It inherits from the ServiceBuilding class.

4.90.2 Constructor & Destructor Documentation

4.90.2.1 School() [1/2]

```
School::School (
            EntityConfig ec,
            Size size,
            int xPos,
            int yPos)
```

Parameterized constructor for the School class.

Parameters

ec	Entity configuration for initializing the school.
size	The size of the school.
xPos	The x-coordinate of the school's location.
yPos	The y-coordinate of the school's location.

4.90.2.2 School() [2/2]

```
School::School (
            School * school)
```

Copy constructor for the School class.

Parameters

school	A pointer to an existing School object to copy from.

4.90.2.3 ~School()

```
School::~School ()
```

Destructor for the School class.

Cleans up resources used by the School object.

4.90.3 Member Function Documentation

4.90.3.1 clone()

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

Clones the school entity.

Returns

A pointer to a deep copy of the School object.

Implements ServiceBuilding.

4.90.3.2 update()

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

Updates the state of the school entity.

Handles changes in the school's state.

Implements ServiceBuilding.

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

- src/entities/building/service/School.h
- src/entities/building/service/School.cpp

4.91 Section Struct Reference

Represents a section in the menu, containing a heading and multiple options.

```
#include <IMenu.h>
```

Public Attributes

- std::string **heading**

  The heading/title of the section.
- std::vector< Option > **options**

  The list of options within the section.

4.91.1 Detailed Description

Represents a section in the menu, containing a heading and multiple options.

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

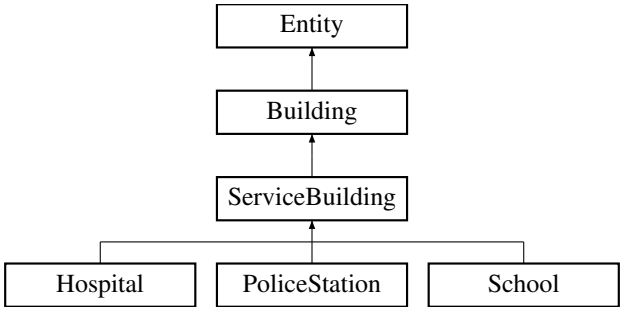
- src/menus/base/IMenu.h

4.92 ServiceBuilding Class Reference

Abstract class representing a service building in the city.

`#include <ServiceBuilding.h>`

Inheritance diagram for ServiceBuilding:



Public Member Functions

- ServiceBuilding (EntityConfig ec, Size size, int xPos, int yPos)

  Parameterized constructor for the ServiceBuilding class.
- ServiceBuilding (ServiceBuilding *service)

  Copy constructor for the ServiceBuilding class.
- virtual ~ServiceBuilding ()

  Destructor for the ServiceBuilding class.
- virtual void update ()=0

  Pure virtual function for updating the state of the service building.
- virtual Entity * clone ()=0

  Pure virtual function for cloning the service building.

Public Member Functions inherited from Building

- Building (EntityConfig ec, Size size, int xPos, int yPos)

  Parameterized constructor for the Building class.
- Building (Building *building)

  Copy constructor for the Building class.
- virtual ~Building ()

  Destructor for the Building class.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

Gets the list of entities observing this entity.

- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).
- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.92.1 Detailed Description

Abstract class representing a service building in the city.

Service buildings provide essential services such as healthcare, law enforcement, and education. It inherits from the Building and Subject classes.

4.92.2 Constructor & Destructor Documentation

4.92.2.1 ServiceBuilding() [1/2]

```
ServiceBuilding::ServiceBuilding (
            EntityConfig ec,
            Size size,
            int xPos,
            int yPos)
```

Parameterized constructor for the ServiceBuilding class.

Parameters

ec	Entity configuration for initializing the building.
size	The size of the building.
xPos	The x-coordinate of the building's location.
yPos	The y-coordinate of the building's location.

4.92.2.2 ServiceBuilding() [2/2]

```
ServiceBuilding::ServiceBuilding (
            ServiceBuilding * service)
```

Copy constructor for the ServiceBuilding class.

Parameters

service	A pointer to an existing ServiceBuilding object to copy from.

4.92.2.3 ~ServiceBuilding()

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

Destructor for the ServiceBuilding class.

Cleans up resources used by the ServiceBuilding object.

4.92.3 Member Function Documentation

4.92.3.1 clone()

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

Pure virtual function for cloning the service building.

Returns

A pointer to a deep copy of the service building.

Implements Building.

Implemented in Hospital, PoliceStation, and School.

4.92.3.2 update()

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

Pure virtual function for updating the state of the service building.

Must be implemented by derived classes to handle building-specific updates.

Implements Building.

Implemented in Hospital, PoliceStation, and School.

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

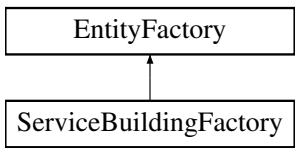
- src/entities/building/service/ServiceBuilding.h
- src/entities/building/service/ServiceBuilding.cpp

4.93 ServiceBuildingFactory Class Reference

Factory class for creating service buildings such as hospitals, police stations, and schools.

```
#include <ServiceBuildingFactory.h>
```

Inheritance diagram for ServiceBuildingFactory:



Public Member Functions

- **ServiceBuildingFactory** ()

  Default constructor for ServiceBuildingFactory.
- ∼**ServiceBuildingFactory** ()

  Destructor for ServiceBuildingFactory.
- virtual Entity * createEntity (EntityType type, Size size, int xPos, int yPos) override

  Creates a service building of the specified type and size at the given position.

Public Member Functions inherited from EntityFactory

- **EntityFactory** ()

  Default constructor for EntityFactory.
- virtual ∼**EntityFactory** ()

  Virtual destructor for EntityFactory.

4.93.1 Detailed Description

Factory class for creating service buildings such as hospitals, police stations, and schools.

Inherits from EntityFactory and provides methods to create various-sized service buildings (small, medium, and large) of different types, positioned at specified coordinates.

4.93.2 Member Function Documentation

4.93.2.1 createEntity()

```
Entity * ServiceBuildingFactory::createEntity (
            EntityType type,
            Size size,
            int xPos,
            int yPos)  [override], [virtual]
```

Creates a service building of the specified type and size at the given position.

Parameters

type	The type of service building to create (e.g., Hospital, PoliceStation, School).
size	The size of the service building (small, medium, or large).
xPos	The x-coordinate of the building's position.
yPos	The y-coordinate of the building's position.

Returns

A pointer to the created Entity.

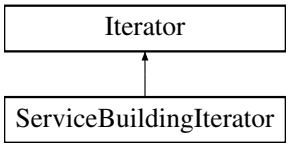
Implements EntityFactory.

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

- src/factory/building/ServiceBuildingFactory.h
- src/factory/building/ServiceBuildingFactory.cpp

4.94 ServiceBuildingIterator Class Reference

Inheritance diagram for ServiceBuildingIterator:



Public Member Functions

- **ServiceBuildingIterator** ()

  Construct a new Service Building Iterator:: Service Building Iterator object.
- ∼**ServiceBuildingIterator** ()

  Destroy the Service Building Iterator:: Service Building Iterator object.
- ServiceBuildingIterator (std::vector< std::vector< Entity * > > &grid)

  Construct a new Service Building Iterator:: Service Building Iterator object.
- void first ()

  Sets the iterator to the first unvisited ServiceBuilding.
- void next ()

  Advances to the next unvisited ServiceBuilding.
- bool hasNext ()

  Checks if there is another unvisited ServiceBuilding.
- Entity * current ()

  Returns the current ServiceBuilding.

Public Member Functions inherited from Iterator

- **Iterator** ()

  Construct a new Iterator object, initializing row and column to zero.
- virtual ∼**Iterator** ()

  Destroy the Iterator object.
- **Iterator** (std::vector< std::vector< Entity * > > &grid)
- virtual int getRow ()

  Get the current row index of the iterator.
- virtual int getCol ()

  Get the current column index of the iterator.

Additional Inherited Members

Protected Member Functions inherited from Iterator

- bool isVisited (Entity *entity)

  Check if the specified entity has been visited.
- void markVisited (Entity *entity)

  Mark the specified entity as visited.

Protected Attributes inherited from Iterator

- std::vector< std::vector< Entity * > > **grid**
- std::vector< std::vector< Entity * > >::iterator **currRow**
- std::vector< Entity * >::iterator **curr**
- int **row**
- int **col**
- std::unordered_set< Entity * > **visitedEntities**

4.94.1 Constructor & Destructor Documentation

4.94.1.1 ServiceBuildingIterator()

```
ServiceBuildingIterator::ServiceBuildingIterator (
            std::vector< std::vector< Entity * > > & grid)
```

Construct a new Service Building Iterator:: Service Building Iterator object.

Parameters

grid	

4.94.2 Member Function Documentation

4.94.2.1 current()

```
Entity * ServiceBuildingIterator::current ()  [virtual]
```

Returns the current ServiceBuilding.

Returns

Entity*

Implements Iterator.

4.94.2.2 first()

```
void ServiceBuildingIterator::first ()  [virtual]
```

Sets the iterator to the first unvisited ServiceBuilding.

Implements Iterator.

4.94.2.3 hasNext()

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

Checks if there is another unvisited ServiceBuilding.

Returns

true if there is another unvisited ServiceBuilding, false otherwise

Implements Iterator.

4.94.2.4 next()

```
void ServiceBuildingIterator::next () [virtual]
```

Advances to the next unvisited ServiceBuilding.

Implements Iterator.

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

- src/iterators/building/service/ServiceBuildingIterator.h
- src/iterators/building/service/ServiceBuildingIterator.cpp

4.95 ServiceManager Class Reference

Manages the creation and destruction of service buildings.

```
#include <ServiceManager.h>
```

Public Member Functions

- **ServiceManager** ()
  Constructs a new ServiceManager object.
- ∼**ServiceManager** ()
  Destroys the ServiceManager object.
- bool buildService (EntityType type, Size size, int xPos, int yPos)
  Builds a service of the specified type, size, and location within the city.

4.95.1 Detailed Description

Manages the creation and destruction of service buildings.

The ServiceManager class is responsible for constructing and initializing service buildings of various types and sizes at specified coordinates.

4.95.2 Member Function Documentation

4.95.2.1 buildService()

```
bool ServiceManager::buildService (
            EntityType type,
            Size size,
            int xPos,
            int yPos)
```

Builds a service of the specified type, size, and location within the city.

Builds a new service building.

Parameters

type	Type of service to build (e.g., School, Hospital).
size	Size of the service (e.g., small, medium, large).
x	X-coordinate in the city grid.
y	Y-coordinate in the city grid.

Creates and initializes a service building of the specified type and size at the given position coordinates.

Parameters

type	The entity type of the service building to create.
size	The size of the service building.
xPos	The x-coordinate position of the service building.
yPos	The y-coordinate position of the service building.

Returns

A boolean indicating success or failure of the building creation.

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

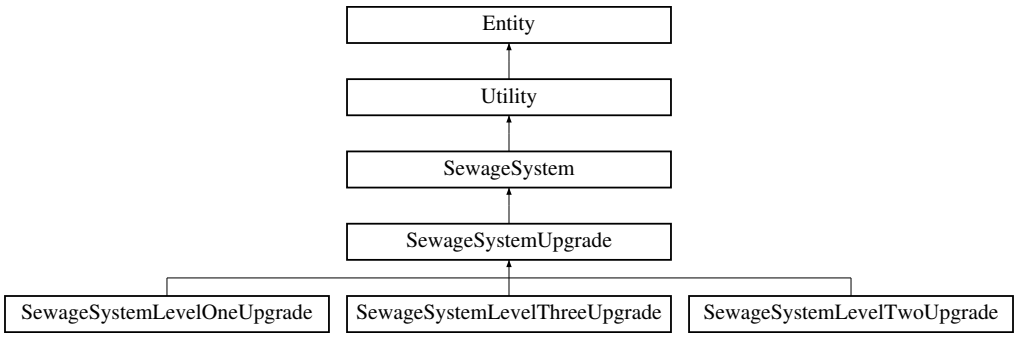
- src/managers/ServiceManager.h
- src/managers/ServiceManager.cpp

4.96 SewageSystem Class Reference

Represents a sewage system in the city builder simulation.

`#include <SewageSystem.h>`

Inheritance diagram for SewageSystem:



Public Member Functions

- SewageSystem (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a SewageSystem object with specified attributes.
- SewageSystem (SewageSystem ∗sewage)

  Copy constructor for the SewageSystem class.
- virtual ∼**SewageSystem** ()

  Destructor for the SewageSystem object.
- void update () override

  Updates the state of the sewage system.
- Entity ∗ clone () override

  Clones the current SewageSystem object.
- Entity ∗ upgrade () override

  Upgrades the current utility to the next level.

Public Member Functions inherited from Utility

- Utility (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a Utility object with the specified parameters.
- Utility (Utility ∗utility)

  Copy constructor for the Utility class.
- virtual ∼**Utility** ()

  Destructor for the Utility object.
- virtual int getOutput ()

  Retrieves the output of the utility.
- void setOutput (int output)

  Sets the output value of the utility.
- virtual Cost getCost ()

  Retrieves the cost of the utility or its upgraded version.
- virtual int getLevel ()

  Gets the level of the utility.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity ∗entity)

  Copy constructor for the Entity class.
- virtual ∼**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity ∗entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

Sets the Y-coordinate position of the entity.

- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.

- int **height**
  Height of the entity.
- int **xPosition**
  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**
  Y-coordinate of the entity's position (bottom left corner).
- Size **size**
  Size object representing the entity's dimensions.
- EntityType **type**
  The type of entity.
- State * **state**
  Pointer to the current state of the entity.
- int **revenue**
  Revenue generated by the entity.
- float **electricityConsumption**
  Electricity consumption of the entity.
- float **waterConsumption**
  Water consumption of the entity.
- std::vector< Entity * > **observers**
  List of other entities observing this entity.

4.96.1 Detailed Description

Represents a sewage system in the city builder simulation.

The SewageSystem class is a type of Utility that handles the city's sewage management.

4.96.2 Constructor & Destructor Documentation

4.96.2.1 SewageSystem() [1/2]

```
SewageSystem::SewageSystem (
            EntityConfig ec,
            Size size,
            int xPos,
            int yPos)
```

Constructs a SewageSystem object with specified attributes.

Initializes a SewageSystem with detailed parameters, including utility consumption, effects, and dimensions.

Parameters

ec	EntityConfig.
size	Size.
xPos	xPosition
yPos	yPosition

4.96.2.2 SewageSystem() [2/2]

```
SewageSystem::SewageSystem (
            SewageSystem * sewage)
```

Copy constructor for the SewageSystem class.

Creates a new SewageSystem object by copying the attributes of an existing SewageSystem.

Parameters

sewage	Pointer to the existing SewageSystem object to be copied.

4.96.3 Member Function Documentation

4.96.3.1 clone()

```
Entity * SewageSystem::clone ()  [override], [virtual]
```

Clones the current SewageSystem object.

Creates and returns a copy of the current SewageSystem instance.

Returns

A pointer to the newly cloned SewageSystem object.

Implements Utility.

Reimplemented in SewageSystemLevelOneUpgrade, SewageSystemLevelThreeUpgrade, SewageSystemLevelTwoUpgrade, and SewageSystemUpgrade.

4.96.3.2 update()

```
void SewageSystem::update ()  [override], [virtual]
```

Updates the state of the sewage system.

Defines the specific behavior of the SewageSystem when it is updated in the simulation.

Implements Utility.

Reimplemented in SewageSystemLevelOneUpgrade, SewageSystemLevelThreeUpgrade, SewageSystemLevelTwoUpgrade, and SewageSystemUpgrade.

4.96.3.3 upgrade()

```
Entity * SewageSystem::upgrade ()  [override], [virtual]
```

Upgrades the current utility to the next level.

Returns

A pointer to the upgraded utility instance, or nullptr if already at maximum level.

Implements Utility.

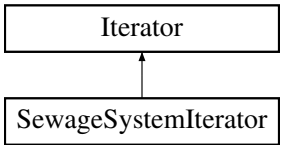
Reimplemented in SewageSystemLevelOneUpgrade, SewageSystemLevelThreeUpgrade, SewageSystemLevelTwoUpgrade, and SewageSystemUpgrade.

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

- src/entities/utility/sewagesystem/SewageSystem.h
- src/entities/utility/sewagesystem/SewageSystem.cpp

4.97 SewageSystemIterator Class Reference

Inheritance diagram for SewageSystemIterator:



Public Member Functions

- **SewageSystemIterator** ()

  Construct a new Sewage System Iterator object.
- ~**SewageSystemIterator** ()

  Destroy the Sewage System Iterator object.
- SewageSystemIterator (std::vector< std::vector< Entity * > > &grid)

  Construct a new Sewage System Iterator object with grid.
- void first ()

  Resets the iterator to the first unvisited SewageSystem.
- void next ()

  Advances to the next unvisited SewageSystem.
- bool hasNext ()

  Checks if there is another unvisited SewageSystem.
- Entity * current ()

  Returns the current SewageSystem.

Public Member Functions inherited from Iterator

- **Iterator** ()

  Construct a new Iterator object, initializing row and column to zero.
- virtual ~**Iterator** ()

  Destroy the Iterator object.
- **Iterator** (std::vector< std::vector< Entity * > > &grid)
- virtual int getRow ()

  Get the current row index of the iterator.
- virtual int getCol ()

  Get the current column index of the iterator.

Additional Inherited Members

Protected Member Functions inherited from Iterator

- bool isVisited (Entity *entity)

  Check if the specified entity has been visited.
- void markVisited (Entity *entity)

  Mark the specified entity as visited.

Protected Attributes inherited from Iterator

- std::vector< std::vector< Entity * > > **grid**
- std::vector< std::vector< Entity * > >::iterator **currRow**
- std::vector< Entity * >::iterator **curr**
- int **row**
- int **col**
- std::unordered_set< Entity * > **visitedEntities**

4.97.1 Constructor & Destructor Documentation

4.97.1.1 SewageSystemIterator()

```
SewageSystemIterator::SewageSystemIterator (
            std::vector< std::vector< Entity * > > & grid)
```

Construct a new Sewage System Iterator object with grid.

Parameters

grid	

4.97.2 Member Function Documentation

4.97.2.1 current()

```
Entity * SewageSystemIterator::current ()  [virtual]
```

Returns the current SewageSystem.

Returns

Entity*

Implements Iterator.

4.97.2.2 first()

```
void SewageSystemIterator::first ()  [virtual]
```

Resets the iterator to the first unvisited SewageSystem.

Implements Iterator.

4.97.2.3 hasNext()

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

Checks if there is another unvisited SewageSystem.

Returns

true if there is another unvisited SewageSystem, false otherwise

Implements Iterator.

4.97.2.4 next()

```
void SewageSystemIterator::next ()  [virtual]
```

Advances to the next unvisited SewageSystem.

Implements Iterator.

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

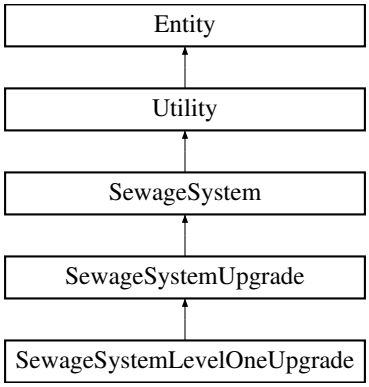
- src/iterators/utility/SewageSystemIterator.h
- src/iterators/utility/SewageSystemIterator.cpp

4.98 SewageSystemLevelOneUpgrade Class Reference

Represents the first level upgrade to a SewageSystem entity.

```
#include <SewageSystemLevelOneUpgrade.h>
```

Inheritance diagram for SewageSystemLevelOneUpgrade:



Public Member Functions

- SewageSystemLevelOneUpgrade (SewageSystem ∗sewage)

  Constructs a SewageSystemLevelOneUpgrade object.
- SewageSystemLevelOneUpgrade (SewageSystemLevelOneUpgrade ∗sSLOU)

  Copy constructor for SewageSystemLevelOneUpgrade.
- ∼SewageSystemLevelOneUpgrade ()

  Destructor for SewageSystemLevelOneUpgrade.
- void update () override

  Updates the state of the upgraded sewage system.
- Entity ∗ clone () override

  Clones the current SewageSystemLevelOneUpgrade object.
- Entity ∗ upgrade () override

  Upgrades the current utility to the next level.
- int getOutput () override

  Retrieves the upgraded sewage system's output.
- Cost getCost () override

  Retrieves the cost of the utility or its upgraded version.
- int getLevel () override

  Gets the level of the sewage system upgrade.

Public Member Functions inherited from SewageSystemUpgrade

- SewageSystemUpgrade (SewageSystem ∗sewage)

  Constructs a SewageSystemUpgrade object based on an existing SewageSystem.
- SewageSystemUpgrade (SewageSystemUpgrade ∗sSU)

  Copy constructor for the SewageSystemUpgrade class.
- virtual ∼**SewageSystemUpgrade** ()

  Destructor for the SewageSystemUpgrade object.

Public Member Functions inherited from SewageSystem

- SewageSystem (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a SewageSystem object with specified attributes.
- SewageSystem (SewageSystem ∗sewage)

  Copy constructor for the SewageSystem class.
- virtual ∼**SewageSystem** ()

  Destructor for the SewageSystem object.

Public Member Functions inherited from Utility

- Utility (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a Utility object with the specified parameters.
- Utility (Utility ∗utility)

  Copy constructor for the Utility class.
- virtual ∼**Utility** ()

  Destructor for the Utility object.
- void setOutput (int output)

  Sets the output value of the utility.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from SewageSystemUpgrade

- SewageSystem * **sewageSystem**

  Pointer to the original SewageSystem that is being upgraded.

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).
- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.98.1 Detailed Description

Represents the first level upgrade to a SewageSystem entity.

The SewageSystemLevelOneUpgrade class enhances the base functionality of a SewageSystem by increasing its output. This class is the first upgrade level in a series of potential sewage system improvements.

4.98.2 Constructor & Destructor Documentation

4.98.2.1 SewageSystemLevelOneUpgrade() [1/2]

```
SewageSystemLevelOneUpgrade::SewageSystemLevelOneUpgrade (
            SewageSystem * sewage)
```

Constructs a SewageSystemLevelOneUpgrade object.

Initializes the upgrade by enhancing the specified SewageSystem with a level one upgrade.

Parameters

sewage	Pointer to the original SewageSystem to be upgraded.

4.98.2.2 SewageSystemLevelOneUpgrade() [2/2]

```
SewageSystemLevelOneUpgrade::SewageSystemLevelOneUpgrade (
            SewageSystemLevelOneUpgrade * sSLOU)
```

Copy constructor for SewageSystemLevelOneUpgrade.

Creates a new SewageSystemLevelOneUpgrade object by copying the attributes of an existing SewageSystemLevelOneUpgrade object.

Parameters

sSLOU	Pointer to the existing SewageSystemLevelOneUpgrade to be copied.

4.98.2.3 ~SewageSystemLevelOneUpgrade()

```
SewageSystemLevelOneUpgrade::~SewageSystemLevelOneUpgrade ()
```

Destructor for SewageSystemLevelOneUpgrade.

Cleans up any resources associated with the upgrade.

4.98.3 Member Function Documentation

4.98.3.1 clone()

```
Entity * SewageSystemLevelOneUpgrade::clone ()  [override], [virtual]
```

Clones the current SewageSystemLevelOneUpgrade object.

Creates a new instance of SewageSystemLevelOneUpgrade with the same attributes as the current object.

Returns

A pointer to the newly cloned SewageSystemLevelOneUpgrade object.

Implements SewageSystemUpgrade.

4.98.3.2 getCost()

```
Cost SewageSystemLevelOneUpgrade::getCost ()  [override], [virtual]
```

Retrieves the cost of the utility or its upgraded version.

Returns

A Cost object representing the monetary and material costs.

Implements SewageSystemUpgrade.

4.98.3.3 getLevel()

```
int SewageSystemLevelOneUpgrade::getLevel ()  [override], [virtual]
```

Gets the level of the sewage system upgrade.

Returns

The level of the sewage system upgrade.

Reimplemented from Utility.

4.98.3.4 getOutput()

```
int SewageSystemLevelOneUpgrade::getOutput ()  [override], [virtual]
```

Retrieves the upgraded sewage system's output.

Returns the output of the level one upgraded sewage system.

Returns

The updated output as an integer.

Implements SewageSystemUpgrade.

4.98.3.5 update()

```
void SewageSystemLevelOneUpgrade::update ()  [override], [virtual]
```

Updates the state of the upgraded sewage system.

Implements specific behavior for the sewage system after applying the level one upgrade.

Implements SewageSystemUpgrade.

4.98.3.6 upgrade()

```
Entity * SewageSystemLevelOneUpgrade::upgrade ()  [override], [virtual]
```

Upgrades the current utility to the next level.

Returns

A pointer to the upgraded utility instance, or nullptr if already at maximum level.

Implements SewageSystemUpgrade.

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

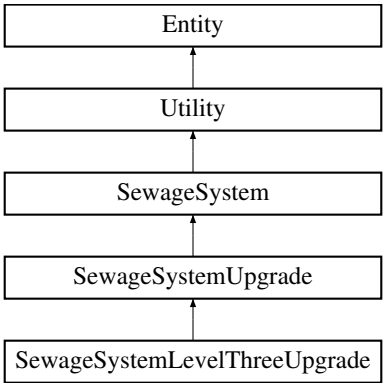
- src/entities/utility/sewagesystem/SewageSystemLevelOneUpgrade.h
- src/entities/utility/sewagesystem/SewageSystemLevelOneUpgrade.cpp

4.99 SewageSystemLevelThreeUpgrade Class Reference

Represents the third level upgrade to a SewageSystem entity.

`#include <SewageSystemLevelThreeUpgrade.h>`

Inheritance diagram for SewageSystemLevelThreeUpgrade:



Public Member Functions

- SewageSystemLevelThreeUpgrade (SewageSystem *sewage)
  Constructs a SewageSystemLevelThreeUpgrade object.
- SewageSystemLevelThreeUpgrade (SewageSystemLevelThreeUpgrade *sSLTU)
  Copy constructor for SewageSystemLevelThreeUpgrade.
- ~SewageSystemLevelThreeUpgrade ()
  Destructor for SewageSystemLevelThreeUpgrade.
- void update () override
  Updates the state of the upgraded sewage system.
- Entity * clone () override
  Clones the current SewageSystemLevelThreeUpgrade object.
- Entity * upgrade () override
  Upgrades the current utility to the next level.
- int getOutput () override
  Retrieves the upgraded sewage system's output.
- Cost getCost () override
  Retrieves the cost of the utility or its upgraded version.
- int getLevel () override
  Gets the level of the sewage system upgrade.

Public Member Functions inherited from SewageSystemUpgrade

- SewageSystemUpgrade (SewageSystem *sewage)

  Constructs a SewageSystemUpgrade object based on an existing SewageSystem.
- SewageSystemUpgrade (SewageSystemUpgrade *sSU)

  Copy constructor for the SewageSystemUpgrade class.
- virtual ~**SewageSystemUpgrade** ()

  Destructor for the SewageSystemUpgrade object.

Public Member Functions inherited from SewageSystem

- SewageSystem (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a SewageSystem object with specified attributes.
- SewageSystem (SewageSystem *sewage)

  Copy constructor for the SewageSystem class.
- virtual ~**SewageSystem** ()

  Destructor for the SewageSystem object.

Public Member Functions inherited from Utility

- Utility (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a Utility object with the specified parameters.
- Utility (Utility *utility)

  Copy constructor for the Utility class.
- virtual ~**Utility** ()

  Destructor for the Utility object.
- void setOutput (int output)

  Sets the output value of the utility.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.

- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from SewageSystemUpgrade

- SewageSystem * **sewageSystem**

  Pointer to the original SewageSystem that is being upgraded.

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

Global effect strength of the entity.

- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).
- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.99.1 Detailed Description

Represents the third level upgrade to a SewageSystem entity.

The SewageSystemLevelThreeUpgrade class enhances the base functionality of a SewageSystem by increasing its processing capabilities. This class is the third upgrade level in a series of potential sewage system improvements.

4.99.2 Constructor & Destructor Documentation

4.99.2.1 SewageSystemLevelThreeUpgrade() [1/2]

```
SewageSystemLevelThreeUpgrade::SewageSystemLevelThreeUpgrade (
            SewageSystem * sewage)
```

Constructs a SewageSystemLevelThreeUpgrade object.

Initializes the upgrade by enhancing the specified SewageSystem with a level three upgrade.

Parameters

sewage	Pointer to the original SewageSystem to be upgraded.

4.99.2.2 SewageSystemLevelThreeUpgrade() [2/2]

```
SewageSystemLevelThreeUpgrade::SewageSystemLevelThreeUpgrade (
            SewageSystemLevelThreeUpgrade * sSLTU)
```

Copy constructor for SewageSystemLevelThreeUpgrade.

Creates a new SewageSystemLevelThreeUpgrade object by copying the attributes of an existing SewageSystemLevelThreeUpgrade object.

Parameters

sSLTU	Pointer to the existing SewageSystemLevelThreeUpgrade to be copied.

4.99.2.3 ~SewageSystemLevelThreeUpgrade()

```
SewageSystemLevelThreeUpgrade::~SewageSystemLevelThreeUpgrade ()
```

Destructor for SewageSystemLevelThreeUpgrade.

Cleans up any resources associated with the upgrade.

4.99.3 Member Function Documentation

4.99.3.1 clone()

```
Entity * SewageSystemLevelThreeUpgrade::clone ()  [override], [virtual]
```

Clones the current SewageSystemLevelThreeUpgrade object.

Creates a new instance of SewageSystemLevelThreeUpgrade with the same attributes as the current object.

Returns

A pointer to the newly cloned SewageSystemLevelThreeUpgrade object.

Implements SewageSystemUpgrade.

4.99.3.2 getCost()

```
Cost SewageSystemLevelThreeUpgrade::getCost ()  [override], [virtual]
```

Retrieves the cost of the utility or its upgraded version.

Returns

A Cost object representing the monetary and material costs.

Implements SewageSystemUpgrade.

4.99.3.3 getLevel()

```
int SewageSystemLevelThreeUpgrade::getLevel ()  [override], [virtual]
```

Gets the level of the sewage system upgrade.

Returns

The level of the sewage system upgrade.

Reimplemented from Utility.

4.99.3.4 getOutput()

```
int SewageSystemLevelThreeUpgrade::getOutput ()  [override], [virtual]
```

Retrieves the upgraded sewage system's output.

Returns the processing output of the level three upgraded sewage system.

Returns

The updated processing output as an integer.

Implements SewageSystemUpgrade.

4.99.3.5 update()

```
void SewageSystemLevelThreeUpgrade::update ()  [override], [virtual]
```

Updates the state of the upgraded sewage system.

Implements specific behavior for the sewage system after applying the level three upgrade.

Implements SewageSystemUpgrade.

4.99.3.6 upgrade()

```
Entity * SewageSystemLevelThreeUpgrade::upgrade ()  [override], [virtual]
```

Upgrades the current utility to the next level.

Returns

A pointer to the upgraded utility instance, or nullptr if already at maximum level.

Implements SewageSystemUpgrade.

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

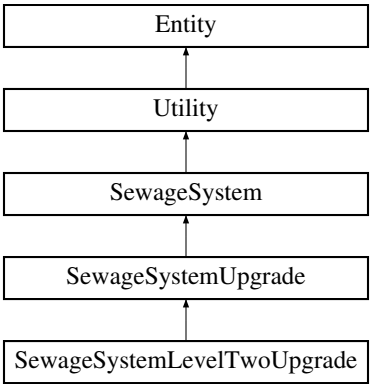
- src/entities/utility/sewagesystem/SewageSystemLevelThreeUpgrade.h
- src/entities/utility/sewagesystem/SewageSystemLevelThreeUpgrade.cpp

4.100 SewageSystemLevelTwoUpgrade Class Reference

Represents the second level upgrade to a SewageSystem entity.

`#include <SewageSystemLevelTwoUpgrade.h>`

Inheritance diagram for SewageSystemLevelTwoUpgrade:



Public Member Functions

- SewageSystemLevelTwoUpgrade (SewageSystem *sewage)
  Constructs a SewageSystemLevelTwoUpgrade object.
- SewageSystemLevelTwoUpgrade (SewageSystemLevelTwoUpgrade *sSLTU)
  Copy constructor for SewageSystemLevelTwoUpgrade.
- ~SewageSystemLevelTwoUpgrade ()
  Destructor for SewageSystemLevelTwoUpgrade.
- void update () override
  Updates the state of the upgraded sewage system.
- Entity * clone () override
  Clones the current SewageSystemLevelTwoUpgrade object.
- Entity * upgrade () override
  Upgrades the current utility to the next level.
- int getOutput () override
  Retrieves the upgraded sewage system's output.
- Cost getCost () override
  Retrieves the cost of the utility or its upgraded version.
- int getLevel () override
  Gets the level of the sewage system upgrade.

Public Member Functions inherited from SewageSystemUpgrade

- SewageSystemUpgrade (SewageSystem *sewage)
  Constructs a SewageSystemUpgrade object based on an existing SewageSystem.
- SewageSystemUpgrade (SewageSystemUpgrade *sSU)
  Copy constructor for the SewageSystemUpgrade class.
- virtual ~**SewageSystemUpgrade** ()
  Destructor for the SewageSystemUpgrade object.

Public Member Functions inherited from SewageSystem

- SewageSystem (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a SewageSystem object with specified attributes.
- SewageSystem (SewageSystem ∗sewage)

  Copy constructor for the SewageSystem class.
- virtual ∼**SewageSystem** ()

  Destructor for the SewageSystem object.

Public Member Functions inherited from Utility

- Utility (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a Utility object with the specified parameters.
- Utility (Utility ∗utility)

  Copy constructor for the Utility class.
- virtual ∼**Utility** ()

  Destructor for the Utility object.
- void setOutput (int output)

  Sets the output value of the utility.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity ∗entity)

  Copy constructor for the Entity class.
- virtual ∼**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity ∗entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

Sets the symbol of the entity.

- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from SewageSystemUpgrade

- SewageSystem * **sewageSystem**

  Pointer to the original SewageSystem that is being upgraded.

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).

- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.100.1 Detailed Description

Represents the second level upgrade to a SewageSystem entity.

The SewageSystemLevelTwoUpgrade class enhances the base functionality of a SewageSystem by increasing its output. This class is the second upgrade level in a series of potential sewage system improvements.

4.100.2 Constructor & Destructor Documentation

4.100.2.1 SewageSystemLevelTwoUpgrade() [1/2]

```
SewageSystemLevelTwoUpgrade::SewageSystemLevelTwoUpgrade (
            SewageSystem * sewage)
```

Constructs a SewageSystemLevelTwoUpgrade object.

Initializes the upgrade by enhancing the specified SewageSystem with a level two upgrade.

Parameters

sewage	Pointer to the original SewageSystem to be upgraded.

4.100.2.2 SewageSystemLevelTwoUpgrade() [2/2]

```
SewageSystemLevelTwoUpgrade::SewageSystemLevelTwoUpgrade (
            SewageSystemLevelTwoUpgrade * sSLTU)
```

Copy constructor for SewageSystemLevelTwoUpgrade.

Creates a new SewageSystemLevelTwoUpgrade object by copying the attributes of an existing SewageSystemLevelTwoUpgrade object.

Parameters

sSLTU	Pointer to the existing SewageSystemLevelTwoUpgrade to be copied.

4.100.2.3 ~SewageSystemLevelTwoUpgrade()

```
SewageSystemLevelTwoUpgrade::~SewageSystemLevelTwoUpgrade ()
```

Destructor for SewageSystemLevelTwoUpgrade.

Cleans up any resources associated with the upgrade.

4.100.3 Member Function Documentation

4.100.3.1 clone()

```
Entity * SewageSystemLevelTwoUpgrade::clone ()  [override], [virtual]
```

Clones the current SewageSystemLevelTwoUpgrade object.

Creates a new instance of SewageSystemLevelTwoUpgrade with the same attributes as the current object.

Returns

A pointer to the newly cloned SewageSystemLevelTwoUpgrade object.

Implements SewageSystemUpgrade.

4.100.3.2 getCost()

```
Cost SewageSystemLevelTwoUpgrade::getCost ()  [override], [virtual]
```

Retrieves the cost of the utility or its upgraded version.

Returns

A Cost object representing the monetary and material costs.

Implements SewageSystemUpgrade.

4.100.3.3 getLevel()

```
int SewageSystemLevelTwoUpgrade::getLevel ()  [override], [virtual]
```

Gets the level of the sewage system upgrade.

Returns

The level of the sewage system upgrade.

Reimplemented from Utility.

4.100.3.4 getOutput()

```
int SewageSystemLevelTwoUpgrade::getOutput ()  [override], [virtual]
```

Retrieves the upgraded sewage system's output.

Returns the sewage system output after the level two upgrade.

Returns

The updated output as an integer.

Implements SewageSystemUpgrade.

4.100.3.5 update()

```
void SewageSystemLevelTwoUpgrade::update ()  [override], [virtual]
```

Updates the state of the upgraded sewage system.

Implements specific behavior for the sewage system after applying the level two upgrade.

Implements SewageSystemUpgrade.

4.100.3.6 upgrade()

```
Entity * SewageSystemLevelTwoUpgrade::upgrade ()  [override], [virtual]
```

Upgrades the current utility to the next level.

Returns

A pointer to the upgraded utility instance, or nullptr if already at maximum level.

Implements SewageSystemUpgrade.

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

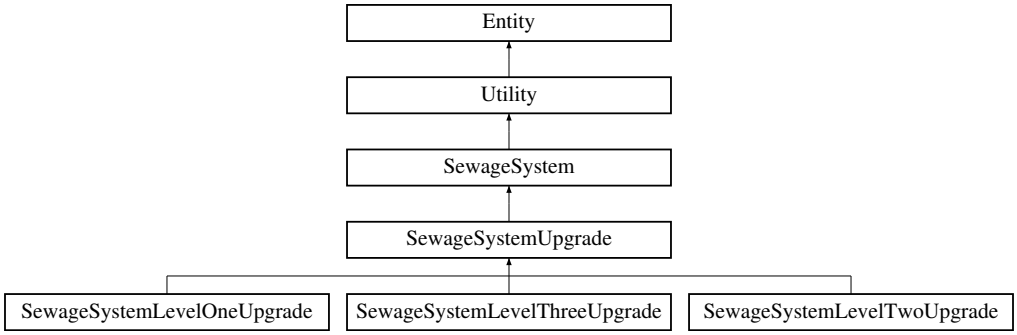
- src/entities/utility/sewagesystem/SewageSystemLevelTwoUpgrade.h
- src/entities/utility/sewagesystem/SewageSystemLevelTwoUpgrade.cpp

4.101 SewageSystemUpgrade Class Reference

Represents an upgrade to a SewageSystem entity in the city builder simulation.

`#include <SewageSystemUpgrade.h>`

Inheritance diagram for SewageSystemUpgrade:



Public Member Functions

- SewageSystemUpgrade (SewageSystem *sewage)

  Constructs a SewageSystemUpgrade object based on an existing SewageSystem.
- SewageSystemUpgrade (SewageSystemUpgrade *sSU)

  Copy constructor for the SewageSystemUpgrade class.
- virtual ~**SewageSystemUpgrade** ()

  Destructor for the SewageSystemUpgrade object.
- virtual void update ()=0

  Pure virtual function to update the upgraded sewage system.
- virtual Entity * clone ()=0

  Pure virtual function to clone the upgraded sewage system.
- virtual Entity * upgrade ()=0

  Upgrades the current utility to the next level.
- virtual int getOutput ()=0

  Retrieves the output of the upgraded sewage system.
- virtual Cost getCost ()=0

  Retrieves the cost of the utility or its upgraded version.

Public Member Functions inherited from SewageSystem

- SewageSystem (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a SewageSystem object with specified attributes.
- SewageSystem (SewageSystem *sewage)

  Copy constructor for the SewageSystem class.
- virtual ~**SewageSystem** ()

  Destructor for the SewageSystem object.

Public Member Functions inherited from Utility

- Utility (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a Utility object with the specified parameters.
- Utility (Utility *utility)

  Copy constructor for the Utility class.
- virtual ~**Utility** ()

  Destructor for the Utility object.
- void setOutput (int output)

  Sets the output value of the utility.
- virtual int getLevel ()

  Gets the level of the utility.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

Unsubscribes this entity from all buildings it is observing.

- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Protected Attributes

- SewageSystem * **sewageSystem**

  Pointer to the original SewageSystem that is being upgraded.

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).
- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.101.1 Detailed Description

Represents an upgrade to a SewageSystem entity in the city builder simulation.

The SewageSystemUpgrade class extends the functionality of a SewageSystem, enhancing its capabilities and acting as a wrapper around the existing SewageSystem object.

4.101.2 Constructor & Destructor Documentation

4.101.2.1 SewageSystemUpgrade() [1/2]

```
SewageSystemUpgrade::SewageSystemUpgrade (
            SewageSystem * sewage)
```

Constructs a SewageSystemUpgrade object based on an existing SewageSystem.

Initializes the upgrade with a reference to an existing SewageSystem, enhancing its features.

Parameters

sewage	Pointer to the SewageSystem being upgraded.

4.101.2.2 SewageSystemUpgrade() [2/2]

```
SewageSystemUpgrade::SewageSystemUpgrade (
            SewageSystemUpgrade * sSU)
```

Copy constructor for the SewageSystemUpgrade class.

Creates a new SewageSystemUpgrade object by copying the attributes of an existing SewageSystemUpgrade.

Parameters

sSU	Pointer to the existing SewageSystemUpgrade object to be copied.

4.101.3 Member Function Documentation

4.101.3.1 clone()

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

Pure virtual function to clone the upgraded sewage system.

Returns

A pointer to a new cloned SewageSystemUpgrade object.

Reimplemented from SewageSystem.

Implemented in SewageSystemLevelOneUpgrade, SewageSystemLevelThreeUpgrade, and SewageSystemLevelTwoUpgrade.

4.101.3.2 getCost()

```
virtual Cost SewageSystemUpgrade::getCost ()  [pure virtual]
```

Retrieves the cost of the utility or its upgraded version.

Returns

A Cost object representing the monetary and material costs.

Reimplemented from Utility.

Implemented in SewageSystemLevelOneUpgrade, SewageSystemLevelThreeUpgrade, and SewageSystemLevelTwoUpgrade.

4.101.3.3 getOutput()

```
virtual int SewageSystemUpgrade::getOutput ()  [pure virtual]
```

Retrieves the output of the upgraded sewage system.

Returns

The output value as an integer.

Reimplemented from Utility.

Implemented in SewageSystemLevelOneUpgrade, SewageSystemLevelThreeUpgrade, and SewageSystemLevelTwoUpgrade.

4.101.3.4 update()

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

Pure virtual function to update the upgraded sewage system.

Reimplemented from SewageSystem.

Implemented in SewageSystemLevelOneUpgrade, SewageSystemLevelThreeUpgrade, and SewageSystemLevelTwoUpgrade.

4.101.3.5 upgrade()

```
virtual Entity * SewageSystemUpgrade::upgrade ()  [pure virtual]
```

Upgrades the current utility to the next level.

Returns

A pointer to the upgraded utility instance, or nullptr if already at maximum level.

Reimplemented from SewageSystem.

Implemented in SewageSystemLevelOneUpgrade, SewageSystemLevelThreeUpgrade, and SewageSystemLevelTwoUpgrade.

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

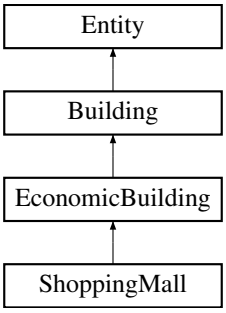
- src/entities/utility/sewagesystem/SewageSystemUpgrade.h
- src/entities/utility/sewagesystem/SewageSystemUpgrade.cpp

4.102 ShoppingMall Class Reference

Concrete class representing a shopping mall in the city builder/manager game.

`#include <ShoppingMall.h>`

Inheritance diagram for ShoppingMall:



Public Member Functions

- ShoppingMall (EntityConfig ec, Size size, int xPos, int yPos)
  Parameterized constructor for the ShoppingMall class.
- ShoppingMall (ShoppingMall ∗mall)
  Copy constructor for the ShoppingMall class.
- ∼ShoppingMall ()
  Destructor for the ShoppingMall class.
- void update ()
  Updates the state of the shopping mall entity.
- Entity ∗ clone ()
  Clones the shopping mall entity.

Public Member Functions inherited from EconomicBuilding

- EconomicBuilding (EntityConfig ec, Size size, int xPos, int yPos)
  Parameterized constructor for the EconomicBuilding class.
- EconomicBuilding (EconomicBuilding ∗economic)
  Copy constructor for the EconomicBuilding class.
- virtual ∼EconomicBuilding ()
  Destructor for the EconomicBuilding class.

Public Member Functions inherited from Building

- Building (EntityConfig ec, Size size, int xPos, int yPos)
  Parameterized constructor for the Building class.
- Building (Building ∗building)
  Copy constructor for the Building class.
- virtual ∼Building ()
  Destructor for the Building class.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity ∗entity)

  Copy constructor for the Entity class.
- virtual ∼**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity ∗entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity ∗entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity ∗entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity ∗ > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).
- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.102.1 Detailed Description

Concrete class representing a shopping mall in the city builder/manager game.

ShoppingMall is a type of EconomicBuilding that provides retail spaces and attracts customers for shopping activities.

4.102.2 Constructor & Destructor Documentation

4.102.2.1 ShoppingMall() [1/2]

```
ShoppingMall::ShoppingMall (
            EntityConfig ec,
            Size size,
            int xPos,
            int yPos)
```

Parameterized constructor for the ShoppingMall class.

Parameters

ec	The configuration object containing general entity properties.
size	The size of the shopping mall entity.
xPos	The x-coordinate position of the shopping mall on the map.
yPos	The y-coordinate position of the shopping mall on the map.

Initializes a new instance of the ShoppingMall class with specific values.

4.102.2.2 ShoppingMall() [2/2]

```
ShoppingMall::ShoppingMall (
            ShoppingMall * mall)
```

Copy constructor for the ShoppingMall class.

Parameters

mall	A pointer to an existing ShoppingMall object to copy from.

Creates a new ShoppingMall instance as a copy of the provided object.

4.102.2.3 ~ShoppingMall()

```
ShoppingMall::~ShoppingMall ()
```

Destructor for the ShoppingMall class.

Ensures proper cleanup of resources when a ShoppingMall object is destroyed.

4.102.3 Member Function Documentation

4.102.3.1 clone()

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

Clones the shopping mall entity.

Returns a deep copy of the current ShoppingMall object.

Returns

A pointer to the newly cloned ShoppingMall entity.

Implements EconomicBuilding.

4.102.3.2 update()

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

Updates the state of the shopping mall entity.

This function handles changes in the mall's state.

Implements EconomicBuilding.

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

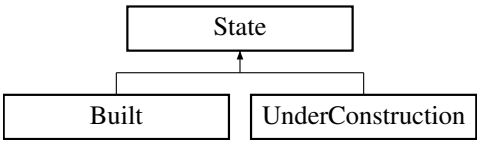
- src/entities/building/economic/ShoppingMall.h
- src/entities/building/economic/ShoppingMall.cpp

4.103 State Class Reference

Abstract base class representing the state of an entity.

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

Inheritance diagram for State:



Public Member Functions

- State (int buildTime)

  Constructs a State with the specified build time.
- State (State *state)

  Copy constructor for the State class.
- virtual ~**State** ()

  Destructor for the State.
- virtual State * update ()=0

  Updates the current state.
- virtual State * clone ()=0

  Creates a deep copy of the current State object.
- int getGameLoopCounter ()

  Gets the current game loop counter.
- int getBuildTime ()

  Gets the build time of the state.
- void **incrementGameLoopCounter** ()

  Increments the game loop counter.

4.103.1 Detailed Description

Abstract base class representing the state of an entity.

The State class defines the interface for different states an entity can have. Derived classes must implement the update and initialize methods to handle state transitions.

4.103.2 Constructor & Destructor Documentation

4.103.2.1 State() [1/2]

```
State::State (
            int buildTime)
```

Constructs a State with the specified build time.

Parameters

buildTime	The time required to build the entity.

4.103.2.2 State() [2/2]

```
State::State (
            State * state)
```

Copy constructor for the State class.

Creates a new State by copying the attributes of an existing State object.

Parameters

state	Pointer to the existing State object to be copied.

4.103.3 Member Function Documentation

4.103.3.1 clone()

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

Creates a deep copy of the current State object.

This method is responsible for cloning the concrete subclass of State. This allows for proper polymorphic copying of abstract State objects.

Returns

A pointer to a new State object that is a copy of the current instance.

Implemented in Built, and UnderConstruction.

4.103.3.2 getBuildTime()

```
int State::getBuildTime ()
```

Gets the build time of the state.

Returns

The time required to complete the build.

4.103.3.3 getGameLoopCounter()

```
int State::getGameLoopCounter ()
```

Gets the current game loop counter.

Returns

The current game loop counter.

4.103.3.4 update()

```
virtual State * State::update ()  [pure virtual]
```

Updates the current state.

Returns

A pointer to the next state after update.

Implemented in Built, and UnderConstruction.

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

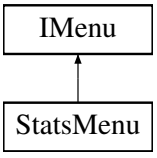
- src/entities/state/State.h
- src/entities/state/State.cpp

4.104 StatsMenu Class Reference

Provides functionality for displaying city statistics and various entity listings.

```
#include <StatsMenu.h>
```

Inheritance diagram for StatsMenu:



Public Member Functions

- StatsMenu ()

  Constructs a StatsMenu object.
- ~StatsMenu ()

  Destructor for StatsMenu.
- void display () const override

  Displays the statistics menu.
- void handleInput () override

  Handles user input in the statistics menu.

Public Member Functions inherited from IMenu

- **IMenu** ()=default

  Default constructor for IMenu.
- IMenu (std::string heading)

  Constructor to initialize a menu with a specified heading.
- virtual ~**IMenu** ()=default

  Virtual destructor for IMenu.
- void setHeading (const std::string &heading)

  Sets the heading of the menu.

Additional Inherited Members

Protected Member Functions inherited from IMenu

- std::string repeat (const std::string &str, int times) const

  Utility function to repeat a string multiple times.
- int calculateMaxWidth (const std::string &menuHeading, const std::vector< Section > &sections) const

  Calculates the maximum width required for the menu.
- void printTopBorder (int width) const

  Prints the top border of the menu using box-drawing characters.
- void printBottomBorder (int width) const

  Prints the bottom border of the menu using box-drawing characters.
- void printSectionDivider (int width) const

  Prints a section divider in the menu using box-drawing characters.
- void printDoubleLineDivider (int width) const

  Prints a double-line divider for the main heading of the menu.
- std::string centerText (const std::string &text, int width) const

  Centers text within a specified width using space padding.
- std::string centerTextWithChar (const std::string &text, int width, const std::string &padChar) const

  Centers text within a specified width using a custom character for padding.
- void displayMenu () const

  Displays the formatted menu, including sections and options.
- void **displayChoicePrompt** () const

  Displays the choice prompt for user input.
- void displayChoiceMessagePrompt (const std::string &message) const

  Displays a custom message prompt for user input.
- void **displayInvalidChoice** () const

Displays an error message when the user makes an invalid choice.

- void displayErrorMessage (const std::string &message) const

  Displays a general error message.
- void displaySuccessMessage (const std::string &message) const

  Displays a success message in green color.
- void **displayPressEnterToContinue** () const

  Displays a message asking the user to press Enter to continue.
- void **clearScreen** () const

  Clears the terminal screen.
- std::string stripColorCodes (const std::string &input) const

  Strips ANSI color codes from a string.
- virtual void displayAvailablePositions (const std::vector< std::vector< int > > &positions) const

  Displays available positions on the city grid for an entity.

Static Protected Member Functions inherited from IMenu

- static char indexToExtendedChar (int index)

  Converts a numeric index (0-99) to a single character in an extended set.
- static std::string coordinatesToLabel (int x, int y)

  Converts x and y coordinates to a labeled string (e.g., "A, 1").

Protected Attributes inherited from IMenu

- std::vector< Section > **sections**

  List of sections contained in the menu.
- std::string **menuHeading**

  The heading/title of the menu.
- bool **hasExited**

  Flag indicating if the menu has been exited.
- CityManager **cityManager**

  Manager for city-related operations.
- bool **displayResources**

  Flag indicating whether to display resources in the menu.
- bool **isInfoMenu**

  Flag indicating whether to display option numbers.

Static Protected Attributes inherited from IMenu

- static const char * **RESET** = "\033[0m"

  ANSI color codes and styles for use in all menus.
- static const char * **BOLD_WHITE** = "\033[1;37m"
- static const char * **NORMAL_WHITE** = "\033[0;37m"
- static const char * **DARK_GRAY** = "\033[1;30m"
- static const char * **BOLD_YELLOW** = "\033[1;33m"
- static const char * **BOLD_GREEN** = "\033[1;32m"
- static const char * **BOLD_RED** = "\033[1;31m"
- static const char * **BOLD_CYAN** = "\033[1;36m"
- static const char * **BLUE** = "\033[34m"

4.104.1 Detailed Description

Provides functionality for displaying city statistics and various entity listings.

The StatsMenu class includes options to view general city statistics, list all city entities, and display detailed information for specific types of buildings, utilities, and producers.

4.104.2 Constructor & Destructor Documentation

4.104.2.1 StatsMenu()

```
StatsMenu::StatsMenu ()
```

Constructs a StatsMenu object.

Initializes the StatsMenu with relevant headings and sections.

4.104.2.2 ~StatsMenu()

```
StatsMenu::~StatsMenu ()
```

Destructor for StatsMenu.

Cleans up resources used by the StatsMenu.

4.104.3 Member Function Documentation

4.104.3.1 display()

```
void StatsMenu::display () const  [override], [virtual]
```

Displays the statistics menu.

Overrides the display method of IMenu to show city statistics and available entity listings.

Implements IMenu.

4.104.3.2 handleInput()

```
void StatsMenu::handleInput ()  [override], [virtual]
```

Handles user input in the statistics menu.

Processes user selections for viewing various statistics and entity lists.

Implements IMenu.

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

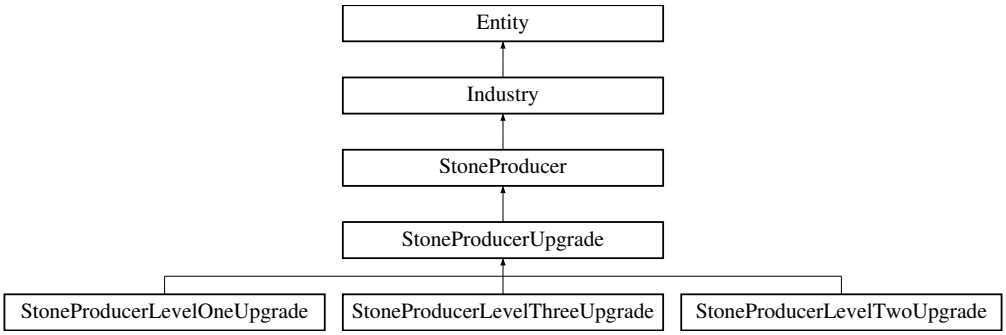
- src/menus/stats/StatsMenu.h
- src/menus/stats/StatsMenu.cpp

4.105 StoneProducer Class Reference

Represents a stone producer entity in the industry.

`#include <StoneProducer.h>`

Inheritance diagram for StoneProducer:



Public Member Functions

- StoneProducer (StoneProducer *stoneProducer)

  Constructs a StoneProducer from another StoneProducer instance.
- StoneProducer (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a StoneProducer with specified configuration and position.
- virtual ~**StoneProducer** ()

  Destructor for the StoneProducer class.
- void update () override

  Updates the state of the stone producer.
- Entity * clone () override

  Clones the current StoneProducer instance.
- Entity * upgrade () override

  Upgrades the stone producer to the next level.

Public Member Functions inherited from Industry

- Industry (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Industry entity with specified attributes.
- Industry (Industry *industry)

  Copy constructor for the Industry class.
- virtual ~**Industry** ()

  Virtual destructor for the Industry class.
- virtual int getOutput ()

  Gets the production output of the industry.
- void setOutput (int output)

  Sets the production output of the industry.
- virtual int getLevel ()

  Gets the current level of the industry.
- virtual Cost getCost ()

  Gets the cost of an upgrade.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).
- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.105.1 Detailed Description

Represents a stone producer entity in the industry.

This class is responsible for producing stone resources and managing its state.

4.105.2 Constructor & Destructor Documentation

4.105.2.1 StoneProducer() [1/2]

```
StoneProducer::StoneProducer (
            StoneProducer * stoneProducer)
```

Constructs a StoneProducer from another StoneProducer instance.

Parameters

stoneProducer	Pointer to the StoneProducer to copy.

4.105.2.2 StoneProducer() [2/2]

```
StoneProducer::StoneProducer (
            EntityConfig ec,
            Size size,
            int xPos,
            int yPos)
```

Constructs a StoneProducer with specified configuration and position.

Parameters

ec	Entity configuration for the stone producer.
size	Size of the producer entity.
xPos	X position in the grid.
yPos	Y position in the grid.

4.105.3 Member Function Documentation

4.105.3.1 clone()

```
Entity * StoneProducer::clone ()  [override], [virtual]
```

Clones the current StoneProducer instance.

Returns

A pointer to a new StoneProducer that is a copy of this instance.

Implements Industry.

Reimplemented in StoneProducerLevelOneUpgrade, StoneProducerLevelThreeUpgrade, StoneProducerLevelTwoUpgrade, and StoneProducerUpgrade.

4.105.3.2 update()

```
void StoneProducer::update ()  [override], [virtual]
```

Updates the state of the stone producer.

This method notifies observers and updates the build state if necessary.

Implements Industry.

Reimplemented in StoneProducerLevelOneUpgrade, StoneProducerLevelThreeUpgrade, StoneProducerLevelTwoUpgrade, and StoneProducerUpgrade.

4.105.3.3 upgrade()

```
Entity * StoneProducer::upgrade ()  [override], [virtual]
```

Upgrades the stone producer to the next level.

Returns

A pointer to the upgraded StoneProducerLevelOneUpgrade instance.

Implements Industry.

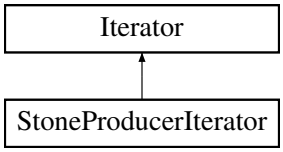
Reimplemented in StoneProducerLevelOneUpgrade, StoneProducerLevelThreeUpgrade, StoneProducerLevelTwoUpgrade, and StoneProducerUpgrade.

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

- src/entities/industry/stoneproducer/StoneProducer.h
- src/entities/industry/stoneproducer/StoneProducer.cpp

4.106 StoneProducerIterator Class Reference

Inheritance diagram for StoneProducerIterator:



Public Member Functions

- **StoneProducerIterator** ()

  Construct a new Stone Producer Iterator:: Stone Producer Iterator object.
- ~**StoneProducerIterator** ()

  Destroy the Stone Producer Iterator:: Stone Producer Iterator object.
- StoneProducerIterator (std::vector< std::vector< Entity * > > &grid)

  Construct a new Stone Producer Iterator:: Stone Producer Iterator object.
- void first ()

  Sets the iterator to the first unvisited StoneProducer.
- void next ()

  Advances to the next unvisited StoneProducer.
- bool hasNext ()

  Checks if there is another unvisited StoneProducer.
- Entity * current ()

  Returns the current StoneProducer.

Public Member Functions inherited from Iterator

- **Iterator** ()

  Construct a new Iterator object, initializing row and column to zero.
- virtual ~**Iterator** ()

  Destroy the Iterator object.
- **Iterator** (std::vector< std::vector< Entity * > > &grid)
- virtual int getRow ()

  Get the current row index of the iterator.
- virtual int getCol ()

  Get the current column index of the iterator.

Additional Inherited Members

Protected Member Functions inherited from Iterator

- bool isVisited (Entity *entity)

  Check if the specified entity has been visited.
- void markVisited (Entity *entity)

  Mark the specified entity as visited.

Protected Attributes inherited from Iterator

- std::vector< std::vector< Entity * > > **grid**
- std::vector< std::vector< Entity * > >::iterator **currRow**
- std::vector< Entity * >::iterator **curr**
- int **row**
- int **col**
- std::unordered_set< Entity * > **visitedEntities**

4.106.1 Constructor & Destructor Documentation

4.106.1.1 StoneProducerIterator()

```
StoneProducerIterator::StoneProducerIterator (
            std::vector< std::vector< Entity * > > & grid)
```

Construct a new Stone Producer Iterator:: Stone Producer Iterator object.

Parameters

grid	

4.106.2 Member Function Documentation

4.106.2.1 current()

```
Entity * StoneProducerIterator::current ()  [virtual]
```

Returns the current StoneProducer.

Returns

Entity∗

Implements Iterator.

4.106.2.2 first()

```
void StoneProducerIterator::first ()  [virtual]
```

Sets the iterator to the first unvisited StoneProducer.

Implements Iterator.

4.106.2.3 hasNext()

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

Checks if there is another unvisited StoneProducer.

Returns

true if there is another unvisited StoneProducer, false otherwise

Implements Iterator.

4.106.2.4 next()

```
void StoneProducerIterator::next ()  [virtual]
```

Advances to the next unvisited StoneProducer.

Implements Iterator.

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

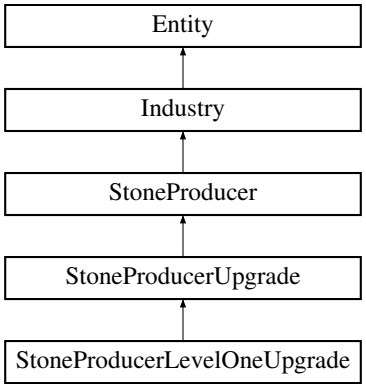
- src/iterators/industry/StoneProducerIterator.h
- src/iterators/industry/StoneProducerIterator.cpp

4.107 StoneProducerLevelOneUpgrade Class Reference

Represents a level one upgrade for a stone producer.

`#include <StoneProducerLevelOneUpgrade.h>`

Inheritance diagram for StoneProducerLevelOneUpgrade:



Public Member Functions

- StoneProducerLevelOneUpgrade (StoneProducer *stoneProd)

  Constructs a StoneProducerLevelOneUpgrade from a given StoneProducer.
- StoneProducerLevelOneUpgrade (StoneProducerLevelOneUpgrade *stoneProd)

  Constructs a StoneProducerLevelOneUpgrade from another StoneProducerLevelOneUpgrade instance.
- ~**StoneProducerLevelOneUpgrade** ()

  Destructor for the StoneProducerLevelOneUpgrade class.
- int getOutput () override

  Gets the output of the stone producer upgrade.
- int getLevel () override

  Gets the level of the upgrade.
- Entity * clone () override

  Clones the current StoneProducerLevelOneUpgrade instance.
- void update () override

  Updates the state of the stone producer upgrade.
- Entity * upgrade () override

  Upgrades the stone producer to the next level.
- Cost getCost () override

  Gets the cost of the stone producer upgrade.

Public Member Functions inherited from StoneProducerUpgrade

- StoneProducerUpgrade (StoneProducer *stoneProd)

  Constructs a StoneProducerUpgrade from a given StoneProducer.
- StoneProducerUpgrade (StoneProducerUpgrade *stoneProd)

  Constructs a StoneProducerUpgrade from another StoneProducerUpgrade instance.
- virtual ~**StoneProducerUpgrade** ()

  Destructor for the StoneProducerUpgrade class.

Public Member Functions inherited from StoneProducer

- StoneProducer (StoneProducer *stoneProducer)

  Constructs a StoneProducer from another StoneProducer instance.
- StoneProducer (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a StoneProducer with specified configuration and position.
- virtual ~**StoneProducer** ()

  Destructor for the StoneProducer class.

Public Member Functions inherited from Industry

- Industry (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Industry entity with specified attributes.
- Industry (Industry *industry)

  Copy constructor for the Industry class.
- virtual ~**Industry** ()

  Virtual destructor for the Industry class.
- void setOutput (int output)

  Sets the production output of the industry.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

Sets the symbol of the entity.

- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from StoneProducerUpgrade

- StoneProducer * **stoneProducer**

  Pointer to the base stone producer being upgraded.

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).

- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.107.1 Detailed Description

Represents a level one upgrade for a stone producer.

This class extends the StoneProducerUpgrade class to provide functionality for a level one upgrade of a stone producer.

4.107.2 Constructor & Destructor Documentation

4.107.2.1 StoneProducerLevelOneUpgrade() [1/2]

```
StoneProducerLevelOneUpgrade::StoneProducerLevelOneUpgrade (
            StoneProducer * stoneProd)
```

Constructs a StoneProducerLevelOneUpgrade from a given StoneProducer.

Parameters

stoneProd	Pointer to the StoneProducer to upgrade.

4.107.2.2 StoneProducerLevelOneUpgrade() [2/2]

```
StoneProducerLevelOneUpgrade::StoneProducerLevelOneUpgrade (
            StoneProducerLevelOneUpgrade * stoneProd)
```

Constructs a StoneProducerLevelOneUpgrade from another StoneProducerLevelOneUpgrade instance.

Parameters

stoneProd	Pointer to the StoneProducerLevelOneUpgrade to copy.

4.107.3 Member Function Documentation

4.107.3.1 clone()

```
Entity * StoneProducerLevelOneUpgrade::clone ()  [override], [virtual]
```

Clones the current StoneProducerLevelOneUpgrade instance.

Returns

A pointer to a new StoneProducerLevelOneUpgrade that is a copy of this instance.

Implements StoneProducerUpgrade.

4.107.3.2 getCost()

```
Cost StoneProducerLevelOneUpgrade::getCost ()  [override], [virtual]
```

Gets the cost of the stone producer upgrade.

Returns

The cost associated with the upgrade.

Implements StoneProducerUpgrade.

4.107.3.3 getLevel()

```
int StoneProducerLevelOneUpgrade::getLevel ()  [override], [virtual]
```

Gets the level of the upgrade.

Returns

The level of the upgrade, which is 1 for this class.

Reimplemented from Industry.

4.107.3.4 getOutput()

```
int StoneProducerLevelOneUpgrade::getOutput ()  [override], [virtual]
```

Gets the output of the stone producer upgrade.

Returns

The output value of the upgraded producer.

Implements StoneProducerUpgrade.

4.107.3.5 update()

```
void StoneProducerLevelOneUpgrade::update ()  [override], [virtual]
```

Updates the state of the stone producer upgrade.

Implements StoneProducerUpgrade.

4.107.3.6 upgrade()

```
Entity * StoneProducerLevelOneUpgrade::upgrade ()  [override], [virtual]
```

Upgrades the stone producer to the next level.

Returns

A pointer to the upgraded StoneProducerLevelTwoUpgrade instance.

Implements StoneProducerUpgrade.

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

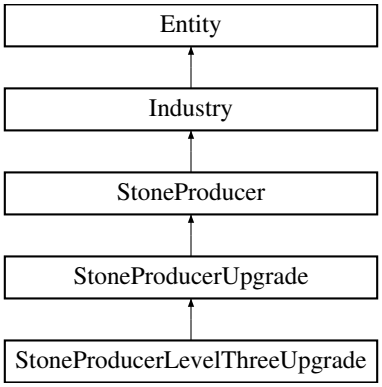
- src/entities/industry/stoneproducer/StoneProducerLevelOneUpgrade.h
- src/entities/industry/stoneproducer/StoneProducerLevelOneUpgrade.cpp

4.108 StoneProducerLevelThreeUpgrade Class Reference

Represents a level three upgrade for a stone producer.

```
#include <StoneProducerLevelThreeUpgrade.h>
```

Inheritance diagram for StoneProducerLevelThreeUpgrade:



Public Member Functions

- StoneProducerLevelThreeUpgrade (StoneProducer *stoneProd)

  Constructs a StoneProducerLevelThreeUpgrade from a given StoneProducer.
- StoneProducerLevelThreeUpgrade (StoneProducerLevelThreeUpgrade *stoneProd)

  Constructs a StoneProducerLevelThreeUpgrade from another StoneProducerLevelThreeUpgrade instance.
- ~**StoneProducerLevelThreeUpgrade** ()

  Destructor for the StoneProducerLevelThreeUpgrade class.
- void update () override

  Updates the state of the stone producer upgrade.
- int getOutput () override

  Gets the output of the stone producer upgrade.
- int getLevel () override

  Gets the level of the upgrade.
- Entity * clone () override

  Clones the current StoneProducerLevelThreeUpgrade instance.
- Entity * upgrade () override

  Upgrades the stone producer to the next level.
- Cost getCost () override

  Gets the cost of the stone producer upgrade.

Public Member Functions inherited from StoneProducerUpgrade

- StoneProducerUpgrade (StoneProducer *stoneProd)

  Constructs a StoneProducerUpgrade from a given StoneProducer.
- StoneProducerUpgrade (StoneProducerUpgrade *stoneProd)

  Constructs a StoneProducerUpgrade from another StoneProducerUpgrade instance.
- virtual ~**StoneProducerUpgrade** ()

  Destructor for the StoneProducerUpgrade class.

Public Member Functions inherited from StoneProducer

- StoneProducer (StoneProducer *stoneProducer)

  Constructs a StoneProducer from another StoneProducer instance.
- StoneProducer (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a StoneProducer with specified configuration and position.
- virtual ~**StoneProducer** ()

  Destructor for the StoneProducer class.

Public Member Functions inherited from Industry

- Industry (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Industry entity with specified attributes.
- Industry (Industry *industry)

  Copy constructor for the Industry class.
- virtual ~**Industry** ()

  Virtual destructor for the Industry class.
- void setOutput (int output)

  Sets the production output of the industry.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from StoneProducerUpgrade

- StoneProducer * **stoneProducer**

  Pointer to the base stone producer being upgraded.

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).
- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.108.1 Detailed Description

Represents a level three upgrade for a stone producer.

This class extends the StoneProducerUpgrade class to provide functionality for a level three upgrade of a stone producer.

4.108.2 Constructor & Destructor Documentation

4.108.2.1 StoneProducerLevelThreeUpgrade() [1/2]

```
StoneProducerLevelThreeUpgrade::StoneProducerLevelThreeUpgrade (
            StoneProducer * stoneProd)
```

Constructs a StoneProducerLevelThreeUpgrade from a given StoneProducer.

Parameters

stoneProd	Pointer to the StoneProducer to upgrade.

4.108.2.2 StoneProducerLevelThreeUpgrade() [2/2]

```
StoneProducerLevelThreeUpgrade::StoneProducerLevelThreeUpgrade (
            StoneProducerLevelThreeUpgrade * stoneProd)
```

Constructs a StoneProducerLevelThreeUpgrade from another StoneProducerLevelThreeUpgrade instance.

Parameters

stoneProd	Pointer to the StoneProducerLevelThreeUpgrade to copy.

4.108.3 Member Function Documentation

4.108.3.1 clone()

```
Entity * StoneProducerLevelThreeUpgrade::clone ()  [override], [virtual]
```

Clones the current StoneProducerLevelThreeUpgrade instance.

Returns

A pointer to a new StoneProducerLevelThreeUpgrade that is a copy of this instance.

Implements StoneProducerUpgrade.

4.108.3.2 getCost()

```
Cost StoneProducerLevelThreeUpgrade::getCost ()  [override], [virtual]
```

Gets the cost of the stone producer upgrade.

Returns

The cost associated with the upgrade.

Implements StoneProducerUpgrade.

4.108.3.3 getLevel()

```
int StoneProducerLevelThreeUpgrade::getLevel ()  [override], [virtual]
```

Gets the level of the upgrade.

Returns

The level of the upgrade, which is 3 for this class.

Reimplemented from Industry.

4.108.3.4 getOutput()

```
int StoneProducerLevelThreeUpgrade::getOutput ()  [override], [virtual]
```

Gets the output of the stone producer upgrade.

Returns

The output value of the upgraded producer.

Implements StoneProducerUpgrade.

4.108.3.5 update()

```
void StoneProducerLevelThreeUpgrade::update ()  [override], [virtual]
```

Updates the state of the stone producer upgrade.

Implements StoneProducerUpgrade.

4.108.3.6 upgrade()

```
Entity * StoneProducerLevelThreeUpgrade::upgrade ()  [override], [virtual]
```

Upgrades the stone producer to the next level.

Returns

A pointer to the next upgraded instance, or nullptr if at the highest level.

Implements StoneProducerUpgrade.

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

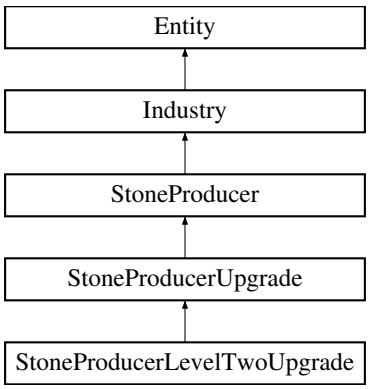
- src/entities/industry/stoneproducer/StoneProducerLevelThreeUpgrade.h
- src/entities/industry/stoneproducer/StoneProducerLevelThreeUpgrade.cpp

4.109 StoneProducerLevelTwoUpgrade Class Reference

Represents a level two upgrade for a stone producer.

```
#include <StoneProducerLevelTwoUpgrade.h>
```

Inheritance diagram for StoneProducerLevelTwoUpgrade:



Public Member Functions

- StoneProducerLevelTwoUpgrade (StoneProducer ∗stoneProd)

  Constructs a StoneProducerLevelTwoUpgrade from a given StoneProducer.
- StoneProducerLevelTwoUpgrade (StoneProducerLevelTwoUpgrade ∗stoneProd)

  Constructs a StoneProducerLevelTwoUpgrade from another StoneProducerLevelTwoUpgrade instance.
- ∼**StoneProducerLevelTwoUpgrade** ()

  Destructor for the StoneProducerLevelTwoUpgrade class.
- Entity ∗ clone () override

  Clones the current StoneProducerLevelTwoUpgrade instance.
- void update () override

  Updates the state of the stone producer upgrade.
- int getOutput () override

  Gets the output of the stone producer upgrade.
- int getLevel () override

  Gets the level of the upgrade.
- Entity ∗ upgrade () override

  Upgrades the stone producer to the next level.
- Cost getCost () override

  Gets the cost of the stone producer upgrade.

Public Member Functions inherited from StoneProducerUpgrade

- StoneProducerUpgrade (StoneProducer ∗stoneProd)

  Constructs a StoneProducerUpgrade from a given StoneProducer.
- StoneProducerUpgrade (StoneProducerUpgrade ∗stoneProd)

  Constructs a StoneProducerUpgrade from another StoneProducerUpgrade instance.
- virtual ∼**StoneProducerUpgrade** ()

  Destructor for the StoneProducerUpgrade class.

Public Member Functions inherited from StoneProducer

- StoneProducer (StoneProducer ∗stoneProducer)

  Constructs a StoneProducer from another StoneProducer instance.
- StoneProducer (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a StoneProducer with specified configuration and position.
- virtual ∼**StoneProducer** ()

  Destructor for the StoneProducer class.

Public Member Functions inherited from Industry

- Industry (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Industry entity with specified attributes.
- Industry (Industry ∗industry)

  Copy constructor for the Industry class.
- virtual ∼**Industry** ()

  Virtual destructor for the Industry class.
- void setOutput (int output)

  Sets the production output of the industry.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from StoneProducerUpgrade

- StoneProducer * **stoneProducer**

  Pointer to the base stone producer being upgraded.

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).
- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.109.1 Detailed Description

Represents a level two upgrade for a stone producer.

This class extends the StoneProducerUpgrade class to provide functionality for a level two upgrade of a stone producer.

4.109.2 Constructor & Destructor Documentation

4.109.2.1 StoneProducerLevelTwoUpgrade() [1/2]

```
StoneProducerLevelTwoUpgrade::StoneProducerLevelTwoUpgrade (
            StoneProducer * stoneProd)
```

Constructs a StoneProducerLevelTwoUpgrade from a given StoneProducer.

Parameters

stoneProd	Pointer to the StoneProducer to upgrade.

4.109.2.2 StoneProducerLevelTwoUpgrade() [2/2]

```
StoneProducerLevelTwoUpgrade::StoneProducerLevelTwoUpgrade (
            StoneProducerLevelTwoUpgrade * stoneProd)
```

Constructs a StoneProducerLevelTwoUpgrade from another StoneProducerLevelTwoUpgrade instance.

Parameters

stoneProd	Pointer to the StoneProducerLevelTwoUpgrade to copy.

4.109.3 Member Function Documentation

4.109.3.1 clone()

```
Entity * StoneProducerLevelTwoUpgrade::clone ()  [override], [virtual]
```

Clones the current StoneProducerLevelTwoUpgrade instance.

Returns

A pointer to a new StoneProducerLevelTwoUpgrade that is a copy of this instance.

Implements StoneProducerUpgrade.

4.109.3.2 getCost()

```
Cost StoneProducerLevelTwoUpgrade::getCost ()  [override], [virtual]
```

Gets the cost of the stone producer upgrade.

Returns

The cost associated with the upgrade.

Implements StoneProducerUpgrade.

4.109.3.3 getLevel()

```
int StoneProducerLevelTwoUpgrade::getLevel ()  [override], [virtual]
```

Gets the level of the upgrade.

Returns

The level of the upgrade, which is 2 for this class.

Reimplemented from Industry.

4.109.3.4 getOutput()

```
int StoneProducerLevelTwoUpgrade::getOutput () [override], [virtual]
```

Gets the output of the stone producer upgrade.

Returns

The output value of the upgraded producer.

Implements StoneProducerUpgrade.

4.109.3.5 update()

```
void StoneProducerLevelTwoUpgrade::update () [override], [virtual]
```

Updates the state of the stone producer upgrade.

Implements StoneProducerUpgrade.

4.109.3.6 upgrade()

```
Entity * StoneProducerLevelTwoUpgrade::upgrade () [override], [virtual]
```

Upgrades the stone producer to the next level.

Returns

A pointer to the upgraded StoneProducerLevelThreeUpgrade instance.

Implements StoneProducerUpgrade.

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

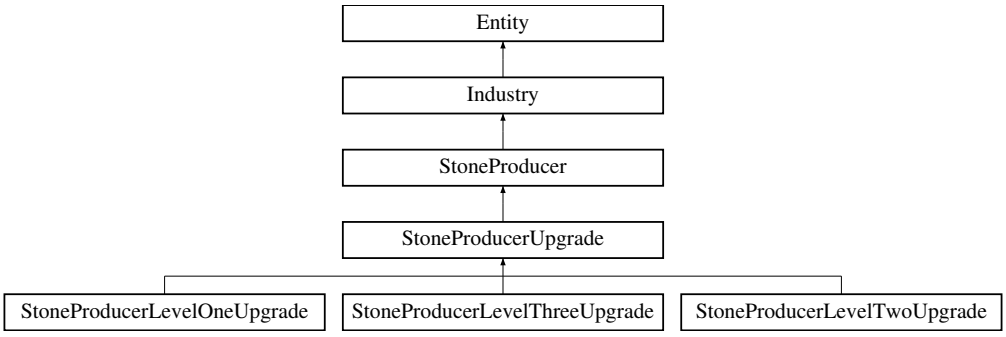
- src/entities/industry/stoneproducer/StoneProducerLevelTwoUpgrade.h
- src/entities/industry/stoneproducer/StoneProducerLevelTwoUpgrade.cpp

4.110 StoneProducerUpgrade Class Reference

Abstract base class for stone producer upgrades.

`#include <StoneProducerUpgrade.h>`

Inheritance diagram for StoneProducerUpgrade:



Public Member Functions

- StoneProducerUpgrade (StoneProducer *stoneProd)

  Constructs a StoneProducerUpgrade from a given StoneProducer.
- StoneProducerUpgrade (StoneProducerUpgrade *stoneProd)

  Constructs a StoneProducerUpgrade from another StoneProducerUpgrade instance.
- virtual ~**StoneProducerUpgrade** ()

  Destructor for the StoneProducerUpgrade class.
- virtual void update ()=0

  Updates the state of the stone producer upgrade.
- virtual Entity * clone ()=0

  Clones the current StoneProducerUpgrade instance.
- virtual int getOutput ()=0

  Gets the output of the stone producer upgrade.
- virtual Entity * upgrade ()=0

  Upgrades the stone producer to the next level.
- virtual Cost getCost ()=0

  Gets the cost of the stone producer upgrade.

Public Member Functions inherited from StoneProducer

- StoneProducer (StoneProducer *stoneProducer)

  Constructs a StoneProducer from another StoneProducer instance.
- StoneProducer (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a StoneProducer with specified configuration and position.
- virtual ~**StoneProducer** ()

  Destructor for the StoneProducer class.

Public Member Functions inherited from Industry

- Industry (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Industry entity with specified attributes.
- Industry (Industry *industry)

  Copy constructor for the Industry class.
- virtual ~**Industry** ()

  Virtual destructor for the Industry class.
- void setOutput (int output)

  Sets the production output of the industry.
- virtual int getLevel ()

  Gets the current level of the industry.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Protected Attributes

- StoneProducer * **stoneProducer**

  Pointer to the base stone producer being upgraded.

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).
- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.110.1 Detailed Description

Abstract base class for stone producer upgrades.

This class provides a common interface for various upgrades of stone producers.

4.110.2 Constructor & Destructor Documentation

4.110.2.1 StoneProducerUpgrade() [1/2]

```
StoneProducerUpgrade::StoneProducerUpgrade (
            StoneProducer * stoneProd)
```

Constructs a StoneProducerUpgrade from a given StoneProducer.

Parameters

stoneProd	Pointer to the StoneProducer to upgrade.

4.110.2.2 StoneProducerUpgrade() [2/2]

```
StoneProducerUpgrade::StoneProducerUpgrade (
            StoneProducerUpgrade * stoneProd)
```

Constructs a StoneProducerUpgrade from another StoneProducerUpgrade instance.

Parameters

stoneProd	Pointer to the StoneProducerUpgrade to copy.

4.110.3 Member Function Documentation

4.110.3.1 clone()

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

Clones the current StoneProducerUpgrade instance.

Returns

A pointer to a new StoneProducerUpgrade that is a copy of this instance.

Reimplemented from StoneProducer.

Implemented in StoneProducerLevelOneUpgrade, StoneProducerLevelThreeUpgrade, and StoneProducerLevelTwoUpgrade.

4.110.3.2 getCost()

```
virtual Cost StoneProducerUpgrade::getCost ()  [pure virtual]
```

Gets the cost of the stone producer upgrade.

Returns

The cost associated with the upgrade.

Reimplemented from Industry.

Implemented in StoneProducerLevelOneUpgrade, StoneProducerLevelThreeUpgrade, and StoneProducerLevelTwoUpgrade.

4.110.3.3 getOutput()

```
virtual int StoneProducerUpgrade::getOutput ()  [pure virtual]
```

Gets the output of the stone producer upgrade.

Returns

The output value of the upgraded producer.

Reimplemented from Industry.

Implemented in StoneProducerLevelOneUpgrade, StoneProducerLevelThreeUpgrade, and StoneProducerLevelTwoUpgrade.

4.110.3.4 update()

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

Updates the state of the stone producer upgrade.

This method is implemented by derived classes.

Reimplemented from StoneProducer.

Implemented in StoneProducerLevelOneUpgrade, StoneProducerLevelThreeUpgrade, and StoneProducerLevelTwoUpgrade.

4.110.3.5 upgrade()

```
virtual Entity * StoneProducerUpgrade::upgrade ()  [pure virtual]
```

Upgrades the stone producer to the next level.

Returns

A pointer to the upgraded StoneProducer instance.

Reimplemented from StoneProducer.

Implemented in StoneProducerLevelOneUpgrade, StoneProducerLevelThreeUpgrade, and StoneProducerLevelTwoUpgrade.

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

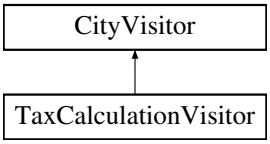
- src/entities/industry/stoneproducer/StoneProducerUpgrade.h
- src/entities/industry/stoneproducer/StoneProducerUpgrade.cpp

4.111 TaxCalculationVisitor Class Reference

Visitor that calculates total tax from residential and economic buildings in a city.

`#include <TaxCalculationVisitor.h>`

Inheritance diagram for TaxCalculationVisitor:



Public Member Functions

- **TaxCalculationVisitor** ()
  Constructs a TaxCalculationVisitor with zeroed tax values.
- ~**TaxCalculationVisitor** ()
  Default destructor.
- void visit (City *city) override
  Visits a city to calculate tax from buildings.
- int getTotalResidentialTax () const
  Gets the total tax from residential buildings.
- int getTotalEconomicTax () const
  Gets the total tax from economic buildings.
- int getTotalTax () const
  Gets the combined tax from residential and economic buildings.

Public Member Functions inherited from CityVisitor

- **CityVisitor** ()=default
  Default constructor.
- virtual ~**CityVisitor** ()=default
  Default destructor.

4.111.1 Detailed Description

Visitor that calculates total tax from residential and economic buildings in a city.

4.111.2 Member Function Documentation

4.111.2.1 getTotalEconomicTax()

```
int TaxCalculationVisitor::getTotalEconomicTax () const  [inline]
```

Gets the total tax from economic buildings.

Returns

Total economic tax.

4.111.2.2 getTotalResidentialTax()

```
int TaxCalculationVisitor::getTotalResidentialTax () const  [inline]
```

Gets the total tax from residential buildings.

Returns

Total residential tax.

4.111.2.3 getTotalTax()

```
int TaxCalculationVisitor::getTotalTax () const  [inline]
```

Gets the combined tax from residential and economic buildings.

Returns

Sum of residential and economic taxes.

4.111.2.4 visit()

```
void TaxCalculationVisitor::visit (
            City * city)  [override], [virtual]
```

Visits a city to calculate tax from buildings.

Parameters

city	Pointer to the City object being visited.

Implements CityVisitor.

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

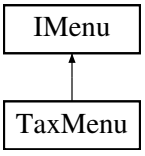
- src/visitors/tax/TaxCalculationVisitor.h
- src/visitors/tax/TaxCalculationVisitor.cpp

4.112 TaxMenu Class Reference

Provides functionality for managing and adjusting tax rates in the game.

```
#include <TaxMenu.h>
```

Inheritance diagram for TaxMenu:



Public Member Functions

- TaxMenu ()

  Constructs a TaxMenu object.
- ~TaxMenu ()

  Destructor for TaxMenu.
- void display () const override

  Displays the tax management menu.
- void handleInput () override

  Handles user input for the Tax menu.

Public Member Functions inherited from IMenu

- **IMenu** ()=default

  Default constructor for IMenu.
- IMenu (std::string heading)

  Constructor to initialize a menu with a specified heading.
- virtual ~**IMenu** ()=default

  Virtual destructor for IMenu.
- void setHeading (const std::string &heading)

  Sets the heading of the menu.

Additional Inherited Members

Protected Member Functions inherited from IMenu

- std::string repeat (const std::string &str, int times) const

  Utility function to repeat a string multiple times.
- int calculateMaxWidth (const std::string &menuHeading, const std::vector< Section > &sections) const

  Calculates the maximum width required for the menu.
- void printTopBorder (int width) const

  Prints the top border of the menu using box-drawing characters.
- void printBottomBorder (int width) const

  Prints the bottom border of the menu using box-drawing characters.
- void printSectionDivider (int width) const

  Prints a section divider in the menu using box-drawing characters.
- void printDoubleLineDivider (int width) const

  Prints a double-line divider for the main heading of the menu.
- std::string centerText (const std::string &text, int width) const

  Centers text within a specified width using space padding.
- std::string centerTextWithChar (const std::string &text, int width, const std::string &padChar) const

  Centers text within a specified width using a custom character for padding.
- void displayMenu () const

  Displays the formatted menu, including sections and options.
- void **displayChoicePrompt** () const

  Displays the choice prompt for user input.
- void displayChoiceMessagePrompt (const std::string &message) const

  Displays a custom message prompt for user input.
- void **displayInvalidChoice** () const

Displays an error message when the user makes an invalid choice.

- void displayErrorMessage (const std::string &message) const

  Displays a general error message.
- void displaySuccessMessage (const std::string &message) const

  Displays a success message in green color.
- void **displayPressEnterToContinue** () const

  Displays a message asking the user to press Enter to continue.
- void **clearScreen** () const

  Clears the terminal screen.
- std::string stripColorCodes (const std::string &input) const

  Strips ANSI color codes from a string.
- virtual void displayAvailablePositions (const std::vector< std::vector< int > > &positions) const

  Displays available positions on the city grid for an entity.

Static Protected Member Functions inherited from IMenu

- static char indexToExtendedChar (int index)

  Converts a numeric index (0-99) to a single character in an extended set.
- static std::string coordinatesToLabel (int x, int y)

  Converts x and y coordinates to a labeled string (e.g., "A, 1").

Protected Attributes inherited from IMenu

- std::vector< Section > **sections**

  List of sections contained in the menu.
- std::string **menuHeading**

  The heading/title of the menu.
- bool **hasExited**

  Flag indicating if the menu has been exited.
- CityManager **cityManager**

  Manager for city-related operations.
- bool **displayResources**

  Flag indicating whether to display resources in the menu.
- bool **isInfoMenu**

  Flag indicating whether to display option numbers.

Static Protected Attributes inherited from IMenu

- static const char * **RESET** = "\033[0m"

  ANSI color codes and styles for use in all menus.
- static const char * **BOLD_WHITE** = "\033[1;37m"
- static const char * **NORMAL_WHITE** = "\033[0;37m"
- static const char * **DARK_GRAY** = "\033[1;30m"
- static const char * **BOLD_YELLOW** = "\033[1;33m"
- static const char * **BOLD_GREEN** = "\033[1;32m"
- static const char * **BOLD_RED** = "\033[1;31m"
- static const char * **BOLD_CYAN** = "\033[1;36m"
- static const char * **BLUE** = "\033[34m"

4.112.1 Detailed Description

Provides functionality for managing and adjusting tax rates in the game.

The TaxMenu class allows players to increase or decrease economic and residential tax rates through user-friendly menu interactions.

4.112.2 Constructor & Destructor Documentation

4.112.2.1 TaxMenu()

```
TaxMenu::TaxMenu ()
```

Constructs a TaxMenu object.

Initializes the TaxMenu with options for adjusting tax rates and navigation.

4.112.2.2 ~TaxMenu()

```
TaxMenu::~TaxMenu ()
```

Destructor for TaxMenu.

Cleans up resources used by the TaxMenu.

4.112.3 Member Function Documentation

4.112.3.1 display()

```
void TaxMenu::display () const  [override], [virtual]
```

Displays the tax management menu.

Overrides the display method from IMenu to present tax adjustment options.

Implements IMenu.

4.112.3.2 handleInput()

```
void TaxMenu::handleInput ()  [override], [virtual]
```

Handles user input for the Tax menu.

Processes user selections to increase or decrease economic and residential taxes, and navigates back to the main menu as needed.

Implements IMenu.

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

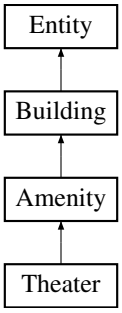
- src/menus/tax/TaxMenu.h
- src/menus/tax/TaxMenu.cpp

4.113 Theater Class Reference

Represents a theater entity within the game.

`#include <Theater.h>`

Inheritance diagram for Theater:



Public Member Functions

- Theater (EntityConfig ec, Size size, int xPos, int yPos)
  Constructs a Theater with specified attributes.
- Theater (Theater ∗theater)
  Copy constructor for the Theater class.
- virtual ∼**Theater** ()
  Destructor for the Theater class.
- void update ()
  Updates the theater's state.
- Entity ∗ clone ()
  Creates a clone of the theater.

Public Member Functions inherited from Amenity

- Amenity (EntityConfig ec, Size size, int xPos, int yPos)
  Constructs an Amenity with specified attributes.
- Amenity (Amenity ∗amenity)
  Copy constructor for the Amenity class.
- virtual ∼**Amenity** ()
  Virtual destructor for the Amenity class.

Public Member Functions inherited from Building

- Building (EntityConfig ec, Size size, int xPos, int yPos)
  Parameterized constructor for the Building class.
- Building (Building ∗building)
  Copy constructor for the Building class.
- virtual ∼Building ()
  Destructor for the Building class.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).
- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.113.1 Detailed Description

Represents a theater entity within the game.

The Theater class is a specific type of Amenity, providing unique attributes and behaviors related to theaters, such as increasing local entertainment value. This class includes implementations for updating the theater's state and cloning itself.

4.113.2 Constructor & Destructor Documentation

4.113.2.1 Theater() [1/2]

```
Theater::Theater (
            EntityConfig ec,
            Size size,
            int xPos,
            int yPos)
```

Constructs a Theater with specified attributes.

Parameters

ec	Configuration containing resource consumption and properties.
size	Size of the theater.
xPos	X-coordinate position of the theater.
yPos	Y-coordinate position of the theater.

4.113.2.2 Theater() [2/2]

```
Theater::Theater (
            Theater * theater)
```

Copy constructor for the Theater class.

Creates a new Theater by copying the attributes of an existing Theater.

Parameters

theater	Pointer to the Theater object to be copied.

4.113.3 Member Function Documentation

4.113.3.1 clone()

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

Creates a clone of the theater.

Returns

A pointer to the cloned Theater.

Implements Amenity.

4.113.3.2 update()

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

Updates the theater's state.

Implements Amenity.

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

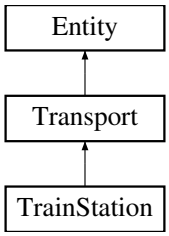
- src/entities/building/amenity/Theater.h
- src/entities/building/amenity/Theater.cpp

4.114 TrainStation Class Reference

Represents a train station entity within the game.

`#include <TrainStation.h>`

Inheritance diagram for TrainStation:



Public Member Functions

- TrainStation (EntityConfig ec, Size size, int xPos, int yPos)
  Constructs a TrainStation entity with specified attributes.
- TrainStation (TrainStation ∗trainStation)
  Copy constructor for the TrainStation class.
- virtual ∼**TrainStation** ()
  Destructor for the TrainStation class.
- void update ()
  Updates the state of the train station entity.
- Entity ∗ clone ()
  Creates a clone of the train station entity.

Public Member Functions inherited from Transport

- Transport (EntityConfig ec, Size size, int xPos, int yPos)
  Constructs a Transport entity with specified attributes.
- Transport (Transport ∗transport)
  Copy constructor for the Transport class.
- virtual ∼**Transport** ()
  Virtual destructor for the Transport class.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)
  Constructs an Entity with specified attributes.
- Entity (Entity ∗entity)
  Copy constructor for the Entity class.
- virtual ∼**Entity** ()
  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity ∗entity)
  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

Gets the X-coordinate position of the entity.

- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.

- int **localEffectStrength**
  Local effect strength of the entity.
- int **globalEffectStrength**
  Global effect strength of the entity.
- int **width**
  Width of the entity.
- int **height**
  Height of the entity.
- int **xPosition**
  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**
  Y-coordinate of the entity's position (bottom left corner).
- Size **size**
  Size object representing the entity's dimensions.
- EntityType **type**
  The type of entity.
- State * **state**
  Pointer to the current state of the entity.
- int **revenue**
  Revenue generated by the entity.
- float **electricityConsumption**
  Electricity consumption of the entity.
- float **waterConsumption**
  Water consumption of the entity.
- std::vector< Entity * > **observers**
  List of other entities observing this entity.

4.114.1 Detailed Description

Represents a train station entity within the game.

The TrainStation class manages the properties and behavior of train station entities, including their position, size, and functionality related to transportation.

4.114.2 Constructor & Destructor Documentation

4.114.2.1 TrainStation() [1/2]

```
TrainStation::TrainStation (
            EntityConfig ec,
            Size size,
            int xPos,
            int yPos)
```

Constructs a TrainStation entity with specified attributes.

Parameters

ec	Configuration settings for the entity.
size	Size of the train station entity.
xPos	X-coordinate position of the train station.
yPos	Y-coordinate position of the train station.

4.114.2.2 TrainStation() [2/2]

```
TrainStation::TrainStation (
            TrainStation * trainStation)
```

Copy constructor for the TrainStation class.

Creates a new TrainStation entity by copying the attributes of an existing TrainStation.

Parameters

trainStation	Pointer to the TrainStation object to be copied.

4.114.3 Member Function Documentation

4.114.3.1 clone()

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

Creates a clone of the train station entity.

Returns

A pointer to the cloned TrainStation entity.

Implements Transport.

4.114.3.2 update()

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

Updates the state of the train station entity.

Implements Transport.

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

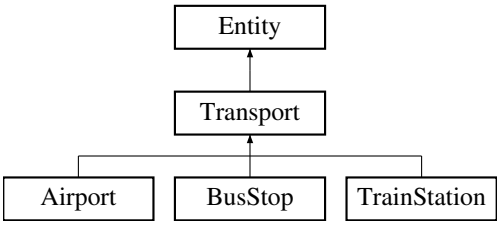
- src/entities/transport/TrainStation.h
- src/entities/transport/TrainStation.cpp

4.115 Transport Class Reference

Abstract base class representing a transport entity within the game.

```
#include <Transport.h>
```

Inheritance diagram for Transport:



Public Member Functions

- Transport (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a Transport entity with specified attributes.
- Transport (Transport *transport)

  Copy constructor for the Transport class.
- virtual ~**Transport** ()

  Virtual destructor for the Transport class.
- virtual void update ()=0

  Updates the state of the transport entity.
- virtual Entity * clone ()=0

  Creates a clone of the transport entity.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

Unsubscribes this entity from all buildings it is observing.

- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.

- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.

- EntityType getType () const

  Gets the entity type of this entity.

- Size getSize () const

  Gets the size of this entity.

- std::string getSymbol ()

  Gets the symbol of the entity.

- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.

- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.

- int **effectRadius**

  Radius of effect for this entity.

- int **localEffectStrength**

  Local effect strength of the entity.

- int **globalEffectStrength**

  Global effect strength of the entity.

- int **width**

  Width of the entity.

- int **height**

  Height of the entity.

- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).

- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).

- Size **size**

  Size object representing the entity's dimensions.

- EntityType **type**

  The type of entity.

- State * **state**

  Pointer to the current state of the entity.

- int **revenue**

  Revenue generated by the entity.

- float **electricityConsumption**

  Electricity consumption of the entity.

- float **waterConsumption**

  Water consumption of the entity.

- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.115.1 Detailed Description

Abstract base class representing a transport entity within the game.

The Transport class serves as a base for different transport entities, managing shared properties and behaviors such as position, size, and the need for updates in derived classes.

4.115.2 Constructor & Destructor Documentation

4.115.2.1 Transport() [1/2]

```
Transport::Transport (
            EntityConfig ec,
            Size size,
            int xPos,
            int yPos)
```

Constructs a Transport entity with specified attributes.

Parameters

ec	Configuration settings for the entity.
size	Size of the transport entity.
xPos	X-coordinate position of the transport.
yPos	Y-coordinate position of the transport.

4.115.2.2 Transport() [2/2]

```
Transport::Transport (
            Transport * transport)
```

Copy constructor for the Transport class.

Creates a new Transport entity by copying the attributes of an existing Transport.

Parameters

transport	Pointer to the Transport object to be copied.

4.115.3 Member Function Documentation

4.115.3.1 clone()

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

Creates a clone of the transport entity.

This method must be implemented in derived classes to return a copy of the entity.

Returns

A pointer to the cloned Transport entity.

Implements Entity.

Implemented in Airport, BusStop, and TrainStation.

4.115.3.2 update()

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

Updates the state of the transport entity.

This method must be implemented in derived classes to define specific behaviors.

Implements Entity.

Implemented in Airport, BusStop, and TrainStation.

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

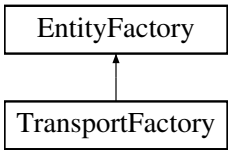
- src/entities/transport/Transport.h
- src/entities/transport/Transport.cpp

4.116 TransportFactory Class Reference

Factory class for creating transport-related entities, including bus stops, train stations, and airports.

```
#include <TransportFactory.h>
```

Inheritance diagram for TransportFactory:



Public Member Functions

- **TransportFactory** ()

  Default constructor for TransportFactory.
- ~**TransportFactory** ()

  Destructor for TransportFactory.
- virtual Entity * createEntity (EntityType type, Size size, int xPos, int yPos)

  Creates a transport entity of the specified type and size at the given position.

Public Member Functions inherited from EntityFactory

- **EntityFactory** ()

  Default constructor for EntityFactory.
- virtual ~**EntityFactory** ()

  Virtual destructor for EntityFactory.

4.116.1 Detailed Description

Factory class for creating transport-related entities, including bus stops, train stations, and airports.

Inherits from EntityFactory and provides methods for creating transport entities of various sizes (small, medium, and large) positioned at specified coordinates.

4.116.2 Member Function Documentation

4.116.2.1 createEntity()

```
Entity * TransportFactory::createEntity (
            EntityType type,
            Size size,
            int xPos,
            int yPos)  [virtual]
```

Creates a transport entity of the specified type and size at the given position.

Parameters

type	The type of transport entity to create (e.g., BusStop, TrainStation, Airport).
size	The size of the entity (small, medium, or large).
xPos	The x-coordinate for the entity's position.
yPos	The y-coordinate for the entity's position.

Returns

A pointer to the created Entity.

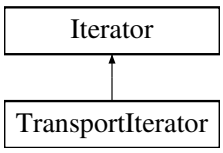
Implements EntityFactory.

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

- src/factory/transport/TransportFactory.h
- src/factory/transport/TransportFactory.cpp

4.117 TransportIterator Class Reference

Inheritance diagram for TransportIterator:



Public Member Functions

- **TransportIterator** ()

  Construct a new Transport Iterator:: Transport Iterator object.
- ~**TransportIterator** ()

  Destroy the Transport Iterator:: Transport Iterator object.
- TransportIterator (std::vector< std::vector< Entity * > > &grid)

  Construct a new Transport Iterator:: Transport Iterator object.
- void first ()

  Sets the iterator to the first unvisited Transport.
- void next ()

  Advances to the next unvisited Transport.
- bool hasNext ()

  Checks if there is another unvisited Transport.
- Entity * current ()

  Returns the current Transport.

Public Member Functions inherited from Iterator

- **Iterator** ()

  Construct a new Iterator object, initializing row and column to zero.
- virtual ~**Iterator** ()

  Destroy the Iterator object.
- **Iterator** (std::vector< std::vector< Entity * > > &grid)
- virtual int getRow ()

  Get the current row index of the iterator.
- virtual int getCol ()

  Get the current column index of the iterator.

Additional Inherited Members

Protected Member Functions inherited from Iterator

- bool isVisited (Entity *entity)

  Check if the specified entity has been visited.
- void markVisited (Entity *entity)

  Mark the specified entity as visited.

Protected Attributes inherited from Iterator

- std::vector< std::vector< Entity * > > **grid**
- std::vector< std::vector< Entity * > >::iterator **currRow**
- std::vector< Entity * >::iterator **curr**
- int **row**
- int **col**
- std::unordered_set< Entity * > **visitedEntities**

4.117.1 Constructor & Destructor Documentation

4.117.1.1 TransportIterator()

```
TransportIterator::TransportIterator (
            std::vector< std::vector< Entity * > > & grid)
```

Construct a new Transport Iterator:: Transport Iterator object.

Parameters

grid	

4.117.2 Member Function Documentation

4.117.2.1 current()

```
Entity * TransportIterator::current ()  [virtual]
```

Returns the current Transport.

Returns

Entity*

Implements Iterator.

4.117.2.2 first()

```
void TransportIterator::first ()  [virtual]
```

Sets the iterator to the first unvisited Transport.

Implements Iterator.

4.117.2.3 hasNext()

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

Checks if there is another unvisited Transport.

Returns

true if there is another unvisited Transport, false otherwise

Implements Iterator.

4.117.2.4 next()

```
void TransportIterator::next ()  [virtual]
```

Advances to the next unvisited Transport.

Implements Iterator.

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

- src/iterators/transport/TransportIterator.h
- src/iterators/transport/TransportIterator.cpp

4.118 TransportManager Class Reference

Manages the construction and maintenance of transportation infrastructure.

`#include <TransportManager.h>`

Public Member Functions

- **TransportManager** ()

  Constructs a new TransportManager object.
- ~**TransportManager** ()

  Destroys the TransportManager object.
- bool canAffordRoad ()

  Checks if there are enough resources to build a road.
- bool buildRoad (int x, int y)

  Builds a road at the specified coordinates.
- bool buildPublicTransit (EntityType type, Size size, int x, int y)

  Builds a public transit system at the specified coordinates.
- bool canAffordPublicTransit (EntityType type, Size size)

  Determines if public transit can be afforded based on the entity type and size.

4.118.1 Detailed Description

Manages the construction and maintenance of transportation infrastructure.

The TransportManager class provides functionalities to check if a road can be afforded, build roads, and build public transit systems.

4.118.2 Member Function Documentation

4.118.2.1 buildPublicTransit()

```
bool TransportManager::buildPublicTransit (
            EntityType type,
            Size size,
            int x,
            int y)
```

Builds a public transit system at the specified coordinates.

Parameters

type	The type of public transit system to build.
size	The size of the public transit system.
x	The x-coordinate where the public transit system will be built.
y	The y-coordinate where the public transit system will be built.

Returns

true if the public transit system was successfully built, false otherwise.

4.118.2.2 buildRoad()

```
bool TransportManager::buildRoad (
            int x,
            int y)
```

Builds a road at the specified coordinates.

Parameters

x	The x-coordinate where the road will be built.
y	The y-coordinate where the road will be built.

Returns

true if the road was successfully built, false otherwise.

4.118.2.3 canAffordPublicTransit()

```
bool TransportManager::canAffordPublicTransit (
            EntityType type,
            Size size)
```

Determines if public transit can be afforded based on the entity type and size.

Parameters

type	The type of the entity (e.g., individual, organization).
size	The size of the entity (e.g., small, medium, large).

Returns

true if public transit can be afforded, false otherwise.

4.118.2.4 canAffordRoad()

```
bool TransportManager::canAffordRoad ()
```

Checks if there are enough resources to build a road.

Returns

true if a road can be afforded, false otherwise.

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

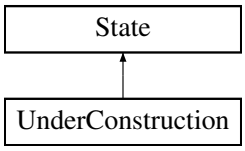
- src/managers/TransportManager.h
- src/managers/TransportManager.cpp

4.119 UnderConstruction Class Reference

Represents the state of an entity that is currently under construction.

`#include <UnderConstruction.h>`

Inheritance diagram for UnderConstruction:



Public Member Functions

- UnderConstruction (int buildTime)

  Constructs an UnderConstruction state with the specified build time.
- UnderConstruction (UnderConstruction ∗underConstruction)

  Copy constructor for the UnderConstruction class.
- ∼**UnderConstruction** ()

  Destructor for the UnderConstruction state.
- State ∗ update ()

  Updates the current construction state.
- State ∗ clone ()

  Creates a deep copy of the UnderConstruction state.

Public Member Functions inherited from State

- State (int buildTime)

  Constructs a State with the specified build time.
- State (State ∗state)

  Copy constructor for the State class.
- virtual ∼**State** ()

  Destructor for the State.
- int getGameLoopCounter ()

  Gets the current game loop counter.
- int getBuildTime ()

  Gets the build time of the state.
- void **incrementGameLoopCounter** ()

  Increments the game loop counter.

4.119.1 Detailed Description

Represents the state of an entity that is currently under construction.

The UnderConstruction class inherits from the State class and implements the behavior for an entity that is in the process of being built.

4.119.2 Constructor & Destructor Documentation

4.119.2.1 UnderConstruction() [1/2]

```
UnderConstruction::UnderConstruction (
            int buildTime)
```

Constructs an UnderConstruction state with the specified build time.

Parameters

buildTime	The time required for the construction to complete.

4.119.2.2 UnderConstruction() [2/2]

```
UnderConstruction::UnderConstruction (
            UnderConstruction * underConstruction)
```

Copy constructor for the UnderConstruction class.

Creates a new UnderConstruction state by copying the attributes of an existing UnderConstruction object.

Parameters

underConstruction	Pointer to the existing UnderConstruction object to be copied.

4.119.3 Member Function Documentation

4.119.3.1 clone()

```
State * UnderConstruction::clone ()  [virtual]
```

Creates a deep copy of the UnderConstruction state.

This method returns a new UnderConstruction object that is a copy of the current instance. This allows for proper polymorphic copying of State objects.

Returns

A pointer to a new UnderConstruction object that is a copy of this instance.

Implements State.

4.119.3.2 update()

```
State * UnderConstruction::update ()  [virtual]
```

Updates the current construction state.

Returns

A pointer to the next state after the update, which may be a Built state.

Implements State.

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

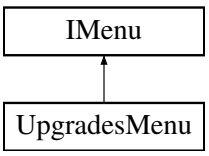
- src/entities/state/UnderConstruction.h
- src/entities/state/UnderConstruction.cpp

4.120 UpgradesMenu Class Reference

Provides a menu interface for upgrading utilities and industries in the game.

`#include <UpgradesMenu.h>`

Inheritance diagram for UpgradesMenu:



Public Member Functions

- UpgradesMenu ()

  Constructs an UpgradesMenu object.
- ~UpgradesMenu ()

  Destructor for UpgradesMenu.
- void display () const override

  Displays the upgrade options menu.
- void handleInput () override

  Handles user input for the Upgrades menu.

Public Member Functions inherited from IMenu

- **IMenu** ()=default

  Default constructor for IMenu.
- IMenu (std::string heading)

  Constructor to initialize a menu with a specified heading.
- virtual ~**IMenu** ()=default

  Virtual destructor for IMenu.
- void setHeading (const std::string &heading)

  Sets the heading of the menu.

Additional Inherited Members

Protected Member Functions inherited from IMenu

- std::string repeat (const std::string &str, int times) const

  Utility function to repeat a string multiple times.
- int calculateMaxWidth (const std::string &menuHeading, const std::vector< Section > &sections) const

  Calculates the maximum width required for the menu.
- void printTopBorder (int width) const

  Prints the top border of the menu using box-drawing characters.
- void printBottomBorder (int width) const

  Prints the bottom border of the menu using box-drawing characters.
- void printSectionDivider (int width) const

Prints a section divider in the menu using box-drawing characters.

- void printDoubleLineDivider (int width) const

  Prints a double-line divider for the main heading of the menu.
- std::string centerText (const std::string &text, int width) const

  Centers text within a specified width using space padding.
- std::string centerTextWithChar (const std::string &text, int width, const std::string &padChar) const

  Centers text within a specified width using a custom character for padding.
- void displayMenu () const

  Displays the formatted menu, including sections and options.
- void **displayChoicePrompt** () const

  Displays the choice prompt for user input.
- void displayChoiceMessagePrompt (const std::string &message) const

  Displays a custom message prompt for user input.
- void **displayInvalidChoice** () const

  Displays an error message when the user makes an invalid choice.
- void displayErrorMessage (const std::string &message) const

  Displays a general error message.
- void displaySuccessMessage (const std::string &message) const

  Displays a success message in green color.
- void **displayPressEnterToContinue** () const

  Displays a message asking the user to press Enter to continue.
- void **clearScreen** () const

  Clears the terminal screen.
- std::string stripColorCodes (const std::string &input) const

  Strips ANSI color codes from a string.
- virtual void displayAvailablePositions (const std::vector< std::vector< int > > &positions) const

  Displays available positions on the city grid for an entity.

Static Protected Member Functions inherited from IMenu

- static char indexToExtendedChar (int index)

  Converts a numeric index (0-99) to a single character in an extended set.
- static std::string coordinatesToLabel (int x, int y)

  Converts x and y coordinates to a labeled string (e.g., "A, 1").

Protected Attributes inherited from IMenu

- std::vector< Section > **sections**

  List of sections contained in the menu.
- std::string **menuHeading**

  The heading/title of the menu.
- bool **hasExited**

  Flag indicating if the menu has been exited.
- CityManager **cityManager**

  Manager for city-related operations.
- bool **displayResources**

  Flag indicating whether to display resources in the menu.
- bool **isInfoMenu**

  Flag indicating whether to display option numbers.

Static Protected Attributes inherited from IMenu

- static const char * **RESET** = "\033[0m"

  ANSI color codes and styles for use in all menus.
- static const char * **BOLD_WHITE** = "\033[1;37m"
- static const char * **NORMAL_WHITE** = "\033[0;37m"
- static const char * **DARK_GRAY** = "\033[1;30m"
- static const char * **BOLD_YELLOW** = "\033[1;33m"
- static const char * **BOLD_GREEN** = "\033[1;32m"
- static const char * **BOLD_RED** = "\033[1;31m"
- static const char * **BOLD_CYAN** = "\033[1;36m"
- static const char * **BLUE** = "\033[34m"

4.120.1 Detailed Description

Provides a menu interface for upgrading utilities and industries in the game.

The UpgradesMenu class enables players to navigate and select upgrade options for various game systems, such as utilities and industries. It includes methods to display the menu, handle user input, and confirm specific upgrades.

4.120.2 Constructor & Destructor Documentation

4.120.2.1 UpgradesMenu()

```
UpgradesMenu::UpgradesMenu ()
```

Constructs an UpgradesMenu object.

Initializes the menu options for upgrading utilities and industries.

4.120.2.2 ~UpgradesMenu()

```
UpgradesMenu::~UpgradesMenu ()
```

Destructor for UpgradesMenu.

Cleans up resources used by the UpgradesMenu.

4.120.3 Member Function Documentation

4.120.3.1 display()

```
void UpgradesMenu::display () const  [override], [virtual]
```

Displays the upgrade options menu.

Overrides the display method from IMenu to present the upgrade categories to the user.

Implements IMenu.

4.120.3.2 handleInput()

```
void UpgradesMenu::handleInput () [override], [virtual]
```

Handles user input for the Upgrades menu.

Processes user selections to navigate through the upgrade options and confirms choices.

Implements IMenu.

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

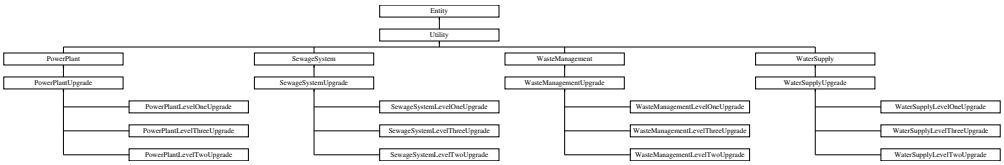
- src/menus/upgrades/UpgradesMenu.h
- src/menus/upgrades/UpgradesMenu.cpp

4.121 Utility Class Reference

Represents a utility entity in the city builder, such as power plants or sewage systems.

```
#include <Utility.h>
```

Inheritance diagram for Utility:



Public Member Functions

- Utility (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a Utility object with the specified parameters.
- Utility (Utility *utility)

  Copy constructor for the Utility class.
- virtual ~**Utility** ()

  Destructor for the Utility object.
- virtual void update ()=0

  Pure virtual function to update the utility's state.
- virtual Entity * clone ()=0

  Pure virtual function to clone the utility.
- virtual Entity * upgrade ()=0

  Upgrades the current utility to the next level.
- virtual int getOutput ()

  Retrieves the output of the utility.
- void setOutput (int output)

  Sets the output value of the utility.
- virtual Cost getCost ()

  Retrieves the cost of the utility or its upgraded version.
- virtual int getLevel ()

  Gets the level of the utility.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).
- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.121.1 Detailed Description

Represents a utility entity in the city builder, such as power plants or sewage systems.

The Utility class is a specialized entity that has an output value, such as electricity or water.

4.121.2 Constructor & Destructor Documentation

4.121.2.1 Utility() [1/2]

```
Utility::Utility (
            EntityConfig ec,
            Size size,
            int xPos,
            int yPos)
```

Constructs a Utility object with the specified parameters.

This constructor initializes a Utility instance with various attributes including output, resource consumption, effects, dimensions, and positioning.

Parameters

ec	EntityConfig.
size	Size.
xPos	xPosition
yPos	yPosition

4.121.2.2 Utility() [2/2]

```
Utility::Utility (
            Utility * utility)
```

Copy constructor for the Utility class.

Creates a deep copy of a Utility object by copying the attributes of an existing Utility object.

Parameters

utility	Pointer to the existing Utility object to be copied.

4.121.3 Member Function Documentation

4.121.3.1 clone()

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

Pure virtual function to clone the utility.

Implement this method to create a copy of the current utility instance.

Returns

A pointer to a new cloned Utility object.

Implements Entity.

Implemented in PowerPlant, PowerPlantLevelOneUpgrade, PowerPlantLevelThreeUpgrade, PowerPlantLevelTwoUpgrade, PowerPlantUpgrade, SewageSystem, SewageSystemLevelOneUpgrade, SewageSystemLevelThreeUpgrade, SewageSystemLevelTwoUpgrade, SewageSystemUpgrade, WasteManagement, WasteManagementLevelOneUpgrade, WasteManagementLevelThreeUpgrade, WasteManagementLevelTwoUpgrade, WasteManagementUpgrade, WaterSupply, WaterSupplyLevelOneUpgrade, WaterSupplyLevelThreeUpgrade, WaterSupplyLevelTwoUpgrade, and WaterSupplyUpgrade.

4.121.3.2 getCost()

```
Cost Utility::getCost ()  [virtual]
```

Retrieves the cost of the utility or its upgraded version.

Returns

A Cost object representing the monetary and material costs.

Reimplemented in PowerPlantLevelOneUpgrade, PowerPlantLevelThreeUpgrade, PowerPlantLevelTwoUpgrade, PowerPlantUpgrade, SewageSystemLevelOneUpgrade, SewageSystemLevelThreeUpgrade, SewageSystemLevelTwoUpgrade, SewageSystemUpgrade, WasteManagementLevelOneUpgrade, WasteManagementLevelThreeUpgrade, WasteManagementLevelTw WasteManagementUpgrade, WaterSupplyLevelOneUpgrade, WaterSupplyLevelThreeUpgrade, WaterSupplyLevelTwoUpgrade, and WaterSupplyUpgrade.

4.121.3.3 getLevel()

```
int Utility::getLevel ()  [virtual]
```

Gets the level of the utility.

Returns

The level of the utility.

Reimplemented in PowerPlantLevelOneUpgrade, PowerPlantLevelThreeUpgrade, PowerPlantLevelTwoUpgrade, SewageSystemLevelOneUpgrade, SewageSystemLevelThreeUpgrade, SewageSystemLevelTwoUpgrade, WasteManagementLevelOneUpgrade, WasteManagementLevelThreeUpgrade, WasteManagementLevelTwoUpgrade, WaterSupplyLevelOneUpgrade, WaterSupplyLevelThreeUpgrade, and WaterSupplyLevelTwoUpgrade.

4.121.3.4 getOutput()

```
int Utility::getOutput ()  [virtual]
```

Retrieves the output of the utility.

Returns

The output value of the utility.

Reimplemented in PowerPlantLevelOneUpgrade, PowerPlantLevelThreeUpgrade, PowerPlantLevelTwoUpgrade, PowerPlantUpgrade, SewageSystemLevelOneUpgrade, SewageSystemLevelThreeUpgrade, SewageSystemLevelTwoUpgrade, SewageSystemUpgrade, WasteManagementLevelOneUpgrade, WasteManagementLevelThreeUpgrade, WasteManagementLevelTw WasteManagementUpgrade, WaterSupplyLevelOneUpgrade, WaterSupplyLevelThreeUpgrade, WaterSupplyLevelTwoUpgrade, and WaterSupplyUpgrade.

4.121.3.5 setOutput()

```
void Utility::setOutput (
            int output)
```

Sets the output value of the utility.

Parameters

output	The new output value to be set.

4.121.3.6 update()

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

Pure virtual function to update the utility's state.

Implement this method to define how the utility behaves when its state is updated.

Implements Entity.

Implemented in PowerPlant, PowerPlantLevelOneUpgrade, PowerPlantLevelThreeUpgrade, PowerPlantLevelTwoUpgrade, PowerPlantUpgrade, SewageSystem, SewageSystemLevelOneUpgrade, SewageSystemLevelThreeUpgrade, SewageSystemLevelTwoUpgrade, SewageSystemUpgrade, WasteManagement, WasteManagementLevelOneUpgrade, WasteManagementLevelThreeUpgrade, WasteManagementLevelTwoUpgrade, WasteManagementUpgrade, WaterSupply, WaterSupplyLevelOneUpgrade, WaterSupplyLevelThreeUpgrade, WaterSupplyLevelTwoUpgrade, and WaterSupplyUpgrade.

4.121.3.7 upgrade()

```
virtual Entity * Utility::upgrade ()  [pure virtual]
```

Upgrades the current utility to the next level.

Returns

A pointer to the upgraded utility instance, or nullptr if already at maximum level.

Implemented in PowerPlant, PowerPlantLevelOneUpgrade, PowerPlantLevelThreeUpgrade, PowerPlantLevelTwoUpgrade, PowerPlantUpgrade, SewageSystem, SewageSystemLevelOneUpgrade, SewageSystemLevelThreeUpgrade, SewageSystemLevelTwoUpgrade, SewageSystemUpgrade, WasteManagement, WasteManagementLevelOneUpgrade, WasteManagementLevelThreeUpgrade, WasteManagementLevelTwoUpgrade, WasteManagementUpgrade, WaterSupply, WaterSupplyLevelOneUpgrade, WaterSupplyLevelThreeUpgrade, WaterSupplyLevelTwoUpgrade, and WaterSupplyUpgrade.

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

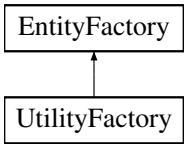
- src/entities/utility/base/Utility.h
- src/entities/utility/base/Utility.cpp

4.122 UtilityFactory Class Reference

Factory class to create utility entities such as power plants, water supplies, waste management facilities, and sewage systems.

```
#include <UtilityFactory.h>
```

Inheritance diagram for UtilityFactory:



Public Member Functions

- **UtilityFactory** ()

  Default constructor for UtilityFactory.
- ~**UtilityFactory** ()

  Destructor for UtilityFactory.
- virtual Entity * createEntity (EntityType type, Size size, int xPos, int yPos)

  Creates a utility entity of a specified type and size at a specified position.

Public Member Functions inherited from EntityFactory

- **EntityFactory** ()

  Default constructor for EntityFactory.
- virtual ~**EntityFactory** ()

  Virtual destructor for EntityFactory.

4.122.1 Detailed Description

Factory class to create utility entities such as power plants, water supplies, waste management facilities, and sewage systems.

Inherits from EntityFactory and provides methods to create different-sized utility entities. Supports creating small, medium, and large facilities, placing them at specified positions.

4.122.2 Member Function Documentation

4.122.2.1 createEntity()

```
Entity * UtilityFactory::createEntity (
            EntityType type,
            Size size,
            int xPos,
            int yPos)  [virtual]
```

Creates a utility entity of a specified type and size at a specified position.

Parameters

type	The type of utility entity to create (e.g., power plant, water supply).
size	The size of the entity to create (small, medium, or large).
xPos	The x-coordinate for the entity's position.
yPos	The y-coordinate for the entity's position.

Returns

A pointer to the created Entity.

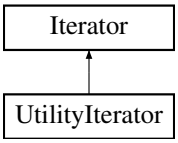
Implements EntityFactory.

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

- src/factory/utility/UtilityFactory.h
- src/factory/utility/UtilityFactory.cpp

4.123 UtilityIterator Class Reference

Inheritance diagram for UtilityIterator:



Public Member Functions

- **UtilityIterator** ()

  Construct a new Utility Iterator object.
- ~**UtilityIterator** ()

  Destroy the Utility Iterator object.
- UtilityIterator (std::vector< std::vector< Entity * > > &grid)

  Construct a new Utility Iterator object with grid.
- void first ()

  Resets the iterator to the first unvisited Utility instance.
- void next ()

  Advances to the next unvisited Utility instance.
- bool hasNext ()

  Checks if there is another unvisited Utility instance.
- Entity * current ()

  Returns the current Utility instance pointed to by the iterator.

Public Member Functions inherited from Iterator

- **Iterator** ()

  Construct a new Iterator object, initializing row and column to zero.
- virtual ~**Iterator** ()

  Destroy the Iterator object.
- **Iterator** (std::vector< std::vector< Entity * > > &grid)
- virtual int getRow ()

  Get the current row index of the iterator.
- virtual int getCol ()

  Get the current column index of the iterator.

Additional Inherited Members

Protected Member Functions inherited from Iterator

- bool isVisited (Entity *entity)

  Check if the specified entity has been visited.
- void markVisited (Entity *entity)

  Mark the specified entity as visited.

Protected Attributes inherited from Iterator

- std::vector< std::vector< Entity * > > **grid**
- std::vector< std::vector< Entity * > >::iterator **currRow**
- std::vector< Entity * >::iterator **curr**
- int **row**
- int **col**
- std::unordered_set< Entity * > **visitedEntities**

4.123.1 Constructor & Destructor Documentation

4.123.1.1 UtilityIterator()

```
UtilityIterator::UtilityIterator (
            std::vector< std::vector< Entity * > > & grid)
```

Construct a new Utility Iterator object with grid.

Parameters

grid	

4.123.2 Member Function Documentation

4.123.2.1 current()

```
Entity * UtilityIterator::current ()  [virtual]
```

Returns the current Utility instance pointed to by the iterator.

Returns

A pointer to the current Utility instance

Implements Iterator.

4.123.2.2 first()

```
void UtilityIterator::first ()  [virtual]
```

Resets the iterator to the first unvisited Utility instance.

Implements Iterator.

4.123.2.3 hasNext()

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

Checks if there is another unvisited Utility instance.

Returns

true if there is another unvisited Utility, false otherwise

Implements Iterator.

4.123.2.4 next()

```
void UtilityIterator::next ()  [virtual]
```

Advances to the next unvisited Utility instance.

Implements Iterator.

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

- src/iterators/utility/UtilityIterator.h
- src/iterators/utility/UtilityIterator.cpp

4.124 UtilityManager Class Reference

Responsible for creating and managing utilities within the city, handling utility upgrades, and gathering utility-related statistics.

```
#include <UtilityManager.h>
```

Public Member Functions

- **UtilityManager** ()

  Constructs a new UtilityManager instance.
- ~**UtilityManager** ()

  Destroys the UtilityManager instance, freeing resources.
- bool buildUtility (EntityType type, Size size, int x, int y)

  Builds a utility of the specified type, size, and location within the city.
- int getElectricityProduction ()

  Gets the total electricity production across all utilities in the city.
- int getElectricityConsumption ()

  Gets the total electricity consumption across all utilities in the city.
- int getWaterProduction ()

  Gets the total water production across all utilities in the city.
- int getWaterConsumption ()

  Gets the total water consumption across all utilities in the city.
- int getWasteProduction ()

Gets the total waste production across all utilities in the city.

- int getWasteConsumption ()

  Gets the total waste consumption across all utilities in the city.
- int getSewageProduction ()

  Gets the total sewage production across all utilities in the city.
- int getSewageConsumption ()

  Gets the total sewage consumption across all utilities in the city.
- std::vector< Utility * > getAllUtilities ()

  Retrieves a list of all utility entities within the city.
- std::vector< Utility * > getAllWaterSupplies ()

  Retrieves a list of all water supply utilities within the city.
- std::vector< Utility * > getAllPowerPlants ()

  Retrieves a list of all power plant utilities within the city.
- std::vector< Utility * > getAllWasteManagements ()

  Retrieves a list of all waste management utilities within the city.
- std::vector< Utility * > getAllSewageSystems ()

  Retrieves a list of all sewage system utilities within the city.
- bool canAffordUpgrade (Utility *utility)

  Checks if the city has sufficient resources to afford an upgrade for a specified utility.
- bool upgrade (Utility *&utility)

  Upgrades a specified utility, if possible.

4.124.1 Detailed Description

Responsible for creating and managing utilities within the city, handling utility upgrades, and gathering utility-related statistics.

4.124.2 Member Function Documentation

4.124.2.1 buildUtility()

```
bool UtilityManager::buildUtility (
            EntityType type,
            Size size,
            int x,
            int y)
```

Builds a utility of the specified type, size, and location within the city.

Parameters

type	Type of utility to build (e.g., PowerPlant, WaterSupply).
size	Size of the utility (e.g., small, medium, large).
x	X-coordinate in the city grid.
y	Y-coordinate in the city grid.

4.124.2.2 canAffordUpgrade()

```
bool UtilityManager::canAffordUpgrade (
            Utility * utility)
```

Checks if the city has sufficient resources to afford an upgrade for a specified utility.

Parameters

utility	Pointer to the utility to check.

Returns

True if the upgrade is affordable, false otherwise.

4.124.2.3 getAllPowerPlants()

```
std::vector< Utility * > UtilityManager::getAllPowerPlants ()
```

Retrieves a list of all power plant utilities within the city.

Returns

Vector of pointers to PowerPlant Utility objects.

4.124.2.4 getAllSewageSystems()

```
std::vector< Utility * > UtilityManager::getAllSewageSystems ()
```

Retrieves a list of all sewage system utilities within the city.

Returns

Vector of pointers to SewageSystem Utility objects.

4.124.2.5 getAllUtilities()

```
std::vector< Utility * > UtilityManager::getAllUtilities ()
```

Retrieves a list of all utility entities within the city.

Returns

Vector of pointers to Utility objects.

4.124.2.6 getAllWasteManagements()

```
std::vector< Utility * > UtilityManager::getAllWasteManagements ()
```

Retrieves a list of all waste management utilities within the city.

Returns

Vector of pointers to WasteManagement Utility objects.

4.124.2.7 getAllWaterSupplies()

```
std::vector< Utility * > UtilityManager::getAllWaterSupplies ()
```

Retrieves a list of all water supply utilities within the city.

Returns

Vector of pointers to WaterSupply Utility objects.

4.124.2.8 getElectricityConsumption()

```
int UtilityManager::getElectricityConsumption ()
```

Gets the total electricity consumption across all utilities in the city.

Returns

Total electricity consumption.

4.124.2.9 getElectricityProduction()

```
int UtilityManager::getElectricityProduction ()
```

Gets the total electricity production across all utilities in the city.

Returns

Total electricity production.

4.124.2.10 getSewageConsumption()

```
int UtilityManager::getSewageConsumption ()
```

Gets the total sewage consumption across all utilities in the city.

Returns

Total sewage consumption.

4.124.2.11 getSewageProduction()

```
int UtilityManager::getSewageProduction ()
```

Gets the total sewage production across all utilities in the city.

Returns

Total sewage production.

4.124.2.12 getWasteConsumption()

```
int UtilityManager::getWasteConsumption ()
```

Gets the total waste consumption across all utilities in the city.

Returns

Total waste consumption.

4.124.2.13 getWasteProduction()

```
int UtilityManager::getWasteProduction ()
```

Gets the total waste production across all utilities in the city.

Returns

Total waste production.

4.124.2.14 getWaterConsumption()

```
int UtilityManager::getWaterConsumption ()
```

Gets the total water consumption across all utilities in the city.

Returns

Total water consumption.

4.124.2.15 getWaterProduction()

```
int UtilityManager::getWaterProduction ()
```

Gets the total water production across all utilities in the city.

Returns

Total water production.

4.124.2.16 upgrade()

```
bool UtilityManager::upgrade (
            Utility *& utility)
```

Upgrades a specified utility, if possible.

Parameters

utility	Reference to the pointer of the utility to be upgraded.

Returns

True if the upgrade was successful, false otherwise.

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

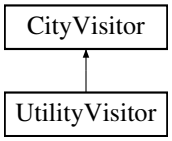
- src/managers/UtilityManager.h
- src/managers/UtilityManager.cpp

4.125 UtilityVisitor Class Reference

Visitor class for collecting data on utility outputs and handling capacities in a city.

`#include <UtilityVisitor.h>`

Inheritance diagram for UtilityVisitor:



Public Member Functions

- **UtilityVisitor** ()

  Constructs a UtilityVisitor with zeroed utility values.
- virtual ~**UtilityVisitor** ()

  Default destructor.
- void visit (City *city) override

  Visits a city to collect utility data from various utility entities.
- int getTotalElectricity () const

  Gets the total electricity output.
- int getTotalWater () const

  Gets the total water output.
- int getTotalSewageHandled () const

  Gets the total sewage handling capacity.
- int getTotalWasteHandled () const

  Gets the total waste handling capacity.

Public Member Functions inherited from CityVisitor

- **CityVisitor** ()=default

  Default constructor.
- virtual ~**CityVisitor** ()=default

  Default destructor.

4.125.1 Detailed Description

Visitor class for collecting data on utility outputs and handling capacities in a city.

4.125.2 Member Function Documentation

4.125.2.1 getTotalElectricity()

```
int UtilityVisitor::getTotalElectricity () const
```

Gets the total electricity output.

Returns

Total electricity generated by power plants.

4.125.2.2 getTotalSewageHandled()

```
int UtilityVisitor::getTotalSewageHandled () const
```

Gets the total sewage handling capacity.

Returns

Total sewage capacity from sewage systems.

4.125.2.3 getTotalWasteHandled()

```
int UtilityVisitor::getTotalWasteHandled () const
```

Gets the total waste handling capacity.

Returns

Total waste handled by waste management facilities.

4.125.2.4 getTotalWater()

```
int UtilityVisitor::getTotalWater () const
```

Gets the total water output.

Returns

Total water supplied by water sources.

4.125.2.5 visit()

```
void UtilityVisitor::visit (
            City * city)  [override], [virtual]
```

Visits a city to collect utility data from various utility entities.

Parameters

city	Pointer to the City object being visited.

Implements CityVisitor.

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

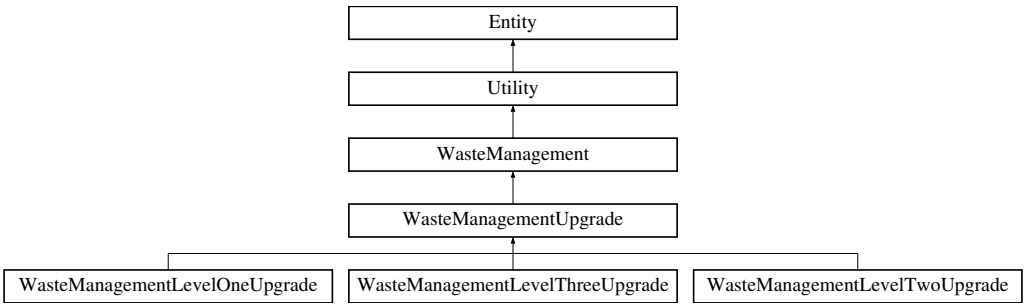
- src/visitors/utility/UtilityVisitor.h
- src/visitors/utility/UtilityVisitor.cpp

4.126 WasteManagement Class Reference

Represents a waste management facility in the city builder simulation.

`#include <WasteManagement.h>`

Inheritance diagram for WasteManagement:



Public Member Functions

- WasteManagement (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a WasteManagement object with specified attributes.
- WasteManagement (WasteManagement ∗waste)

  Copy constructor for the WasteManagement class.
- virtual ∼**WasteManagement** ()

  Destructor for the WasteManagement object.
- void update () override

  Updates the state of the waste management facility.
- Entity ∗ clone () override

  Clones the current WasteManagement object.
- Entity ∗ upgrade () override

  Upgrades the current utility to the next level.

Public Member Functions inherited from Utility

- Utility (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a Utility object with the specified parameters.
- Utility (Utility *utility)

  Copy constructor for the Utility class.
- virtual ~**Utility** ()

  Destructor for the Utility object.
- virtual int getOutput ()

  Retrieves the output of the utility.
- void setOutput (int output)

  Sets the output value of the utility.
- virtual Cost getCost ()

  Retrieves the cost of the utility or its upgraded version.
- virtual int getLevel ()

  Gets the level of the utility.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

Subscribes this entity as an observer of another entity.

- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).
- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.126.1 Detailed Description

Represents a waste management facility in the city builder simulation.

The WasteManagement class is a type of Utility that handles waste disposal for the city.

4.126.2 Constructor & Destructor Documentation

4.126.2.1 WasteManagement() [1/2]

```
WasteManagement::WasteManagement (
            EntityConfig ec,
            Size size,
            int xPos,
            int yPos)
```

Constructs a WasteManagement object with specified attributes.

Initializes a WasteManagement facility with detailed parameters, including utility consumption, effects, and dimensions.

Parameters

ec	EntityConfig.
size	Size.
xPos	xPosition
yPos	yPosition

4.126.2.2 WasteManagement() [2/2]

```
WasteManagement::WasteManagement (
            WasteManagement * waste)
```

Copy constructor for the WasteManagement class.

Creates a new WasteManagement object by copying the attributes of an existing WasteManagement.

Parameters

waste	Pointer to the existing WasteManagement object to be copied.

4.126.3 Member Function Documentation

4.126.3.1 clone()

```
Entity * WasteManagement::clone ()  [override], [virtual]
```

Clones the current WasteManagement object.

Creates and returns a copy of the current WasteManagement instance.

Returns

A pointer to the newly cloned WasteManagement object.

Implements Utility.

Reimplemented in WasteManagementLevelOneUpgrade, WasteManagementLevelThreeUpgrade, WasteManagementLevelTwoUpgra and WasteManagementUpgrade.

4.126.3.2 update()

```
void WasteManagement::update ()  [override], [virtual]
```

Updates the state of the waste management facility.

Defines the specific behavior of the WasteManagement facility when it is updated in the simulation.

Implements Utility.

Reimplemented in WasteManagementLevelOneUpgrade, WasteManagementLevelThreeUpgrade, WasteManagementLevelTwoUpgra and WasteManagementUpgrade.

4.126.3.3 upgrade()

```
Entity * WasteManagement::upgrade ()  [override], [virtual]
```

Upgrades the current utility to the next level.

Returns

A pointer to the upgraded utility instance, or nullptr if already at maximum level.

Implements Utility.

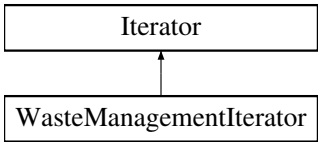
Reimplemented in WasteManagementLevelOneUpgrade, WasteManagementLevelThreeUpgrade, WasteManagementLevelTwoUpgra and WasteManagementUpgrade.

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

- src/entities/utility/wastemanagement/WasteManagement.h
- src/entities/utility/wastemanagement/WasteManagement.cpp

4.127 WasteManagementIterator Class Reference

Inheritance diagram for WasteManagementIterator:



Public Member Functions

- **WasteManagementIterator** ()

  Construct a new Waste Management Iterator object.
- ∼**WasteManagementIterator** ()

  Destroy the Waste Management Iterator object.
- WasteManagementIterator (std::vector< std::vector< Entity * > > &grid)

  Construct a new Waste Management Iterator object with grid.
- void first ()

  Resets the iterator to the first unvisited WasteManagement.
- void next ()

  Advances to the next unvisited WasteManagement.
- bool hasNext ()

  Checks if there is another unvisited WasteManagement.
- Entity * current ()

  Returns the current WasteManagement.

Public Member Functions inherited from Iterator

- **Iterator** ()

  Construct a new Iterator object, initializing row and column to zero.
- virtual ∼**Iterator** ()

  Destroy the Iterator object.
- **Iterator** (std::vector< std::vector< Entity * > > &grid)
- virtual int getRow ()

  Get the current row index of the iterator.
- virtual int getCol ()

  Get the current column index of the iterator.

Additional Inherited Members

Protected Member Functions inherited from Iterator

- bool isVisited (Entity *entity)

  Check if the specified entity has been visited.
- void markVisited (Entity *entity)

  Mark the specified entity as visited.

Protected Attributes inherited from Iterator

- std::vector< std::vector< Entity * > > **grid**
- std::vector< std::vector< Entity * > >::iterator **currRow**
- std::vector< Entity * >::iterator **curr**
- int **row**
- int **col**
- std::unordered_set< Entity * > **visitedEntities**

4.127.1 Constructor & Destructor Documentation

4.127.1.1 WasteManagementIterator()

```
WasteManagementIterator::WasteManagementIterator (
            std::vector< std::vector< Entity * > > & grid)
```

Construct a new Waste Management Iterator object with grid.

Parameters

grid	

4.127.2 Member Function Documentation

4.127.2.1 current()

```
Entity * WasteManagementIterator::current ()  [virtual]
```

Returns the current WasteManagement.

Returns

Entity*

Implements Iterator.

4.127.2.2 first()

```
void WasteManagementIterator::first ()  [virtual]
```

Resets the iterator to the first unvisited WasteManagement.

Implements Iterator.

4.127.2.3 hasNext()

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

Checks if there is another unvisited WasteManagement.

Returns

true if there is another unvisited WasteManagement, false otherwise

Implements Iterator.

4.127.2.4 next()

```
void WasteManagementIterator::next ()  [virtual]
```

Advances to the next unvisited WasteManagement.

Implements Iterator.

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

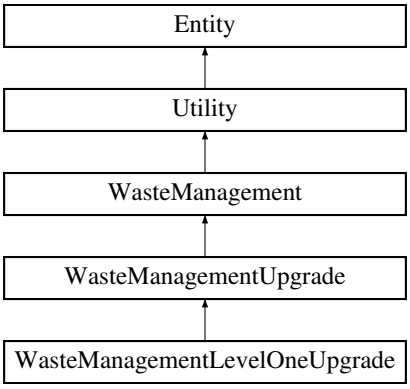
- src/iterators/utility/WasteManagementIterator.h
- src/iterators/utility/WasteManagementIterator.cpp

4.128 WasteManagementLevelOneUpgrade Class Reference

Represents the first level upgrade to a WasteManagement entity.

`#include <WasteManagementLevelOneUpgrade.h>`

Inheritance diagram for WasteManagementLevelOneUpgrade:



Public Member Functions

- WasteManagementLevelOneUpgrade (WasteManagement ∗waste)

  Constructs a WasteManagementLevelOneUpgrade object.
- WasteManagementLevelOneUpgrade (WasteManagementLevelOneUpgrade ∗wMLOU)

  Copy constructor for WasteManagementLevelOneUpgrade.
- ∼WasteManagementLevelOneUpgrade ()

  Destructor for WasteManagementLevelOneUpgrade.
- void update () override

  Updates the state of the upgraded waste management system.
- Entity ∗ clone () override

  Clones the current WasteManagementLevelOneUpgrade object.
- Entity ∗ upgrade () override

  Upgrades the current utility to the next level.
- int getOutput () override

  Retrieves the upgraded waste management system's output.
- Cost getCost () override

  Retrieves the cost of the utility or its upgraded version.
- int getLevel () override

  Gets the level of the waste management upgrade.

Public Member Functions inherited from WasteManagementUpgrade

- WasteManagementUpgrade (WasteManagement ∗waste)

  Constructs a WasteManagementUpgrade object based on an existing WasteManagement.
- WasteManagementUpgrade (WasteManagementUpgrade ∗wMU)

  Copy constructor for the WasteManagementUpgrade class.
- virtual ∼**WasteManagementUpgrade** ()

  Destructor for the WasteManagementUpgrade object.

Public Member Functions inherited from WasteManagement

- WasteManagement (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a WasteManagement object with specified attributes.
- WasteManagement (WasteManagement *waste)

  Copy constructor for the WasteManagement class.
- virtual ~**WasteManagement** ()

  Destructor for the WasteManagement object.

Public Member Functions inherited from Utility

- Utility (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a Utility object with the specified parameters.
- Utility (Utility *utility)

  Copy constructor for the Utility class.
- virtual ~**Utility** ()

  Destructor for the Utility object.
- void setOutput (int output)

  Sets the output value of the utility.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

Sets the symbol of the entity.

- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from WasteManagementUpgrade

- WasteManagement * **wasteManagement**

  Pointer to the original WasteManagement that is being upgraded.

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).

- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.128.1 Detailed Description

Represents the first level upgrade to a WasteManagement entity.

The WasteManagementLevelOneUpgrade class enhances the base functionality of a WasteManagement by increasing its output. This class is the first upgrade level in a series of potential waste management improvements.

4.128.2 Constructor & Destructor Documentation

4.128.2.1 WasteManagementLevelOneUpgrade() [1/2]

```
WasteManagementLevelOneUpgrade::WasteManagementLevelOneUpgrade (
            WasteManagement * waste)
```

Constructs a WasteManagementLevelOneUpgrade object.

Initializes the upgrade by enhancing the specified WasteManagement with a level one upgrade.

Parameters

waste	Pointer to the original WasteManagement to be upgraded.

4.128.2.2 WasteManagementLevelOneUpgrade() [2/2]

```
WasteManagementLevelOneUpgrade::WasteManagementLevelOneUpgrade (
            WasteManagementLevelOneUpgrade * wMLOU)
```

Copy constructor for WasteManagementLevelOneUpgrade.

Creates a new WasteManagementLevelOneUpgrade object by copying the attributes of an existing WasteManagementLevelOneUpgra object.

Parameters

wMLOU	Pointer to the existing WasteManagementLevelOneUpgrade to be copied.

4.128.2.3 ~WasteManagementLevelOneUpgrade()

```
WasteManagementLevelOneUpgrade::~WasteManagementLevelOneUpgrade ()
```

Destructor for WasteManagementLevelOneUpgrade.

Cleans up any resources associated with the upgrade.

4.128.3 Member Function Documentation

4.128.3.1 clone()

```
Entity * WasteManagementLevelOneUpgrade::clone ()  [override], [virtual]
```

Clones the current WasteManagementLevelOneUpgrade object.

Creates a new instance of WasteManagementLevelOneUpgrade with the same attributes as the current object.

Returns

A pointer to the newly cloned WasteManagementLevelOneUpgrade object.

Implements WasteManagementUpgrade.

4.128.3.2 getCost()

```
Cost WasteManagementLevelOneUpgrade::getCost ()  [override], [virtual]
```

Retrieves the cost of the utility or its upgraded version.

Returns

A Cost object representing the monetary and material costs.

Implements WasteManagementUpgrade.

4.128.3.3 getLevel()

```
int WasteManagementLevelOneUpgrade::getLevel ()  [override], [virtual]
```

Gets the level of the waste management upgrade.

Returns

The level of the waste management upgrade.

Reimplemented from Utility.

4.128.3.4 getOutput()

```
int WasteManagementLevelOneUpgrade::getOutput ()  [override], [virtual]
```

Retrieves the upgraded waste management system's output.

Returns the output of the level one upgraded waste management system.

Returns

The updated output as an integer.

Implements WasteManagementUpgrade.

4.128.3.5 update()

```
void WasteManagementLevelOneUpgrade::update ()  [override], [virtual]
```

Updates the state of the upgraded waste management system.

Implements specific behavior for the waste management system after applying the level one upgrade.

Implements WasteManagementUpgrade.

4.128.3.6 upgrade()

```
Entity * WasteManagementLevelOneUpgrade::upgrade ()  [override], [virtual]
```

Upgrades the current utility to the next level.

Returns

A pointer to the upgraded utility instance, or nullptr if already at maximum level.

Implements WasteManagementUpgrade.

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

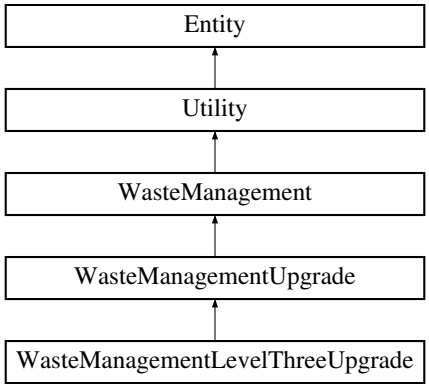
- src/entities/utility/wastemanagement/WasteManagementLevelOneUpgrade.h
- src/entities/utility/wastemanagement/WasteManagementLevelOneUpgrade.cpp

4.129 WasteManagementLevelThreeUpgrade Class Reference

Represents the third level upgrade to a WasteManagement entity.

`#include <WasteManagementLevelThreeUpgrade.h>`

Inheritance diagram for WasteManagementLevelThreeUpgrade:



Public Member Functions

- WasteManagementLevelThreeUpgrade (WasteManagement ∗waste)

  Constructs a WasteManagementLevelThreeUpgrade object.
- WasteManagementLevelThreeUpgrade (WasteManagementLevelThreeUpgrade ∗wMLTU)

  Copy constructor for WasteManagementLevelThreeUpgrade.
- ∼WasteManagementLevelThreeUpgrade ()

  Destructor for WasteManagementLevelThreeUpgrade.
- void update () override

  Updates the state of the upgraded waste management system.
- Entity ∗ clone () override

  Clones the current WasteManagementLevelThreeUpgrade object.
- Entity ∗ upgrade () override

  Upgrades the current utility to the next level.
- int getOutput () override

  Retrieves the upgraded waste management system's output.
- Cost getCost () override

  Retrieves the cost of the utility or its upgraded version.
- int getLevel () override

  Gets the level of the waste management upgrade.

Public Member Functions inherited from WasteManagementUpgrade

- WasteManagementUpgrade (WasteManagement ∗waste)

  Constructs a WasteManagementUpgrade object based on an existing WasteManagement.
- WasteManagementUpgrade (WasteManagementUpgrade ∗wMU)

  Copy constructor for the WasteManagementUpgrade class.
- virtual ∼**WasteManagementUpgrade** ()

  Destructor for the WasteManagementUpgrade object.

Public Member Functions inherited from WasteManagement

- WasteManagement (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a WasteManagement object with specified attributes.
- WasteManagement (WasteManagement *waste)

  Copy constructor for the WasteManagement class.
- virtual ~**WasteManagement** ()

  Destructor for the WasteManagement object.

Public Member Functions inherited from Utility

- Utility (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a Utility object with the specified parameters.
- Utility (Utility *utility)

  Copy constructor for the Utility class.
- virtual ~**Utility** ()

  Destructor for the Utility object.
- void setOutput (int output)

  Sets the output value of the utility.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

Sets the symbol of the entity.

- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from WasteManagementUpgrade

- WasteManagement * **wasteManagement**

  Pointer to the original WasteManagement that is being upgraded.

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).

- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.129.1 Detailed Description

Represents the third level upgrade to a WasteManagement entity.

The WasteManagementLevelThreeUpgrade class enhances the base functionality of a WasteManagement system by increasing its efficiency. This class is the third upgrade level in a series of potential waste management improvements.

4.129.2 Constructor & Destructor Documentation

4.129.2.1 WasteManagementLevelThreeUpgrade() [1/2]

```
WasteManagementLevelThreeUpgrade::WasteManagementLevelThreeUpgrade (
            WasteManagement * waste)
```

Constructs a WasteManagementLevelThreeUpgrade object.

Initializes the upgrade by enhancing the specified WasteManagement with a level three upgrade.

Parameters

waste	Pointer to the original WasteManagement to be upgraded.

4.129.2.2 WasteManagementLevelThreeUpgrade() [2/2]

```
WasteManagementLevelThreeUpgrade::WasteManagementLevelThreeUpgrade (
            WasteManagementLevelThreeUpgrade * wMLTU)
```

Copy constructor for WasteManagementLevelThreeUpgrade.

Creates a new WasteManagementLevelThreeUpgrade object by copying the attributes of an existing WasteManagementLevelThreeUpgrade object.

Parameters

wMLTU	Pointer to the existing WasteManagementLevelThreeUpgrade to be copied.

4.129.2.3 ~WasteManagementLevelThreeUpgrade()

```
WasteManagementLevelThreeUpgrade::~WasteManagementLevelThreeUpgrade ()
```

Destructor for WasteManagementLevelThreeUpgrade.

Cleans up any resources associated with the upgrade.

4.129.3 Member Function Documentation

4.129.3.1 clone()

```
Entity * WasteManagementLevelThreeUpgrade::clone ()  [override], [virtual]
```

Clones the current WasteManagementLevelThreeUpgrade object.

Creates a new instance of WasteManagementLevelThreeUpgrade with the same attributes as the current object.

Returns

A pointer to the newly cloned WasteManagementLevelThreeUpgrade object.

Implements WasteManagementUpgrade.

4.129.3.2 getCost()

```
Cost WasteManagementLevelThreeUpgrade::getCost ()  [override], [virtual]
```

Retrieves the cost of the utility or its upgraded version.

Returns

A Cost object representing the monetary and material costs.

Implements WasteManagementUpgrade.

4.129.3.3 getLevel()

```
int WasteManagementLevelThreeUpgrade::getLevel ()  [override], [virtual]
```

Gets the level of the waste management upgrade.

Returns

The level of the waste management upgrade.

Reimplemented from Utility.

4.129.3.4 getOutput()

```
int WasteManagementLevelThreeUpgrade::getOutput ()  [override], [virtual]
```

Retrieves the upgraded waste management system's output.

Returns the efficiency output of the level three upgraded waste management system.

Returns

The updated efficiency output as an integer.

Implements WasteManagementUpgrade.

4.129.3.5 update()

```
void WasteManagementLevelThreeUpgrade::update ()  [override], [virtual]
```

Updates the state of the upgraded waste management system.

Implements specific behavior for the waste management system after applying the level three upgrade.

Implements WasteManagementUpgrade.

4.129.3.6 upgrade()

```
Entity * WasteManagementLevelThreeUpgrade::upgrade ()  [override], [virtual]
```

Upgrades the current utility to the next level.

Returns

A pointer to the upgraded utility instance, or nullptr if already at maximum level.

Implements WasteManagementUpgrade.

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

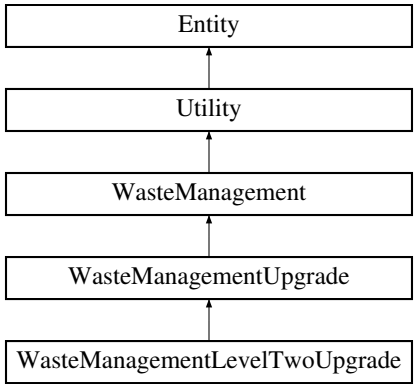
- src/entities/utility/wastemanagement/WasteManagementLevelThreeUpgrade.h
- src/entities/utility/wastemanagement/WasteManagementLevelThreeUpgrade.cpp

4.130 WasteManagementLevelTwoUpgrade Class Reference

Represents the second level upgrade to a WasteManagement entity.

`#include <WasteManagementLevelTwoUpgrade.h>`

Inheritance diagram for WasteManagementLevelTwoUpgrade:



Public Member Functions

- WasteManagementLevelTwoUpgrade (WasteManagement ∗waste)

  Constructs a WasteManagementLevelTwoUpgrade object.
- WasteManagementLevelTwoUpgrade (WasteManagementLevelTwoUpgrade ∗wMLTU)

  Copy constructor for WasteManagementLevelTwoUpgrade.
- ∼**WasteManagementLevelTwoUpgrade** ()

  Destructor for WasteManagementLevelTwoUpgrade.
- void update () override

  Updates the state of the upgraded waste management system.
- Entity ∗ clone () override

  Clones the current WasteManagementLevelTwoUpgrade object.
- Entity ∗ upgrade () override

  Upgrades the current utility to the next level.
- int getOutput () override

  Retrieves the upgraded waste management system's output.
- Cost getCost () override

  Retrieves the cost of the utility or its upgraded version.
- int getLevel () override

  Gets the level of the waste management upgrade.

Public Member Functions inherited from WasteManagementUpgrade

- WasteManagementUpgrade (WasteManagement ∗waste)

  Constructs a WasteManagementUpgrade object based on an existing WasteManagement.
- WasteManagementUpgrade (WasteManagementUpgrade ∗wMU)

  Copy constructor for the WasteManagementUpgrade class.
- virtual ∼**WasteManagementUpgrade** ()

  Destructor for the WasteManagementUpgrade object.

Public Member Functions inherited from WasteManagement

- WasteManagement (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a WasteManagement object with specified attributes.
- WasteManagement (WasteManagement *waste)

  Copy constructor for the WasteManagement class.
- virtual ~**WasteManagement** ()

  Destructor for the WasteManagement object.

Public Member Functions inherited from Utility

- Utility (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a Utility object with the specified parameters.
- Utility (Utility *utility)

  Copy constructor for the Utility class.
- virtual ~**Utility** ()

  Destructor for the Utility object.
- void setOutput (int output)

  Sets the output value of the utility.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

Sets the symbol of the entity.

- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from WasteManagementUpgrade

- WasteManagement * **wasteManagement**

  Pointer to the original WasteManagement that is being upgraded.

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).

- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.130.1 Detailed Description

Represents the second level upgrade to a WasteManagement entity.

The WasteManagementLevelTwoUpgrade class enhances the base functionality of a WasteManagement system by increasing its output. This is the second upgrade level in the series.

4.130.2 Constructor & Destructor Documentation

4.130.2.1 WasteManagementLevelTwoUpgrade() [1/2]

```
WasteManagementLevelTwoUpgrade::WasteManagementLevelTwoUpgrade (
            WasteManagement * waste)
```

Constructs a WasteManagementLevelTwoUpgrade object.

Enhances the specified WasteManagement system with a level two upgrade.

Parameters

waste	Pointer to the original WasteManagement to be upgraded.

4.130.2.2 WasteManagementLevelTwoUpgrade() [2/2]

```
WasteManagementLevelTwoUpgrade::WasteManagementLevelTwoUpgrade (
            WasteManagementLevelTwoUpgrade * wMLTU)
```

Copy constructor for WasteManagementLevelTwoUpgrade.

Copies the attributes of an existing WasteManagementLevelTwoUpgrade object.

Parameters

wMLTU	Pointer to the existing object to be copied.

4.130.3 Member Function Documentation

4.130.3.1 clone()

```
Entity * WasteManagementLevelTwoUpgrade::clone ()  [override], [virtual]
```

Clones the current WasteManagementLevelTwoUpgrade object.

Returns

A pointer to the newly cloned object.

Implements WasteManagementUpgrade.

4.130.3.2 getCost()

```
Cost WasteManagementLevelTwoUpgrade::getCost ()  [override], [virtual]
```

Retrieves the cost of the utility or its upgraded version.

Returns

A Cost object representing the monetary and material costs.

Implements WasteManagementUpgrade.

4.130.3.3 getLevel()

```
int WasteManagementLevelTwoUpgrade::getLevel ()  [override], [virtual]
```

Gets the level of the waste management upgrade.

Returns

The level of the waste management upgrade.

Reimplemented from Utility.

4.130.3.4 getOutput()

```
int WasteManagementLevelTwoUpgrade::getOutput ()  [override], [virtual]
```

Retrieves the upgraded waste management system's output.

Returns the output of the level one upgraded waste management system.

Returns

The updated output as an integer.

Implements WasteManagementUpgrade.

4.130.3.5 update()

```
void WasteManagementLevelTwoUpgrade::update ()  [override], [virtual]
```

Updates the state of the upgraded waste management system.

Implements WasteManagementUpgrade.

4.130.3.6 upgrade()

```
Entity * WasteManagementLevelTwoUpgrade::upgrade ()  [override], [virtual]
```

Upgrades the current utility to the next level.

Returns

A pointer to the upgraded utility instance, or nullptr if already at maximum level.

Implements WasteManagementUpgrade.

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

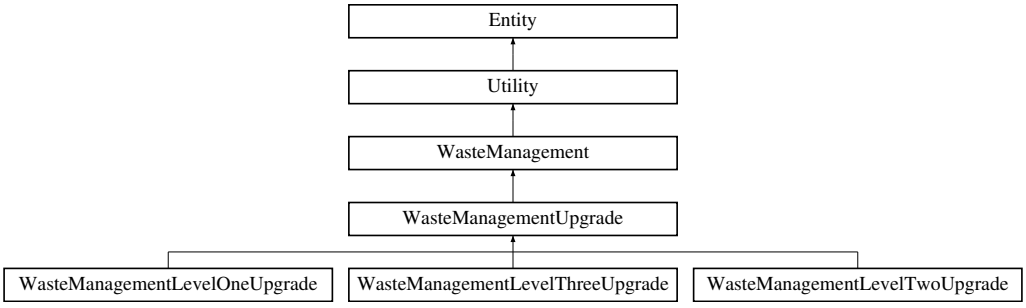
- src/entities/utility/wastemanagement/WasteManagementLevelTwoUpgrade.h
- src/entities/utility/wastemanagement/WasteManagementLevelTwoUpgrade.cpp

4.131 WasteManagementUpgrade Class Reference

Represents an upgrade to a WasteManagement entity in the city builder simulation.

```
#include <WasteManagementUpgrade.h>
```

Inheritance diagram for WasteManagementUpgrade:



Public Member Functions

- WasteManagementUpgrade (WasteManagement ∗waste)

  Constructs a WasteManagementUpgrade object based on an existing WasteManagement.
- WasteManagementUpgrade (WasteManagementUpgrade ∗wMU)

  Copy constructor for the WasteManagementUpgrade class.
- virtual ∼**WasteManagementUpgrade** ()

  Destructor for the WasteManagementUpgrade object.
- virtual void update ()=0

  Pure virtual function to update the upgraded waste management system.
- virtual Entity ∗ clone ()=0

  Pure virtual function to clone the upgraded waste management system.
- virtual Entity ∗ upgrade ()=0

  Upgrades the current utility to the next level.
- virtual int getOutput ()=0

  Retrieves the output of the upgraded waste management system.
- virtual Cost getCost ()=0

  Retrieves the cost of the utility or its upgraded version.

Public Member Functions inherited from WasteManagement

- WasteManagement (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a WasteManagement object with specified attributes.
- WasteManagement (WasteManagement ∗waste)

  Copy constructor for the WasteManagement class.
- virtual ∼**WasteManagement** ()

  Destructor for the WasteManagement object.

Public Member Functions inherited from Utility

- Utility (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a Utility object with the specified parameters.
- Utility (Utility ∗utility)

  Copy constructor for the Utility class.
- virtual ∼**Utility** ()

  Destructor for the Utility object.
- void setOutput (int output)

  Sets the output value of the utility.
- virtual int getLevel ()

  Gets the level of the utility.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Protected Attributes

- WasteManagement * **wasteManagement**

  Pointer to the original WasteManagement that is being upgraded.

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).
- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.131.1 Detailed Description

Represents an upgrade to a WasteManagement entity in the city builder simulation.

The WasteManagementUpgrade class extends the functionality of a WasteManagement, enhancing its capabilities and acting as a wrapper around the existing WasteManagement object.

4.131.2 Constructor & Destructor Documentation

4.131.2.1 WasteManagementUpgrade() [1/2]

```
WasteManagementUpgrade::WasteManagementUpgrade (
            WasteManagement * waste)
```

Constructs a WasteManagementUpgrade object based on an existing WasteManagement.

Initializes the upgrade with a reference to an existing WasteManagement, enhancing its features.

Parameters

waste	Pointer to the WasteManagement being upgraded.

4.131.2.2 WasteManagementUpgrade() [2/2]

```
WasteManagementUpgrade::WasteManagementUpgrade (
            WasteManagementUpgrade * wMU)
```

Copy constructor for the WasteManagementUpgrade class.

Creates a new WasteManagementUpgrade object by copying the attributes of an existing WasteManagementUpgrade.

Parameters

wMU	Pointer to the existing WasteManagementUpgrade object to be copied.

4.131.3 Member Function Documentation

4.131.3.1 clone()

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

Pure virtual function to clone the upgraded waste management system.

Returns

A pointer to a new cloned WasteManagementUpgrade object.

Reimplemented from WasteManagement.

Implemented in WasteManagementLevelOneUpgrade, WasteManagementLevelThreeUpgrade, and WasteManagementLevelTwoUpg

4.131.3.2 getCost()

```
virtual Cost WasteManagementUpgrade::getCost ()  [pure virtual]
```

Retrieves the cost of the utility or its upgraded version.

Returns

A Cost object representing the monetary and material costs.

Reimplemented from Utility.

Implemented in WasteManagementLevelOneUpgrade, WasteManagementLevelThreeUpgrade, and WasteManagementLevelTwoUpg

4.131.3.3 getOutput()

```
virtual int WasteManagementUpgrade::getOutput ()  [pure virtual]
```

Retrieves the output of the upgraded waste management system.

Returns

The output value as an integer.

Reimplemented from Utility.

Implemented in WasteManagementLevelOneUpgrade, WasteManagementLevelThreeUpgrade, and WasteManagementLevelTwoUpg

4.131.3.4 update()

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

Pure virtual function to update the upgraded waste management system.

Reimplemented from WasteManagement.

Implemented in WasteManagementLevelOneUpgrade, WasteManagementLevelThreeUpgrade, and WasteManagementLevelTwoUpg

4.131.3.5 upgrade()

```
virtual Entity * WasteManagementUpgrade::upgrade ()  [pure virtual]
```

Upgrades the current utility to the next level.

Returns

A pointer to the upgraded utility instance, or nullptr if already at maximum level.

Reimplemented from WasteManagement.

Implemented in WasteManagementLevelOneUpgrade, WasteManagementLevelThreeUpgrade, and WasteManagementLevelTwoUpg

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

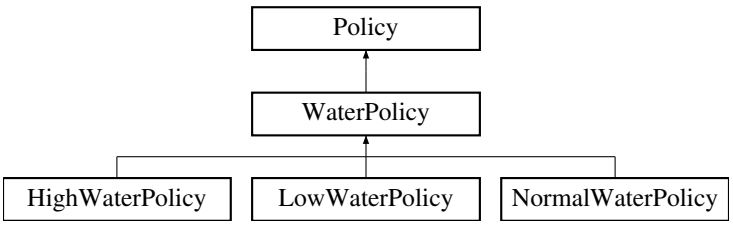
- src/entities/utility/wastemanagement/WasteManagementUpgrade.h
- src/entities/utility/wastemanagement/WasteManagementUpgrade.cpp

4.132 WaterPolicy Class Reference

Abstract class for WaterPolicy.

`#include <WaterPolicy.h>`

Inheritance diagram for WaterPolicy:



Public Member Functions

- WaterPolicy (const std::string &name, const std::string &detail)
  Constructor for WaterPolicy.
- virtual int calculateWaterUsage (int waterUsage)=0
  Pure virtual function to calculate water usage.
- virtual ~**WaterPolicy** ()
  Virtual destructor for WaterPolicy.

Public Member Functions inherited from Policy

- Policy (const std::string &name, const std::string &detail)
  Constructor for Policy.
- Memento * createMemento () const
  Creates a memento to store the current state of the policy.
- void setMemento (const Memento *memento)
  Sets the policy state from a memento.
- std::string getName () const
  Gets the name of the policy.
- std::string getDetail () const
  Gets the detail of the policy.

4.132.1 Detailed Description

Abstract class for WaterPolicy.

Defines the interface for calculating water usage based on different policy strategies.

4.132.2 Constructor & Destructor Documentation

4.132.2.1 WaterPolicy()

```
WaterPolicy::WaterPolicy (
            const std::string & name,
            const std::string & detail)  [inline]
```

Constructor for WaterPolicy.

Parameters

name	Name of the policy.
detail	Details describing the policy.

4.132.3 Member Function Documentation

4.132.3.1 calculateWaterUsage()

```
virtual int WaterPolicy::calculateWaterUsage (
            int waterUsage)  [pure virtual]
```

Pure virtual function to calculate water usage.

Parameters

waterUsage	Initial water usage.

Returns

int Modified water usage based on policy.

Implemented in HighWaterPolicy, LowWaterPolicy, and NormalWaterPolicy.

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

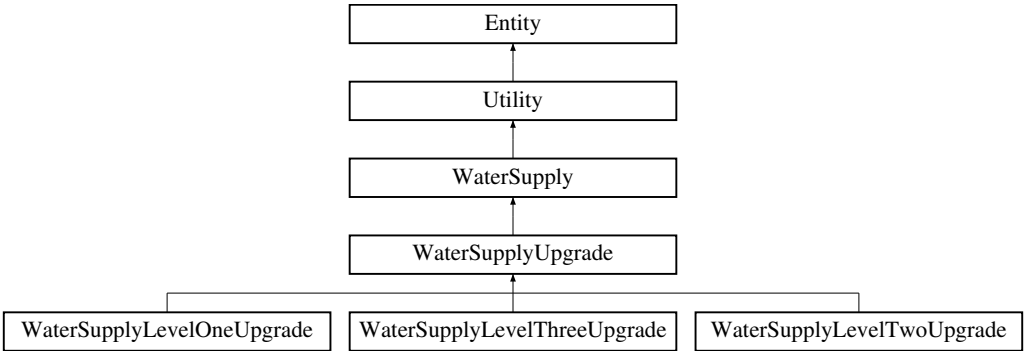
- src/policies/water/WaterPolicy.h

4.133 WaterSupply Class Reference

Represents a water supply system in the city builder simulation.

```
#include <WaterSupply.h>
```

Inheritance diagram for WaterSupply:



Public Member Functions

- WaterSupply (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a WaterSupply object with specified attributes.
- WaterSupply (WaterSupply *water)

  Copy constructor for the WaterSupply class.
- virtual ~**WaterSupply** ()

  Destructor for the WaterSupply object.
- void update () override

  Updates the state of the water supply system.
- Entity * clone () override

  Clones the current WaterSupply object.
- Entity * upgrade () override

  Upgrades the current utility to the next level.

Public Member Functions inherited from Utility

- Utility (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a Utility object with the specified parameters.
- Utility (Utility *utility)

  Copy constructor for the Utility class.
- virtual ~**Utility** ()

  Destructor for the Utility object.
- virtual int getOutput ()

  Retrieves the output of the utility.
- void setOutput (int output)

  Sets the output value of the utility.
- virtual Cost getCost ()

  Retrieves the cost of the utility or its upgraded version.
- virtual int getLevel ()

  Gets the level of the utility.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

Sets the Y-coordinate position of the entity.

- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.

- int **height**
  Height of the entity.
- int **xPosition**
  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**
  Y-coordinate of the entity's position (bottom left corner).
- Size **size**
  Size object representing the entity's dimensions.
- EntityType **type**
  The type of entity.
- State * **state**
  Pointer to the current state of the entity.
- int **revenue**
  Revenue generated by the entity.
- float **electricityConsumption**
  Electricity consumption of the entity.
- float **waterConsumption**
  Water consumption of the entity.
- std::vector< Entity * > **observers**
  List of other entities observing this entity.

4.133.1 Detailed Description

Represents a water supply system in the city builder simulation.

The WaterSupply class is a type of Utility responsible for providing water to the city.

4.133.2 Constructor & Destructor Documentation

4.133.2.1 WaterSupply() [1/2]

```
WaterSupply::WaterSupply (
            EntityConfig ec,
            Size size,
            int xPos,
            int yPos)
```

Constructs a WaterSupply object with specified attributes.

Initializes a WaterSupply facility with detailed parameters, including utility consumption, effects, and dimensions.

Parameters

ec	EntityConfig.
size	Size.
xPos	xPosition
yPos	yPosition

4.133.2.2 WaterSupply() [2/2]

```
WaterSupply::WaterSupply (
            WaterSupply * water)
```

Copy constructor for the WaterSupply class.

Creates a new WaterSupply object by copying the attributes of an existing WaterSupply.

Parameters

water	Pointer to the existing WaterSupply object to be copied.

4.133.3 Member Function Documentation

4.133.3.1 clone()

```
Entity * WaterSupply::clone ()  [override], [virtual]
```

Clones the current WaterSupply object.

Creates and returns a copy of the current WaterSupply instance.

Returns

A pointer to the newly cloned WaterSupply object.

Implements Utility.

Reimplemented in WaterSupplyLevelOneUpgrade, WaterSupplyLevelThreeUpgrade, WaterSupplyLevelTwoUpgrade, and WaterSupplyUpgrade.

4.133.3.2 update()

```
void WaterSupply::update ()  [override], [virtual]
```

Updates the state of the water supply system.

Defines the specific behavior of the WaterSupply system when it is updated in the simulation.

Implements Utility.

Reimplemented in WaterSupplyLevelOneUpgrade, WaterSupplyLevelThreeUpgrade, WaterSupplyLevelTwoUpgrade, and WaterSupplyUpgrade.

4.133.3.3 upgrade()

```
Entity * WaterSupply::upgrade ()  [override], [virtual]
```

Upgrades the current utility to the next level.

Returns

A pointer to the upgraded utility instance, or nullptr if already at maximum level.

Implements Utility.

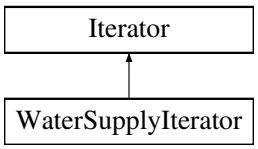
Reimplemented in WaterSupplyLevelOneUpgrade, WaterSupplyLevelThreeUpgrade, WaterSupplyLevelTwoUpgrade, and WaterSupplyUpgrade.

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

- src/entities/utility/watersupply/WaterSupply.h
- src/entities/utility/watersupply/WaterSupply.cpp

4.134 WaterSupplyIterator Class Reference

Inheritance diagram for WaterSupplyIterator:



Public Member Functions

- **WaterSupplyIterator** ()

  Construct a new Water Supply Iterator object.
- ~**WaterSupplyIterator** ()

  Destroy the Water Supply Iterator object.
- WaterSupplyIterator (std::vector< std::vector< Entity * > > &grid)

  Construct a new Water Supply Iterator object with grid.
- void first ()

  Resets the iterator to the first unvisited WaterSupply.
- void next ()

  Advances to the next unvisited WaterSupply.
- bool hasNext ()

  Checks if there is another unvisited WaterSupply.
- Entity * current ()

  Returns the current WaterSupply.

Public Member Functions inherited from Iterator

- **Iterator** ()

  Construct a new Iterator object, initializing row and column to zero.
- virtual ~**Iterator** ()

  Destroy the Iterator object.
- **Iterator** (std::vector< std::vector< Entity * > > &grid)
- virtual int getRow ()

  Get the current row index of the iterator.
- virtual int getCol ()

  Get the current column index of the iterator.

Additional Inherited Members

Protected Member Functions inherited from Iterator

- bool isVisited (Entity *entity)

  Check if the specified entity has been visited.
- void markVisited (Entity *entity)

  Mark the specified entity as visited.

Protected Attributes inherited from Iterator

- std::vector< std::vector< Entity * > > **grid**
- std::vector< std::vector< Entity * > >::iterator **currRow**
- std::vector< Entity * >::iterator **curr**
- int **row**
- int **col**
- std::unordered_set< Entity * > **visitedEntities**

4.134.1 Constructor & Destructor Documentation

4.134.1.1 WaterSupplyIterator()

```
WaterSupplyIterator::WaterSupplyIterator (
            std::vector< std::vector< Entity * > > & grid)
```

Construct a new Water Supply Iterator object with grid.

Parameters

grid	

4.134.2 Member Function Documentation

4.134.2.1 current()

```
Entity * WaterSupplyIterator::current ()  [virtual]
```

Returns the current WaterSupply.

Returns

Entity*

Implements Iterator.

4.134.2.2 first()

```
void WaterSupplyIterator::first ()  [virtual]
```

Resets the iterator to the first unvisited WaterSupply.

Implements Iterator.

4.134.2.3 hasNext()

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

Checks if there is another unvisited WaterSupply.

Returns

true if there is another unvisited WaterSupply, false otherwise

Implements Iterator.

4.134.2.4 next()

```
void WaterSupplyIterator::next () [virtual]
```

Advances to the next unvisited WaterSupply.

Implements Iterator.

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

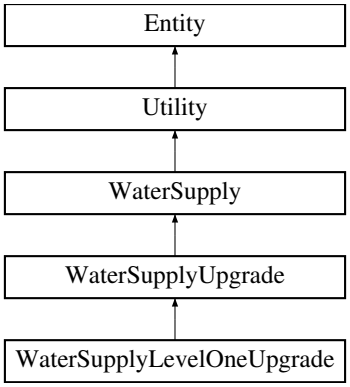
- src/iterators/utility/WaterSupplyIterator.h
- src/iterators/utility/WaterSupplyIterator.cpp

4.135 WaterSupplyLevelOneUpgrade Class Reference

Represents the first level upgrade to a WaterSupply entity.

```
#include <WaterSupplyLevelOneUpgrade.h>
```

Inheritance diagram for WaterSupplyLevelOneUpgrade:



Public Member Functions

- WaterSupplyLevelOneUpgrade (WaterSupply *water)

  Constructs a WaterSupplyLevelOneUpgrade object.
- WaterSupplyLevelOneUpgrade (WaterSupplyLevelOneUpgrade *wSLOU)

  Copy constructor for WaterSupplyLevelOneUpgrade.
- ~WaterSupplyLevelOneUpgrade ()

  Destructor for WaterSupplyLevelOneUpgrade.
- void update () override

  Updates the state of the upgraded water supply system.
- Entity * clone () override

  Clones the current WaterSupplyLevelOneUpgrade object.
- Entity * upgrade () override

  Upgrades the current utility to the next level.
- int getOutput () override

  Retrieves the upgraded water supply system's output.
- Cost getCost () override

  Retrieves the cost of the utility or its upgraded version.
- int getLevel () override

  Gets the level of the water supply upgrade.

Public Member Functions inherited from WaterSupplyUpgrade

- WaterSupplyUpgrade (WaterSupply *water)

  Constructs a WaterSupplyUpgrade object based on an existing WaterSupply.
- WaterSupplyUpgrade (WaterSupplyUpgrade *wSU)

  Copy constructor for the WaterSupplyUpgrade class.
- virtual ~**WaterSupplyUpgrade** ()

  Destructor for the WaterSupplyUpgrade object.

Public Member Functions inherited from WaterSupply

- WaterSupply (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a WaterSupply object with specified attributes.
- WaterSupply (WaterSupply *water)

  Copy constructor for the WaterSupply class.
- virtual ~**WaterSupply** ()

  Destructor for the WaterSupply object.

Public Member Functions inherited from Utility

- Utility (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a Utility object with the specified parameters.
- Utility (Utility *utility)

  Copy constructor for the Utility class.
- virtual ~**Utility** ()

  Destructor for the Utility object.
- void setOutput (int output)

  Sets the output value of the utility.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from WaterSupplyUpgrade

- WaterSupply * **waterSupply**

  Pointer to the original WaterSupply that is being upgraded.

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).
- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.135.1 Detailed Description

Represents the first level upgrade to a WaterSupply entity.

The WaterSupplyLevelOneUpgrade class enhances the base functionality of a WaterSupply by increasing its output. This class is the first upgrade level in a series of potential water supply improvements.

4.135.2 Constructor & Destructor Documentation

4.135.2.1 WaterSupplyLevelOneUpgrade() [1/2]

```
WaterSupplyLevelOneUpgrade::WaterSupplyLevelOneUpgrade (
            WaterSupply * water)
```

Constructs a WaterSupplyLevelOneUpgrade object.

Initializes the upgrade by enhancing the specified WaterSupply with a level one upgrade.

Parameters

water	Pointer to the original WaterSupply to be upgraded.

4.135.2.2 WaterSupplyLevelOneUpgrade() [2/2]

```
WaterSupplyLevelOneUpgrade::WaterSupplyLevelOneUpgrade (
            WaterSupplyLevelOneUpgrade * wSLOU)
```

Copy constructor for WaterSupplyLevelOneUpgrade.

Creates a new WaterSupplyLevelOneUpgrade object by copying the attributes of an existing WaterSupplyLevelOneUpgrade object.

Parameters

wSLOU	Pointer to the existing WaterSupplyLevelOneUpgrade to be copied.

4.135.2.3 ~WaterSupplyLevelOneUpgrade()

```
WaterSupplyLevelOneUpgrade::~WaterSupplyLevelOneUpgrade ()
```

Destructor for WaterSupplyLevelOneUpgrade.

Cleans up any resources associated with the upgrade.

4.135.3 Member Function Documentation

4.135.3.1 clone()

```
Entity * WaterSupplyLevelOneUpgrade::clone ()  [override], [virtual]
```

Clones the current WaterSupplyLevelOneUpgrade object.

Creates a new instance of WaterSupplyLevelOneUpgrade with the same attributes as the current object.

Returns

A pointer to the newly cloned WaterSupplyLevelOneUpgrade object.

Implements WaterSupplyUpgrade.

4.135.3.2 getCost()

```
Cost WaterSupplyLevelOneUpgrade::getCost () [override], [virtual]
```

Retrieves the cost of the utility or its upgraded version.

Returns

A Cost object representing the monetary and material costs.

Implements WaterSupplyUpgrade.

4.135.3.3 getLevel()

```
int WaterSupplyLevelOneUpgrade::getLevel () [override], [virtual]
```

Gets the level of the water supply upgrade.

Returns

The level of the water supply upgrade.

Reimplemented from Utility.

4.135.3.4 getOutput()

```
int WaterSupplyLevelOneUpgrade::getOutput () [override], [virtual]
```

Retrieves the upgraded water supply system's output.

Returns the output of the level one upgraded water supply system.

Returns

The updated output as an integer.

Implements WaterSupplyUpgrade.

4.135.3.5 update()

```
void WaterSupplyLevelOneUpgrade::update () [override], [virtual]
```

Updates the state of the upgraded water supply system.

Implements specific behavior for the water supply system after applying the level one upgrade.

Implements WaterSupplyUpgrade.

4.135.3.6 upgrade()

```
Entity * WaterSupplyLevelOneUpgrade::upgrade ()  [override], [virtual]
```

Upgrades the current utility to the next level.

Returns

A pointer to the upgraded utility instance, or nullptr if already at maximum level.

Implements WaterSupplyUpgrade.

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

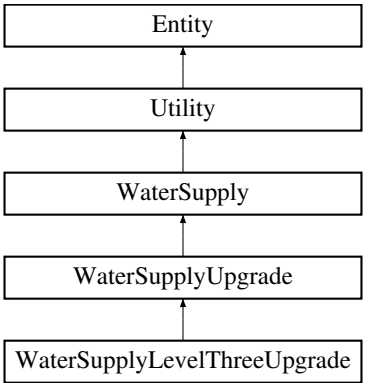
- src/entities/utility/watersupply/WaterSupplyLevelOneUpgrade.h
- src/entities/utility/watersupply/WaterSupplyLevelOneUpgrade.cpp

4.136 WaterSupplyLevelThreeUpgrade Class Reference

Represents the third level upgrade to a WaterSupply entity.

`#include <WaterSupplyLevelThreeUpgrade.h>`

Inheritance diagram for WaterSupplyLevelThreeUpgrade:



Public Member Functions

- WaterSupplyLevelThreeUpgrade (WaterSupply *water)
  Constructs a WaterSupplyLevelThreeUpgrade object.
- WaterSupplyLevelThreeUpgrade (WaterSupplyLevelThreeUpgrade *wSLTU)
  Copy constructor for WaterSupplyLevelThreeUpgrade.
- ~WaterSupplyLevelThreeUpgrade ()
  Destructor for WaterSupplyLevelThreeUpgrade.
- void update () override
  Updates the state of the upgraded water supply system.
- Entity * clone () override
  Clones the current WaterSupplyLevelThreeUpgrade object.
- Entity * upgrade () override
  Upgrades the current utility to the next level.
- int getOutput () override
  Retrieves the upgraded water supply system's output.
- Cost getCost () override
  Retrieves the cost of the utility or its upgraded version.
- int getLevel () override
  Gets the level of the water supply upgrade.

Public Member Functions inherited from WaterSupplyUpgrade

- WaterSupplyUpgrade (WaterSupply *water)

  Constructs a WaterSupplyUpgrade object based on an existing WaterSupply.
- WaterSupplyUpgrade (WaterSupplyUpgrade *wSU)

  Copy constructor for the WaterSupplyUpgrade class.
- virtual ~**WaterSupplyUpgrade** ()

  Destructor for the WaterSupplyUpgrade object.

Public Member Functions inherited from WaterSupply

- WaterSupply (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a WaterSupply object with specified attributes.
- WaterSupply (WaterSupply *water)

  Copy constructor for the WaterSupply class.
- virtual ~**WaterSupply** ()

  Destructor for the WaterSupply object.

Public Member Functions inherited from Utility

- Utility (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a Utility object with the specified parameters.
- Utility (Utility *utility)

  Copy constructor for the Utility class.
- virtual ~**Utility** ()

  Destructor for the Utility object.
- void setOutput (int output)

  Sets the output value of the utility.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.

- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity ∗entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity ∗entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity ∗ > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from WaterSupplyUpgrade

- WaterSupply ∗ **waterSupply**

  Pointer to the original WaterSupply that is being upgraded.

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

Global effect strength of the entity.

- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).
- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.136.1 Detailed Description

Represents the third level upgrade to a WaterSupply entity.

The WaterSupplyLevelThreeUpgrade class enhances the base functionality of a WaterSupply by increasing its delivery capacity. This class is the third upgrade level in a series of potential water supply improvements.

4.136.2 Constructor & Destructor Documentation

4.136.2.1 WaterSupplyLevelThreeUpgrade() [1/2]

```
WaterSupplyLevelThreeUpgrade::WaterSupplyLevelThreeUpgrade (
            WaterSupply * water)
```

Constructs a WaterSupplyLevelThreeUpgrade object.

Initializes the upgrade by enhancing the specified WaterSupply with a level three upgrade.

Parameters

water	Pointer to the original WaterSupply to be upgraded.

4.136.2.2 WaterSupplyLevelThreeUpgrade() [2/2]

```
WaterSupplyLevelThreeUpgrade::WaterSupplyLevelThreeUpgrade (
            WaterSupplyLevelThreeUpgrade * wSLTU)
```

Copy constructor for WaterSupplyLevelThreeUpgrade.

Creates a new WaterSupplyLevelThreeUpgrade object by copying the attributes of an existing WaterSupplyLevelThreeUpgrade object.

Parameters

wSLTU	Pointer to the existing WaterSupplyLevelThreeUpgrade to be copied.

4.136.2.3 ~WaterSupplyLevelThreeUpgrade()

```
WaterSupplyLevelThreeUpgrade::~WaterSupplyLevelThreeUpgrade ()
```

Destructor for WaterSupplyLevelThreeUpgrade.

Cleans up any resources associated with the upgrade.

4.136.3 Member Function Documentation

4.136.3.1 clone()

```
Entity * WaterSupplyLevelThreeUpgrade::clone ()  [override], [virtual]
```

Clones the current WaterSupplyLevelThreeUpgrade object.

Creates a new instance of WaterSupplyLevelThreeUpgrade with the same attributes as the current object.

Returns

A pointer to the newly cloned WaterSupplyLevelThreeUpgrade object.

Implements WaterSupplyUpgrade.

4.136.3.2 getCost()

```
Cost WaterSupplyLevelThreeUpgrade::getCost ()  [override], [virtual]
```

Retrieves the cost of the utility or its upgraded version.

Returns

A Cost object representing the monetary and material costs.

Implements WaterSupplyUpgrade.

4.136.3.3 getLevel()

```
int WaterSupplyLevelThreeUpgrade::getLevel ()  [override], [virtual]
```

Gets the level of the water supply upgrade.

Returns

The level of the water supply upgrade.

Reimplemented from Utility.

4.136.3.4 getOutput()

```
int WaterSupplyLevelThreeUpgrade::getOutput () [override], [virtual]
```

Retrieves the upgraded water supply system's output.

Returns the delivery capacity of the level three upgraded water supply system.

Returns

The updated delivery capacity as an integer.

Implements WaterSupplyUpgrade.

4.136.3.5 update()

```
void WaterSupplyLevelThreeUpgrade::update () [override], [virtual]
```

Updates the state of the upgraded water supply system.

Implements specific behavior for the water supply system after applying the level three upgrade.

Implements WaterSupplyUpgrade.

4.136.3.6 upgrade()

```
Entity * WaterSupplyLevelThreeUpgrade::upgrade () [override], [virtual]
```

Upgrades the current utility to the next level.

Returns

A pointer to the upgraded utility instance, or nullptr if already at maximum level.

Implements WaterSupplyUpgrade.

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

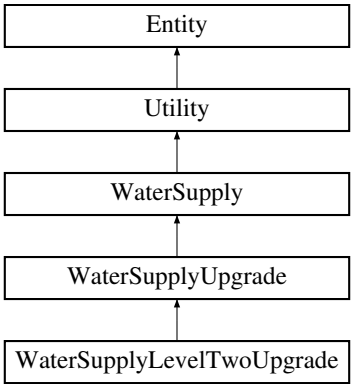
- src/entities/utility/watersupply/WaterSupplyLevelThreeUpgrade.h
- src/entities/utility/watersupply/WaterSupplyLevelThreeUpgrade.cpp

4.137 WaterSupplyLevelTwoUpgrade Class Reference

Represents the second level upgrade to a WaterSupply entity.

`#include <WaterSupplyLevelTwoUpgrade.h>`

Inheritance diagram for WaterSupplyLevelTwoUpgrade:



Public Member Functions

- WaterSupplyLevelTwoUpgrade (WaterSupply *water)
  Constructs a WaterSupplyLevelTwoUpgrade object.
- WaterSupplyLevelTwoUpgrade (WaterSupplyLevelTwoUpgrade *wSLTU)
  Copy constructor for WaterSupplyLevelTwoUpgrade.
- ~**WaterSupplyLevelTwoUpgrade** ()
  Destructor for WaterSupplyLevelTwoUpgrade.
- void update () override
  Updates the state of the upgraded water supply system.
- Entity * clone () override
  Clones the current WaterSupplyLevelTwoUpgrade object.
- Entity * upgrade () override
  Upgrades the current utility to the next level.
- int getOutput () override
  Retrieves the upgraded water supply system's output.
- Cost getCost () override
  Retrieves the cost of the utility or its upgraded version.
- int getLevel () override
  Gets the level of the water supply upgrade.

Public Member Functions inherited from WaterSupplyUpgrade

- WaterSupplyUpgrade (WaterSupply *water)
  Constructs a WaterSupplyUpgrade object based on an existing WaterSupply.
- WaterSupplyUpgrade (WaterSupplyUpgrade *wSU)
  Copy constructor for the WaterSupplyUpgrade class.
- virtual ~**WaterSupplyUpgrade** ()
  Destructor for the WaterSupplyUpgrade object.

Public Member Functions inherited from WaterSupply

- WaterSupply (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a WaterSupply object with specified attributes.
- WaterSupply (WaterSupply *water)

  Copy constructor for the WaterSupply class.
- virtual ~**WaterSupply** ()

  Destructor for the WaterSupply object.

Public Member Functions inherited from Utility

- Utility (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a Utility object with the specified parameters.
- Utility (Utility *utility)

  Copy constructor for the Utility class.
- virtual ~**Utility** ()

  Destructor for the Utility object.
- void setOutput (int output)

  Sets the output value of the utility.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

Sets the symbol of the entity.

- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from WaterSupplyUpgrade

- WaterSupply * **waterSupply**

  Pointer to the original WaterSupply that is being upgraded.

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).

- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.137.1 Detailed Description

Represents the second level upgrade to a WaterSupply entity.

The WaterSupplyLevelTwoUpgrade class enhances the base functionality of a WaterSupply system by increasing its output. This class represents the second upgrade level in the series.

4.137.2 Constructor & Destructor Documentation

4.137.2.1 WaterSupplyLevelTwoUpgrade() [1/2]

```
WaterSupplyLevelTwoUpgrade::WaterSupplyLevelTwoUpgrade (
            WaterSupply * water)
```

Constructs a WaterSupplyLevelTwoUpgrade object.

Enhances the specified WaterSupply system with a level two upgrade.

Parameters

water	Pointer to the original WaterSupply to be upgraded.

4.137.2.2 WaterSupplyLevelTwoUpgrade() [2/2]

```
WaterSupplyLevelTwoUpgrade::WaterSupplyLevelTwoUpgrade (
            WaterSupplyLevelTwoUpgrade * wSLTU)
```

Copy constructor for WaterSupplyLevelTwoUpgrade.

Copies the attributes of an existing WaterSupplyLevelTwoUpgrade object.

Parameters

wSLTU	Pointer to the existing object to be copied.

4.137.3 Member Function Documentation

4.137.3.1 clone()

```
Entity * WaterSupplyLevelTwoUpgrade::clone ()  [override], [virtual]
```

Clones the current WaterSupplyLevelTwoUpgrade object.

Returns

A pointer to the newly cloned object.

Implements WaterSupplyUpgrade.

4.137.3.2 getCost()

```
Cost WaterSupplyLevelTwoUpgrade::getCost ()  [override], [virtual]
```

Retrieves the cost of the utility or its upgraded version.

Returns

A Cost object representing the monetary and material costs.

Implements WaterSupplyUpgrade.

4.137.3.3 getLevel()

```
int WaterSupplyLevelTwoUpgrade::getLevel ()  [override], [virtual]
```

Gets the level of the water supply upgrade.

Returns

The level of the water supply upgrade.

Reimplemented from Utility.

4.137.3.4 getOutput()

```
int WaterSupplyLevelTwoUpgrade::getOutput () [override], [virtual]
```

Retrieves the upgraded water supply system's output.

Returns the output of the level one upgraded water supply system.

Returns

The updated output as an integer.

Implements WaterSupplyUpgrade.

4.137.3.5 update()

```
void WaterSupplyLevelTwoUpgrade::update () [override], [virtual]
```

Updates the state of the upgraded water supply system.

Implements WaterSupplyUpgrade.

4.137.3.6 upgrade()

```
Entity * WaterSupplyLevelTwoUpgrade::upgrade () [override], [virtual]
```

Upgrades the current utility to the next level.

Returns

A pointer to the upgraded utility instance, or nullptr if already at maximum level.

Implements WaterSupplyUpgrade.

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

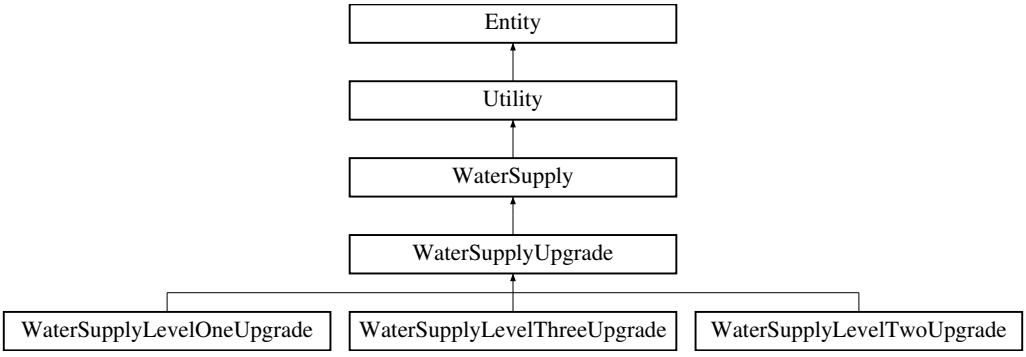
- src/entities/utility/watersupply/WaterSupplyLevelTwoUpgrade.h
- src/entities/utility/watersupply/WaterSupplyLevelTwoUpgrade.cpp

4.138 WaterSupplyUpgrade Class Reference

Represents an upgrade to a WaterSupply entity in the city builder simulation.

`#include <WaterSupplyUpgrade.h>`

Inheritance diagram for WaterSupplyUpgrade:



Public Member Functions

- WaterSupplyUpgrade (WaterSupply *water)

  Constructs a WaterSupplyUpgrade object based on an existing WaterSupply.
- WaterSupplyUpgrade (WaterSupplyUpgrade *wSU)

  Copy constructor for the WaterSupplyUpgrade class.
- virtual ~**WaterSupplyUpgrade** ()

  Destructor for the WaterSupplyUpgrade object.
- virtual void update ()=0

  Pure virtual function to update the upgraded water supply system.
- virtual Entity * clone ()=0

  Pure virtual function to clone the upgraded water supply system.
- virtual Entity * upgrade ()=0

  Upgrades the current utility to the next level.
- virtual int getOutput ()=0

  Retrieves the output of the upgraded water supply system.
- virtual Cost getCost ()=0

  Retrieves the cost of the utility or its upgraded version.

Public Member Functions inherited from WaterSupply

- WaterSupply (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a WaterSupply object with specified attributes.
- WaterSupply (WaterSupply *water)

  Copy constructor for the WaterSupply class.
- virtual ~**WaterSupply** ()

  Destructor for the WaterSupply object.

Public Member Functions inherited from Utility

- Utility (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a Utility object with the specified parameters.
- Utility (Utility *utility)

  Copy constructor for the Utility class.
- virtual ~**Utility** ()

  Destructor for the Utility object.
- void setOutput (int output)

  Sets the output value of the utility.
- virtual int getLevel ()

  Gets the level of the utility.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Protected Attributes

- WaterSupply * **waterSupply**

  Pointer to the original WaterSupply that is being upgraded.

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).
- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.138.1 Detailed Description

Represents an upgrade to a WaterSupply entity in the city builder simulation.

The WaterSupplyUpgrade class extends the functionality of a WaterSupply, enhancing its capabilities and acting as a wrapper around the existing WaterSupply object.

4.138.2 Constructor & Destructor Documentation

4.138.2.1 WaterSupplyUpgrade() [1/2]

```
WaterSupplyUpgrade::WaterSupplyUpgrade (
            WaterSupply * water)
```

Constructs a WaterSupplyUpgrade object based on an existing WaterSupply.

Initializes the upgrade with a reference to an existing WaterSupply, enhancing its features.

Parameters

water	Pointer to the WaterSupply being upgraded.

4.138.2.2 WaterSupplyUpgrade() [2/2]

```
WaterSupplyUpgrade::WaterSupplyUpgrade (
            WaterSupplyUpgrade * wSU)
```

Copy constructor for the WaterSupplyUpgrade class.

Creates a new WaterSupplyUpgrade object by copying the attributes of an existing WaterSupplyUpgrade.

Parameters

wSU	Pointer to the existing WaterSupplyUpgrade object to be copied.

4.138.3 Member Function Documentation

4.138.3.1 clone()

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

Pure virtual function to clone the upgraded water supply system.

Returns

A pointer to a new cloned WaterSupplyUpgrade object.

Reimplemented from WaterSupply.

Implemented in WaterSupplyLevelOneUpgrade, WaterSupplyLevelThreeUpgrade, and WaterSupplyLevelTwoUpgrade.

4.138.3.2 getCost()

```
virtual Cost WaterSupplyUpgrade::getCost ()  [pure virtual]
```

Retrieves the cost of the utility or its upgraded version.

Returns

A Cost object representing the monetary and material costs.

Reimplemented from Utility.

Implemented in WaterSupplyLevelOneUpgrade, WaterSupplyLevelThreeUpgrade, and WaterSupplyLevelTwoUpgrade.

4.138.3.3 getOutput()

```
virtual int WaterSupplyUpgrade::getOutput ()  [pure virtual]
```

Retrieves the output of the upgraded water supply system.

Returns

The output value as an integer.

Reimplemented from Utility.

Implemented in WaterSupplyLevelOneUpgrade, WaterSupplyLevelThreeUpgrade, and WaterSupplyLevelTwoUpgrade.

4.138.3.4 update()

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

Pure virtual function to update the upgraded water supply system.

Reimplemented from WaterSupply.

Implemented in WaterSupplyLevelOneUpgrade, WaterSupplyLevelThreeUpgrade, and WaterSupplyLevelTwoUpgrade.

4.138.3.5 upgrade()

```
virtual Entity * WaterSupplyUpgrade::upgrade ()  [pure virtual]
```

Upgrades the current utility to the next level.

Returns

A pointer to the upgraded utility instance, or nullptr if already at maximum level.

Reimplemented from WaterSupply.

Implemented in WaterSupplyLevelOneUpgrade, WaterSupplyLevelThreeUpgrade, and WaterSupplyLevelTwoUpgrade.

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

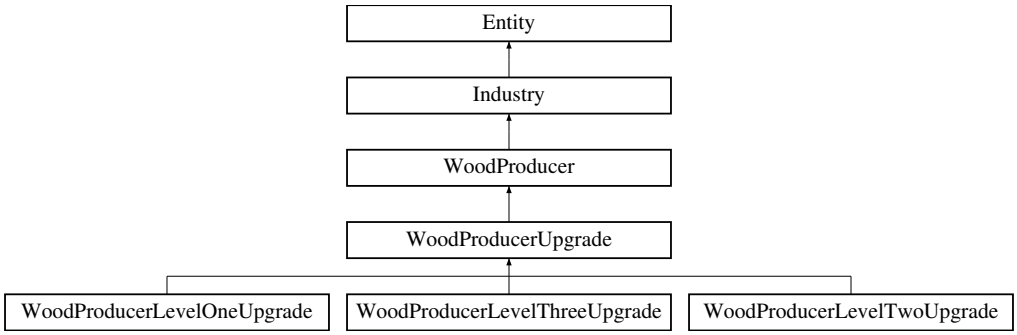
- src/entities/utility/watersupply/WaterSupplyUpgrade.h
- src/entities/utility/watersupply/WaterSupplyUpgrade.cpp

4.139 WoodProducer Class Reference

Represents a wood producer in the game.

`#include <WoodProducer.h>`

Inheritance diagram for WoodProducer:



Public Member Functions

- WoodProducer (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a WoodProducer with the given configuration.
- WoodProducer (WoodProducer *woodProducer)

  Copy constructor for the WoodProducer class.
- virtual ~**WoodProducer** ()

  Destructor for the WoodProducer class.
- void update () override

  Updates the state of the wood producer and notifies observers.
- Entity * clone () override

  Clones the current WoodProducer instance.
- Entity * upgrade () override

  Upgrades the wood producer to the next level.

Public Member Functions inherited from Industry

- Industry (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Industry entity with specified attributes.
- Industry (Industry *industry)

  Copy constructor for the Industry class.
- virtual ~**Industry** ()

  Virtual destructor for the Industry class.
- virtual int getOutput ()

  Gets the production output of the industry.
- void setOutput (int output)

  Sets the production output of the industry.
- virtual int getLevel ()

  Gets the current level of the industry.
- virtual Cost getCost ()

  Gets the cost of an upgrade.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).
- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.139.1 Detailed Description

Represents a wood producer in the game.

The WoodProducer class extends the Industry class to provide functionality for producing wood, which can be upgraded and observed by other entities.

4.139.2 Constructor & Destructor Documentation

4.139.2.1 WoodProducer() [1/2]

```
WoodProducer::WoodProducer (
            EntityConfig ec,
            Size size,
            int xPos,
            int yPos)
```

Constructs a WoodProducer with the given configuration.

Parameters

ec	The configuration for the entity.
size	The size of the wood producer.
xPos	The x position of the wood producer.
yPos	The y position of the wood producer.

4.139.2.2 WoodProducer() [2/2]

```
WoodProducer::WoodProducer (
            WoodProducer * woodProducer)
```

Copy constructor for the WoodProducer class.

Parameters

woodProducer	Pointer to the WoodProducer to copy.

4.139.3 Member Function Documentation

4.139.3.1 clone()

```
Entity * WoodProducer::clone ()  [override], [virtual]
```

Clones the current WoodProducer instance.

Returns

A pointer to a new WoodProducer that is a copy of this instance.

Implements Industry.

Reimplemented in WoodProducerLevelOneUpgrade, WoodProducerLevelThreeUpgrade, WoodProducerLevelTwoUpgrade, and WoodProducerUpgrade.

4.139.3.2 update()

```
void WoodProducer::update ()  [override], [virtual]
```

Updates the state of the wood producer and notifies observers.

Implements Industry.

Reimplemented in WoodProducerLevelOneUpgrade, WoodProducerLevelThreeUpgrade, WoodProducerLevelTwoUpgrade, and WoodProducerUpgrade.

4.139.3.3 upgrade()

```
Entity * WoodProducer::upgrade ()  [override], [virtual]
```

Upgrades the wood producer to the next level.

Returns

A pointer to the upgraded entity, which is a WoodProducerLevelOneUpgrade.

Implements Industry.

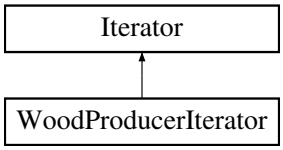
Reimplemented in WoodProducerLevelOneUpgrade, WoodProducerLevelThreeUpgrade, WoodProducerLevelTwoUpgrade, and WoodProducerUpgrade.

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

- src/entities/industry/woodproducer/WoodProducer.h
- src/entities/industry/woodproducer/WoodProducer.cpp

4.140 WoodProducerIterator Class Reference

Inheritance diagram for WoodProducerIterator:



Public Member Functions

- **WoodProducerIterator** ()

  Construct a new Wood Producer Iterator:: Wood Producer Iterator object.
- ~**WoodProducerIterator** ()

  Destroy the Wood Producer Iterator:: Wood Producer Iterator object.
- WoodProducerIterator (std::vector< std::vector< Entity * > > &grid)

  Construct a new Wood Producer Iterator:: Wood Producer Iterator object.
- void first ()

  Sets the iterator to the first unvisited WoodProducer.
- void next ()

  Advances to the next unvisited WoodProducer.
- bool hasNext ()

  Checks if there is another unvisited WoodProducer.
- Entity * current ()

  Returns the current WoodProducer.

Public Member Functions inherited from Iterator

- **Iterator** ()

  Construct a new Iterator object, initializing row and column to zero.
- virtual ~**Iterator** ()

  Destroy the Iterator object.
- **Iterator** (std::vector< std::vector< Entity * > > &grid)
- virtual int getRow ()

  Get the current row index of the iterator.
- virtual int getCol ()

  Get the current column index of the iterator.

Additional Inherited Members

Protected Member Functions inherited from Iterator

- bool isVisited (Entity *entity)

  Check if the specified entity has been visited.
- void markVisited (Entity *entity)

  Mark the specified entity as visited.

Protected Attributes inherited from Iterator

- std::vector< std::vector< Entity * > > **grid**
- std::vector< std::vector< Entity * > >::iterator **currRow**
- std::vector< Entity * >::iterator **curr**
- int **row**
- int **col**
- std::unordered_set< Entity * > **visitedEntities**

4.140.1 Constructor & Destructor Documentation

4.140.1.1 WoodProducerIterator()

```
WoodProducerIterator::WoodProducerIterator (
            std::vector< std::vector< Entity * > > & grid)
```

Construct a new Wood Producer Iterator:: Wood Producer Iterator object.

Parameters

grid	

4.140.2 Member Function Documentation

4.140.2.1 current()

```
Entity * WoodProducerIterator::current ()  [virtual]
```

Returns the current WoodProducer.

Returns

Entity*

Implements Iterator.

4.140.2.2 first()

```
void WoodProducerIterator::first ()  [virtual]
```

Sets the iterator to the first unvisited WoodProducer.

Implements Iterator.

4.140.2.3 hasNext()

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

Checks if there is another unvisited WoodProducer.

Returns

true if there is another unvisited WoodProducer, false otherwise

Implements Iterator.

4.140.2.4 next()

```
void WoodProducerIterator::next ()  [virtual]
```

Advances to the next unvisited WoodProducer.

Implements Iterator.

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

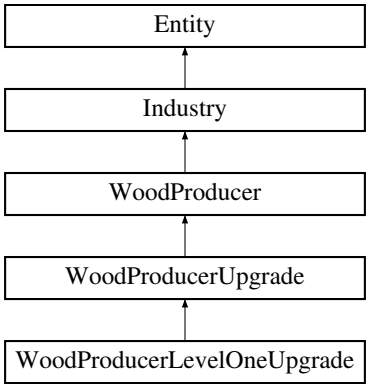
- src/iterators/industry/WoodProducerIterator.h
- src/iterators/industry/WoodProducerIterator.cpp

4.141 WoodProducerLevelOneUpgrade Class Reference

Class representing the first upgrade level for a wood producer.

`#include <WoodProducerLevelOneUpgrade.h>`

Inheritance diagram for WoodProducerLevelOneUpgrade:



Public Member Functions

- WoodProducerLevelOneUpgrade (WoodProducer *woodProducer)

  Constructs a WoodProducerLevelOneUpgrade with the specified wood producer.
- WoodProducerLevelOneUpgrade (WoodProducerLevelOneUpgrade *woodprod)

  Copy constructor for WoodProducerLevelOneUpgrade.
- ~**WoodProducerLevelOneUpgrade** ()

  Destructor for the WoodProducerLevelOneUpgrade class.
- void update () override

  Updates the state of the wood producer upgrade.
- Entity * clone () override

  Clones the current WoodProducerLevelOneUpgrade instance.
- int getOutput () override

  Gets the output of the wood producer upgrade.
- int getLevel () override

  Gets the level of the wood producer upgrade.
- Entity * upgrade () override

  Upgrades the wood producer to the next level.
- Cost getCost () override

  Gets the cost of the wood producer upgrade.

Public Member Functions inherited from WoodProducerUpgrade

- WoodProducerUpgrade (WoodProducer *woodProducer)

  Constructs a WoodProducerUpgrade with the specified wood producer.
- virtual ~**WoodProducerUpgrade** ()

  Destructor for the WoodProducerUpgrade class.
- WoodProducerUpgrade (WoodProducerUpgrade *woodProducerUpgrade)

  Copy constructor for the WoodProducerUpgrade class.

Public Member Functions inherited from WoodProducer

- WoodProducer (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a WoodProducer with the given configuration.
- WoodProducer (WoodProducer *woodProducer)

  Copy constructor for the WoodProducer class.
- virtual ~**WoodProducer** ()

  Destructor for the WoodProducer class.

Public Member Functions inherited from Industry

- Industry (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Industry entity with specified attributes.
- Industry (Industry *industry)

  Copy constructor for the Industry class.
- virtual ~**Industry** ()

  Virtual destructor for the Industry class.
- void setOutput (int output)

  Sets the production output of the industry.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

Sets the symbol of the entity.

- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from WoodProducerUpgrade

- WoodProducer * woodProducer

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).
- Size **size**

Size object representing the entity's dimensions.

- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.141.1 Detailed Description

Class representing the first upgrade level for a wood producer.

The WoodProducerLevelOneUpgrade class is a concrete implementation of WoodProducerUpgrade, providing functionality for the first level of upgrade for wood producers.

4.141.2 Constructor & Destructor Documentation

4.141.2.1 WoodProducerLevelOneUpgrade() [1/2]

```
WoodProducerLevelOneUpgrade::WoodProducerLevelOneUpgrade (
            WoodProducer * woodProducer)
```

Constructs a WoodProducerLevelOneUpgrade with the specified wood producer.

Parameters

woodProducer	Pointer to the wood producer to upgrade.

4.141.2.2 WoodProducerLevelOneUpgrade() [2/2]

```
WoodProducerLevelOneUpgrade::WoodProducerLevelOneUpgrade (
            WoodProducerLevelOneUpgrade * woodprod)
```

Copy constructor for WoodProducerLevelOneUpgrade.

Parameters

woodprod	Pointer to the WoodProducerLevelOneUpgrade to copy.

4.141.3 Member Function Documentation

4.141.3.1 clone()

```
Entity * WoodProducerLevelOneUpgrade::clone ()  [override], [virtual]
```

Clones the current WoodProducerLevelOneUpgrade instance.

Returns

A pointer to a new WoodProducerLevelOneUpgrade that is a copy of this instance.

Implements WoodProducerUpgrade.

4.141.3.2 getCost()

```
Cost WoodProducerLevelOneUpgrade::getCost ()  [override], [virtual]
```

Gets the cost of the wood producer upgrade.

Returns

The cost associated with the wood producer upgrade.

Implements WoodProducerUpgrade.

4.141.3.3 getLevel()

```
int WoodProducerLevelOneUpgrade::getLevel ()  [override], [virtual]
```

Gets the level of the wood producer upgrade.

Returns

The level of the wood producer upgrade.

Reimplemented from Industry.

4.141.3.4 getOutput()

```
int WoodProducerLevelOneUpgrade::getOutput ()  [override], [virtual]
```

Gets the output of the wood producer upgrade.

Returns

The output produced by the wood producer upgrade.

Implements WoodProducerUpgrade.

4.141.3.5 update()

```
void WoodProducerLevelOneUpgrade::update ()  [override], [virtual]
```

Updates the state of the wood producer upgrade.

Implements WoodProducerUpgrade.

4.141.3.6 upgrade()

```
Entity * WoodProducerLevelOneUpgrade::upgrade ()  [override], [virtual]
```

Upgrades the wood producer to the next level.

Returns

A pointer to the upgraded wood producer.

Implements WoodProducerUpgrade.

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

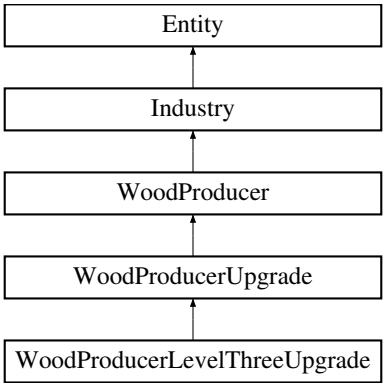
- src/entities/industry/woodproducer/WoodProducerLevelOneUpgrade.h
- src/entities/industry/woodproducer/WoodProducerLevelOneUpgrade.cpp

4.142 WoodProducerLevelThreeUpgrade Class Reference

Class representing the third upgrade level for a wood producer.

```
#include <WoodProducerLevelThreeUpgrade.h>
```

Inheritance diagram for WoodProducerLevelThreeUpgrade:



Public Member Functions

- WoodProducerLevelThreeUpgrade (WoodProducer *woodProd)

  Constructs a WoodProducerLevelThreeUpgrade with the specified wood producer.
- WoodProducerLevelThreeUpgrade (WoodProducerLevelThreeUpgrade *woodProd)

  Copy constructor for WoodProducerLevelThreeUpgrade.
- ~**WoodProducerLevelThreeUpgrade** ()

  Destructor for the WoodProducerLevelThreeUpgrade class.
- void update () override

  Updates the state of the wood producer upgrade.
- Entity * clone () override

  Clones the current WoodProducerLevelThreeUpgrade instance.
- int getOutput () override

  Gets the output of the wood producer upgrade.
- int getLevel () override

  Gets the level of the wood producer upgrade.
- Entity * upgrade () override

  Upgrades the wood producer to the next level (no further upgrade).
- Cost getCost () override

  Gets the cost of the wood producer upgrade.

Public Member Functions inherited from WoodProducerUpgrade

- WoodProducerUpgrade (WoodProducer *woodProducer)

  Constructs a WoodProducerUpgrade with the specified wood producer.
- virtual ~**WoodProducerUpgrade** ()

  Destructor for the WoodProducerUpgrade class.
- WoodProducerUpgrade (WoodProducerUpgrade *woodProducerUpgrade)

  Copy constructor for the WoodProducerUpgrade class.

Public Member Functions inherited from WoodProducer

- WoodProducer (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a WoodProducer with the given configuration.
- WoodProducer (WoodProducer *woodProducer)

  Copy constructor for the WoodProducer class.
- virtual ~**WoodProducer** ()

  Destructor for the WoodProducer class.

Public Member Functions inherited from Industry

- Industry (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Industry entity with specified attributes.
- Industry (Industry *industry)

  Copy constructor for the Industry class.
- virtual ~**Industry** ()

  Virtual destructor for the Industry class.
- void setOutput (int output)

  Sets the production output of the industry.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from WoodProducerUpgrade

- WoodProducer * woodProducer

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).
- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.142.1 Detailed Description

Class representing the third upgrade level for a wood producer.

The WoodProducerLevelThreeUpgrade class is a concrete implementation of WoodProducerUpgrade, providing functionality for the third level of upgrade for wood producers.

4.142.2 Constructor & Destructor Documentation

4.142.2.1 WoodProducerLevelThreeUpgrade() [1/2]

```
WoodProducerLevelThreeUpgrade::WoodProducerLevelThreeUpgrade (
            WoodProducer * woodProd)
```

Constructs a WoodProducerLevelThreeUpgrade with the specified wood producer.

Parameters

woodProd	Pointer to the wood producer to upgrade.

4.142.2.2 WoodProducerLevelThreeUpgrade() [2/2]

```
WoodProducerLevelThreeUpgrade::WoodProducerLevelThreeUpgrade (
            WoodProducerLevelThreeUpgrade * woodProd)
```

Copy constructor for WoodProducerLevelThreeUpgrade.

Parameters

woodProd	Pointer to the WoodProducerLevelThreeUpgrade to copy.

4.142.3 Member Function Documentation

4.142.3.1 clone()

```
Entity * WoodProducerLevelThreeUpgrade::clone ()  [override], [virtual]
```

Clones the current WoodProducerLevelThreeUpgrade instance.

Returns

A pointer to a new WoodProducerLevelThreeUpgrade that is a copy of this instance.

Implements WoodProducerUpgrade.

4.142.3.2 getCost()

```
Cost WoodProducerLevelThreeUpgrade::getCost ()  [override], [virtual]
```

Gets the cost of the wood producer upgrade.

Returns

The cost associated with the wood producer upgrade.

Implements WoodProducerUpgrade.

4.142.3.3 getLevel()

```
int WoodProducerLevelThreeUpgrade::getLevel ()  [override], [virtual]
```

Gets the level of the wood producer upgrade.

Returns

The level of the wood producer upgrade.

Reimplemented from Industry.

4.142.3.4 getOutput()

```
int WoodProducerLevelThreeUpgrade::getOutput () [override], [virtual]
```

Gets the output of the wood producer upgrade.

Returns

The output produced by the wood producer upgrade.

Implements WoodProducerUpgrade.

4.142.3.5 update()

```
void WoodProducerLevelThreeUpgrade::update () [override], [virtual]
```

Updates the state of the wood producer upgrade.

Implements WoodProducerUpgrade.

4.142.3.6 upgrade()

```
Entity * WoodProducerLevelThreeUpgrade::upgrade () [override], [virtual]
```

Upgrades the wood producer to the next level (no further upgrade).

Returns

nullptr as there are no further upgrades.

Implements WoodProducerUpgrade.

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

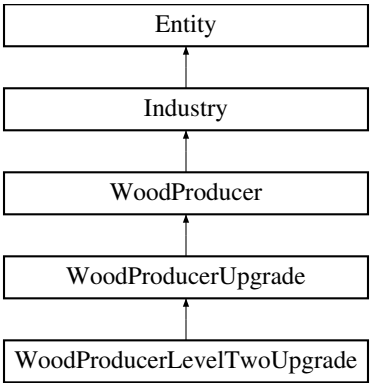
- src/entities/industry/woodproducer/WoodProducerLevelThreeUpgrade.h
- src/entities/industry/woodproducer/WoodProducerLevelThreeUpgrade.cpp

4.143 WoodProducerLevelTwoUpgrade Class Reference

Class representing the second upgrade level for a wood producer.

```
#include <WoodProducerLevelTwoUpgrade.h>
```

Inheritance diagram for WoodProducerLevelTwoUpgrade:



Public Member Functions

- WoodProducerLevelTwoUpgrade (WoodProducer *woodProducer)

  Constructs a WoodProducerLevelTwoUpgrade with the specified wood producer.
- WoodProducerLevelTwoUpgrade (WoodProducerLevelTwoUpgrade *woodProd)

  Copy constructor for WoodProducerLevelTwoUpgrade.
- ~**WoodProducerLevelTwoUpgrade** ()

  Destructor for the WoodProducerLevelTwoUpgrade class.
- void update () override

  Updates the state of the wood producer upgrade.
- Entity * clone () override

  Clones the current WoodProducerLevelTwoUpgrade instance.
- int getOutput () override

  Gets the output of the wood producer upgrade.
- int getLevel () override

  Gets the level of the wood producer upgrade.
- Entity * upgrade () override

  Upgrades the wood producer to the next level.
- Cost getCost () override

  Gets the cost of the wood producer upgrade.

Public Member Functions inherited from WoodProducerUpgrade

- WoodProducerUpgrade (WoodProducer *woodProducer)

  Constructs a WoodProducerUpgrade with the specified wood producer.
- virtual ~**WoodProducerUpgrade** ()

  Destructor for the WoodProducerUpgrade class.
- WoodProducerUpgrade (WoodProducerUpgrade *woodProducerUpgrade)

  Copy constructor for the WoodProducerUpgrade class.

Public Member Functions inherited from WoodProducer

- WoodProducer (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a WoodProducer with the given configuration.
- WoodProducer (WoodProducer *woodProducer)

  Copy constructor for the WoodProducer class.
- virtual ~**WoodProducer** ()

  Destructor for the WoodProducer class.

Public Member Functions inherited from Industry

- Industry (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Industry entity with specified attributes.
- Industry (Industry *industry)

  Copy constructor for the Industry class.
- virtual ~**Industry** ()

  Virtual destructor for the Industry class.
- void setOutput (int output)

  Sets the production output of the industry.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Additional Inherited Members

Protected Attributes inherited from WoodProducerUpgrade

- WoodProducer * woodProducer

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).
- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.143.1 Detailed Description

Class representing the second upgrade level for a wood producer.

The WoodProducerLevelTwoUpgrade class is a concrete implementation of WoodProducerUpgrade, providing functionality for the second level of upgrade for wood producers.

4.143.2 Constructor & Destructor Documentation

4.143.2.1 WoodProducerLevelTwoUpgrade() [1/2]

```
WoodProducerLevelTwoUpgrade::WoodProducerLevelTwoUpgrade (
            WoodProducer * woodProducer)
```

Constructs a WoodProducerLevelTwoUpgrade with the specified wood producer.

Parameters

woodProducer	Pointer to the wood producer to upgrade.

4.143.2.2 WoodProducerLevelTwoUpgrade() [2/2]

```
WoodProducerLevelTwoUpgrade::WoodProducerLevelTwoUpgrade (
            WoodProducerLevelTwoUpgrade * woodProd)
```

Copy constructor for WoodProducerLevelTwoUpgrade.

Parameters

woodProd	Pointer to the WoodProducerLevelTwoUpgrade to copy.

4.143.3 Member Function Documentation

4.143.3.1 clone()

```
Entity * WoodProducerLevelTwoUpgrade::clone ()  [override], [virtual]
```

Clones the current WoodProducerLevelTwoUpgrade instance.

Returns

A pointer to a new WoodProducerLevelTwoUpgrade that is a copy of this instance.

Implements WoodProducerUpgrade.

4.143.3.2 getCost()

```
Cost WoodProducerLevelTwoUpgrade::getCost ()  [override], [virtual]
```

Gets the cost of the wood producer upgrade.

Returns

The cost associated with the wood producer upgrade.

Implements WoodProducerUpgrade.

4.143.3.3 getLevel()

```
int WoodProducerLevelTwoUpgrade::getLevel ()  [override], [virtual]
```

Gets the level of the wood producer upgrade.

Returns

The level of the wood producer upgrade.

Reimplemented from Industry.

4.143.3.4 getOutput()

```
int WoodProducerLevelTwoUpgrade::getOutput ()  [override], [virtual]
```

Gets the output of the wood producer upgrade.

Returns

The output produced by the wood producer upgrade.

Implements WoodProducerUpgrade.

4.143.3.5 update()

```
void WoodProducerLevelTwoUpgrade::update ()  [override], [virtual]
```

Updates the state of the wood producer upgrade.

Implements WoodProducerUpgrade.

4.143.3.6 upgrade()

```
Entity * WoodProducerLevelTwoUpgrade::upgrade ()  [override], [virtual]
```

Upgrades the wood producer to the next level.

Returns

A pointer to the upgraded wood producer.

Implements WoodProducerUpgrade.

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

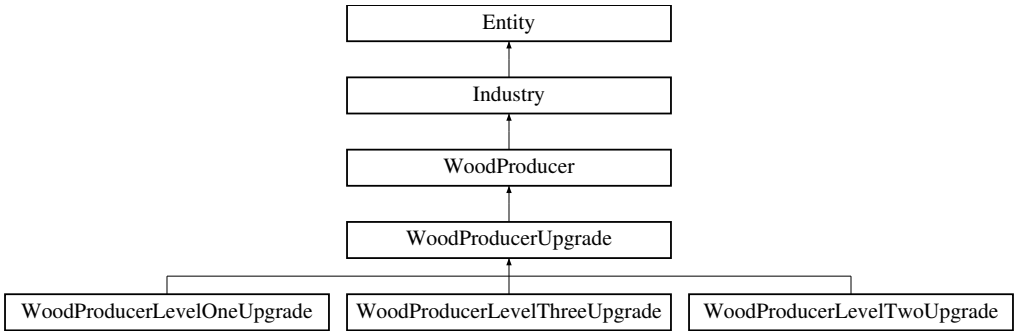
- src/entities/industry/woodproducer/WoodProducerLevelTwoUpgrade.h
- src/entities/industry/woodproducer/WoodProducerLevelTwoUpgrade.cpp

4.144 WoodProducerUpgrade Class Reference

Abstract base class for wood producer upgrades.

`#include <WoodProducerUpgrade.h>`

Inheritance diagram for WoodProducerUpgrade:



Public Member Functions

- WoodProducerUpgrade (WoodProducer *woodProducer)

  Constructs a WoodProducerUpgrade with the specified wood producer.
- virtual ~**WoodProducerUpgrade** ()

  Destructor for the WoodProducerUpgrade class.
- WoodProducerUpgrade (WoodProducerUpgrade *woodProducerUpgrade)

  Copy constructor for the WoodProducerUpgrade class.
- virtual void update ()=0

  Updates the state of the wood producer upgrade.
- virtual int getOutput ()=0

  Gets the output of the wood producer upgrade.
- virtual Cost getCost ()=0

  Gets the cost of the wood producer upgrade.
- virtual Entity * upgrade ()=0

  Upgrades the wood producer to the next level.
- virtual Entity * clone ()=0

  Clones the current WoodProducerUpgrade instance.

Public Member Functions inherited from WoodProducer

- WoodProducer (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs a WoodProducer with the given configuration.
- WoodProducer (WoodProducer *woodProducer)

  Copy constructor for the WoodProducer class.
- virtual ~**WoodProducer** ()

  Destructor for the WoodProducer class.

Public Member Functions inherited from Industry

- Industry (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Industry entity with specified attributes.
- Industry (Industry *industry)

  Copy constructor for the Industry class.
- virtual ~**Industry** ()

  Virtual destructor for the Industry class.
- void setOutput (int output)

  Sets the production output of the industry.
- virtual int getLevel ()

  Gets the current level of the industry.

Public Member Functions inherited from Entity

- Entity (EntityConfig ec, Size size, int xPos, int yPos)

  Constructs an Entity with specified attributes.
- Entity (Entity *entity)

  Copy constructor for the Entity class.
- virtual ~**Entity** ()

  Virtual destructor for the Entity class.
- bool isWithinEffectRadius (Entity *entity)

  Checks if another entity is within the effect radius of this entity.
- int getXPosition ()

  Gets the X-coordinate position of the entity.
- int getYPosition ()

  Gets the Y-coordinate position of the entity.
- void setXPosition (int x)

  Sets the X-coordinate position of the entity.
- void setYPosition (int y)

  Sets the Y-coordinate position of the entity.
- int getRevenue ()

  Gets the revenue generated by the entity.
- int getWidth ()

  Gets the width of the entity.
- int getHeight ()

  Gets the height of the entity.
- bool isBuilt ()

  Checks if the entity is built (i.e., not under construction).
- void **updateBuildState** ()

  Updates the build state of the entity.
- void setSymbol (std::string symbol)

  Sets the symbol of the entity.
- void **subscribeToAllResidentialInRadius** ()

  Subscribes the entity to all residential entities within its effect radius.
- void subscribe (Entity *entity)

  Subscribes this entity as an observer of another entity.
- void unsubscribe (Entity *entity)

  Unsubscribes this entity from observing another entity.
- void **unsubscribeFromAllBuildings** ()

  Unsubscribes this entity from all buildings it is observing.
- void **residentialBuildingPlaced** ()

  Called when a new residential building is placed, triggering updates.
- const std::vector< Entity * > getObservers ()

  Gets the list of entities observing this entity.
- EntityType getType () const

  Gets the entity type of this entity.
- Size getSize () const

  Gets the size of this entity.
- std::string getSymbol ()

  Gets the symbol of the entity.
- float getElectricityConsumption ()

  Gets the electricity consumption of the entity.
- float getWaterConsumption ()

  Gets the water consumption of the entity.

Protected Attributes

- WoodProducer * woodProducer

Protected Attributes inherited from Entity

- std::string **symbol**

  Symbol representing the entity.
- int **effectRadius**

  Radius of effect for this entity.
- int **localEffectStrength**

  Local effect strength of the entity.
- int **globalEffectStrength**

  Global effect strength of the entity.
- int **width**

  Width of the entity.
- int **height**

  Height of the entity.
- int **xPosition**

  X-coordinate of the entity's position (bottom left corner).
- int **yPosition**

  Y-coordinate of the entity's position (bottom left corner).
- Size **size**

  Size object representing the entity's dimensions.
- EntityType **type**

  The type of entity.
- State * **state**

  Pointer to the current state of the entity.
- int **revenue**

  Revenue generated by the entity.
- float **electricityConsumption**

  Electricity consumption of the entity.
- float **waterConsumption**

  Water consumption of the entity.
- std::vector< Entity * > **observers**

  List of other entities observing this entity.

4.144.1 Detailed Description

Abstract base class for wood producer upgrades.

The WoodProducerUpgrade class extends the WoodProducer class, providing a framework for creating various upgrades to wood producers.

4.144.2 Constructor & Destructor Documentation

4.144.2.1 WoodProducerUpgrade() [1/2]

```
WoodProducerUpgrade::WoodProducerUpgrade (
            WoodProducer * woodProducer)
```

Constructs a WoodProducerUpgrade with the specified wood producer.

Parameters

woodProducer	Pointer to the wood producer to upgrade.

4.144.2.2 WoodProducerUpgrade() [2/2]

```
WoodProducerUpgrade::WoodProducerUpgrade (
            WoodProducerUpgrade * woodProducerUpgrade)
```

Copy constructor for the WoodProducerUpgrade class.

Parameters

woodProducerUpgrade	Pointer to the WoodProducerUpgrade to copy.

4.144.3 Member Function Documentation

4.144.3.1 clone()

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

Clones the current WoodProducerUpgrade instance.

This function must be implemented in derived classes.

Returns

A pointer to a new WoodProducerUpgrade that is a copy of this instance.

Reimplemented from WoodProducer.

Implemented in WoodProducerLevelOneUpgrade, WoodProducerLevelThreeUpgrade, and WoodProducerLevelTwoUpgrade.

4.144.3.2 getCost()

```
virtual Cost WoodProducerUpgrade::getCost ()  [pure virtual]
```

Gets the cost of the wood producer upgrade.

This function must be implemented in derived classes.

Returns

The cost associated with the wood producer upgrade.

Reimplemented from Industry.

Implemented in WoodProducerLevelOneUpgrade, WoodProducerLevelThreeUpgrade, and WoodProducerLevelTwoUpgrade.

4.144.3.3 getOutput()

```
virtual int WoodProducerUpgrade::getOutput ()  [pure virtual]
```

Gets the output of the wood producer upgrade.

This function must be implemented in derived classes.

Returns

The output produced by the wood producer upgrade.

Reimplemented from Industry.

Implemented in WoodProducerLevelOneUpgrade, WoodProducerLevelThreeUpgrade, and WoodProducerLevelTwoUpgrade.

4.144.3.4 update()

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

Updates the state of the wood producer upgrade.

This function must be implemented in derived classes.

Reimplemented from WoodProducer.

Implemented in WoodProducerLevelOneUpgrade, WoodProducerLevelThreeUpgrade, and WoodProducerLevelTwoUpgrade.

4.144.3.5 upgrade()

```
virtual Entity * WoodProducerUpgrade::upgrade ()  [pure virtual]
```

Upgrades the wood producer to the next level.

This function must be implemented in derived classes.

Returns

A pointer to the upgraded wood producer.

Reimplemented from WoodProducer.

Implemented in WoodProducerLevelOneUpgrade, WoodProducerLevelThreeUpgrade, and WoodProducerLevelTwoUpgrade.

4.144.4 Member Data Documentation

4.144.4.1 woodProducer

```
WoodProducer* WoodProducerUpgrade::woodProducer  [protected]
```

Pointer to the associated wood producer.

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

- src/entities/industry/woodproducer/WoodProducerUpgrade.h
- src/entities/industry/woodproducer/WoodProducerUpgrade.cpp

Chapter 5

File Documentation

5.1 src/city/City.h File Reference

Manages city entities and resources in the simulation.

```
#include "visitors/base/CityVisitor.h"
#include <vector>
#include <cstdlib>
#include <ctime>
#include "policies/water/WaterPolicy.h"
#include "policies/electricity/ElectricityPolicy.h"
#include "utils/PolicyType.h"
```

Classes

- class City

  Singleton class that represents and manages a simulated city with entities, resources, and policies.

5.1.1 Detailed Description

Manages city entities and resources in the simulation.

5.2 City.h

Go to the documentation of this file.

```
00001
00006 #ifndef CITY_H
00007 #define CITY_H
00008
00009 #include "visitors/base/CityVisitor.h"
00010 #include <vector>
00011 #include <cstdlib> // For rand and srand
00012 #include <ctime>   // For time
00013 #include "policies/water/WaterPolicy.h"
00014 #include "policies/electricity/ElectricityPolicy.h"
00015 #include "utils/PolicyType.h"
00016
00017 class Entity;
```

```
00018 class Iterator;
00019
00024 class City
00025 {
00026 private:
00027     std::vector<std::vector<Entity *» grid;
00028     int width;
00029     int height;
00030     float satisfaction;
00031     int money;
00032     int wood;
00033     int stone;
00034     int concrete;
00035     int populationCapacity;
00036     int population;
00037     int electricityProduction;
00038     int electricityConsumption;
00039     int waterProduction;
00040     int waterConsumption;
00041     int wasteProduction;
00042     int wasteConsumption;
00043     int sewageProduction;
00044     int sewageConsumption;
00045     int residentialTax;
00046     int economicTax;
00047
00048     City();
00049     ~City();
00050
00051     WaterPolicy *waterPolicy = nullptr;
00052     ElectricityPolicy *electricityPolicy = nullptr;
00053
00054 public:
00059     static City *instance();
00060
00061     City(const City &) = delete;
00062     City &operator=(const City &) = delete;
00063
00070     Entity *getEntity(int x, int y);
00071
00076     void addEntity(Entity *entity);
00077
00082     std::vector<std::vector<Entity *» &getGrid();
00083
00089     void deleteEntity(int x, int y);
00090
00095     void accept(CityVisitor &visitor);
00096
00097     // Getters for city properties
00098     int getWidth() const;
00099     int getHeight() const;
00100     float getSatisfaction() const;
00101     int getMoney() const;
00102     int getWood() const;
00103     int getStone() const;
00104     int getConcrete() const;
00105     int getPopulationCapacity() const;
00106     int getPopulation() const;
00107     int getElectricityProduction() const;
00108     int getElectricityConsumption() const;
00109     int getWaterProduction() const;
00110     int getWaterConsumption() const;
00111     int getWasteProduction() const;
00112     int getWasteConsumption() const;
00113     int getSewageProduction() const;
00114     int getSewageConsumption() const;
00115     int getResidentialTax() const;
00116     int getEconomicTax() const;
00117
00122     WaterPolicy *getWaterPolicy() const;
00123
00128     ElectricityPolicy *getElectricityPolicy() const;
00129
00130     // Setters for city properties
00131     void setWidth(int width);
00132     void setHeight(int height);
00133     void setSatisfaction(float satisfaction);
00134     void setMoney(int money);
00135     void setWood(int wood);
00136     void setStone(int stone);
00137     void setConcrete(int concrete);
00138     void setPopulationCapacity(int populationCapacity);
00139     void setPopulation(int population);
00140     void setElectricityProduction(int electricityProduction);
00141     void setElectricityConsumption(int electricityConsumption);
00142     void setWaterProduction(int waterProduction);
00143     void setWaterConsumption(int waterConsumption);
```

```
00144     void setWasteProduction(int wasteProduction);
00145     void setWasteConsumption(int wasteConsumption);
00146     void setSewageProduction(int sewageProduction);
00147     void setSewageConsumption(int sewageConsumption);
00148     void setResidentialTax(int residentialTax);
00149     void setEconomicTax(int economicTax);
00150
00155     void setWaterPolicy(PolicyType policyType);
00156
00161     void setElectricityPolicy(PolicyType policyType);
00162
00168     void reset(int width, int height);
00169     void reset();
00170
00176     Iterator *createCityIterator(bool unique);
00177     Iterator *createBuildingIterator(bool unique);
00178     Iterator *createUtilityIterator(bool unique);
00179     Iterator *createIndustryIterator(bool unique);
00180     Iterator *createRoadIterator(bool unique);
00181     Iterator *createTransportIterator(bool unique);
00182     Iterator *createEconomicBuildingIterator(bool unique);
00183     Iterator *createResidentialBuildingIterator(bool unique);
00184     Iterator *createServiceBuildingIterator(bool unique);
00185     Iterator *createAmenityIterator(bool unique);
00186     Iterator *createPowerPlantIterator(bool unique);
00187     Iterator *createWaterSupplyIterator(bool unique);
00188     Iterator *createWasteManagementIterator(bool unique);
00189     Iterator *createSewageSystemIterator(bool unique);
00190     Iterator *createConcreteProducerIterator(bool unique);
00191     Iterator *createStoneProducerIterator(bool unique);
00192     Iterator *createWoodProducerIterator(bool unique);
00193
00197     void createRandomRoad();
00198
00202     void displayCity() const;
00203 };
00204
00205 #endif // CITY_H
```

5.3 src/city/CivZero.h File Reference

The main game engine file for CivZero.

```
#include "city/City.h"
#include <optional>
```

Classes

- class CivZero

  The main game engine class for CivZero.

5.3.1 Detailed Description

The main game engine file for CivZero.

5.4 CivZero.h

Go to the documentation of this file.

```
00001
00006 #ifndef CIVZERO_H
00007 #define CIVZERO_H
00008
00009 #include "city/City.h"
00010 #include <optional> // For std::optional
00011
00018 class CivZero
00019 {
00020 public:
00025     static CivZero &instance();
00026
00027     // Prevent copying and assignment
00028     CivZero(const CivZero &) = delete;
00029     CivZero &operator=(const CivZero &) = delete;
00030
00037     void startGame(bool generateRandomCity = false, std::optional<unsigned int> seed = std::nullopt);
00038
00042     void quit();
00043
00047     void incrementGameLoop();
00048
00054     int getGameLoop();
00055
00056 private:
00057     static int const GRID_SIZE = 50;
00058     int currentGameLoop = 0;
00059
00063     CivZero();
00064
00068     ~CivZero();
00069
00073     void gameLoop();
00074
00075     bool running;
00076 };
00077
00078 #endif // CIVZERO_H
```

5.5 src/entities/base/Entity.h File Reference

Declaration of the Entity class representing a game entity with various properties and states.

```
#include <string>
#include "utils/Size.h"
#include "entities/state/State.h"
#include "utils/ConfigManager.h"
#include <vector>
```

Classes

- class Entity

  Represents a game entity with properties such as position, size, and state.

5.5.1 Detailed Description

Declaration of the Entity class representing a game entity with various properties and states.

5.6 Entity.h

Go to the documentation of this file.

```
00001
00006 #ifndef ENTITY_H
00007 #define ENTITY_H
00008
00009 #include <string>
00010 #include "utils/Size.h"
00011 #include "entities/state/State.h"
00012 #include "utils/ConfigManager.h"
00013 #include <vector>
00014
00015 // Forward declarations
00016 class UnderConstruction;
00017 class Built;
00018
00026 class Entity
00027 {
00028 protected:
00029     std::string symbol;
00030     int effectRadius;
00031     int localEffectStrength;
00032     int globalEffectStrength;
00033     int width;
00034     int height;
00035     int xPosition;
00036     int yPosition;
00037     Size size;
00038     EntityType type;
00039     State *state;
00040     int revenue;
00041     float electricityConsumption;
00042     float waterConsumption;
00043     std::vector<Entity *> observers;
00044
00045 public:
00054     Entity(EntityConfig ec, Size size, int xPos, int yPos);
00055
00065     Entity(Entity *entity);
00066
00070     virtual ~Entity();
00071
00075     virtual void update() = 0;
00076
00083     bool isWithinEffectRadius(Entity *entity);
00084
00090     int getXPosition();
00091
00097     int getYPosition();
00098
00104     void setXPosition(int x);
00105
00111     void setYPosition(int y);
00112
00118     virtual Entity *clone() = 0;
00119
00125     int getRevenue();
00126
00132     int getWidth();
00133
00139     int getHeight();
00140
00146     bool isBuilt();
00147
00151     void updateBuildState();
00152
00158     void setSymbol(std::string symbol);
00159
00163     void subscribeToAllResidentialInRadius();
00164
00170     void subscribe(Entity *entity);
00171
00177     void unsubscribe(Entity *entity);
00178
00182     void unsubscribeFromAllBuildings();
00183
00187     void residentialBuildingPlaced();
00188
00194     const std::vector<Entity *> getObservers();
00195
00201     EntityType getType() const { return type; }
00202
00208     Size getSize() const { return size; }
00209
```

```
00215     std::string getSymbol();
00216
00222     float getElectricityConsumption();
00223
00229     float getWaterConsumption();
00230 };
00231
00232 #endif // ENTITY_H
```

5.7 Amenity.h

```
00001 #ifndef AMENITY_H
00002 #define AMENITY_H
00003
00004 #include "entities/building/base/Building.h"
00005 #include "entities/building/residential/ResidentialBuilding.h"
00006
00015 class Amenity : public Building
00016 {
00017 public:
00026     Amenity(EntityConfig ec, Size size, int xPos, int yPos);
00027
00035     Amenity(Amenity* amenity);
00036
00040     virtual ~Amenity();
00041
00045     virtual void update() = 0;
00046
00052     virtual Entity* clone() = 0;
00053 };
00054
00055 #endif // AMENITY_H
```

5.8 Monument.h

```
00001 #ifndef MONUMENT_H
00002 #define MONUMENT_H
00003
00004 #include "Amenity.h"
00005
00014 class Monument : public Amenity
00015 {
00016 public:
00025     Monument(EntityConfig ec, Size size, int xPos, int yPos);
00026
00034     Monument(Monument* monument);
00035
00039     virtual ~Monument();
00040
00044     void update();
00045
00051     Entity* clone();
00052 };
00053
00054 #endif // MONUMENT_H
```

5.9 Park.h

```
00001 #ifndef PARK_H
00002 #define PARK_H
00003
00004 #include "Amenity.h"
00005
00014 class Park : public Amenity
00015 {
00016 public:
00025     Park(EntityConfig ec, Size size, int xPos, int yPos);
00026
00034     Park(Park* park);
00035
00039     virtual ~Park();
00040
00044     void update();
00045
00051     Entity* clone();
00052 };
00053
00054 #endif // PARK_H
```

5.10 Theater.h

```
00001 #ifndef THEATER_H
00002 #define THEATER_H
00003
00004 #include "Amenity.h"
00005
00014 class Theater : public Amenity
00015 {
00016 public:
00025     Theater(EntityConfig ec, Size size, int xPos, int yPos);
00026
00034     Theater(Theater* theater);
00035
00039     virtual ~Theater();
00040
00044     void update();
00045
00051     Entity* clone();
00052 };
00053
00054 #endif // THEATER_H
```

5.11 Building.h

```
00001 #ifndef BUILDING_H
00002 #define BUILDING_H
00003
00004 #include "entities/base/Entity.h"
00005
00013 class Building : public Entity
00014 {
00015 public:
00026     Building(EntityConfig ec, Size size, int xPos, int yPos);
00027
00035     Building(Building* building);
00036
00042     virtual ~Building();
00043
00049     virtual void update() = 0;
00050
00058     virtual Entity* clone() = 0;
00059 };
00060
00061 #endif // BUILDING_H
```

5.12 EconomicBuilding.h

```
00001 #ifndef ECONOMICBUILDING_H
00002 #define ECONOMICBUILDING_H
00003
00004 #include "entities/building/base/Building.h"
00005 #include "entities/building/residential/ResidentialBuilding.h"
00006
00014 class EconomicBuilding : public Building
00015 {
00016 public:
00027     EconomicBuilding(EntityConfig ec, Size size, int xPos, int yPos);
00028
00036     EconomicBuilding(EconomicBuilding* economic);
00037
00043     virtual ~EconomicBuilding();
00044
00050     virtual void update() = 0;
00051
00059     virtual Entity* clone() = 0;
00060 };
00061
00062 #endif // ECONOMICBUILDING_H
```

5.13 Factory.h

```
00001 #ifndef FACTORY_H
00002 #define FACTORY_H
00003
```

```
00004 #include "EconomicBuilding.h"
00005
00012 class Factory : public EconomicBuilding
00013 {
00014 public:
00025     Factory(EntityConfig ec, Size size, int xPos, int yPos);
00026
00034     Factory(Factory* factory);
00035
00041     ~Factory();
00042
00048     void update();
00049
00057     Entity* clone();
00058 };
00059
00060 #endif // FACTORY_H
```

5.14 Office.h

```
00001 #ifndef OFFICE_H
00002 #define OFFICE_H
00003
00004 #include "EconomicBuilding.h"
00005
00012 class Office : public EconomicBuilding
00013 {
00014 public:
00025     Office(EntityConfig ec, Size size, int xPos, int yPos);
00026
00034     Office(Office* office);
00035
00041     ~Office();
00042
00048     void update();
00049
00057     Entity* clone();
00058 };
00059
00060 #endif // OFFICE_H
```

5.15 ShoppingMall.h

```
00001 #ifndef SHOPPINGMALL_H
00002 #define SHOPPINGMALL_H
00003
00004 #include "EconomicBuilding.h"
00005
00012 class ShoppingMall : public EconomicBuilding
00013 {
00014 public:
00025     ShoppingMall(EntityConfig ec, Size size, int xPos, int yPos);
00026
00034     ShoppingMall(ShoppingMall* mall);
00035
00041     ~ShoppingMall();
00042
00048     void update();
00049
00057     Entity* clone();
00058 };
00059
00060 #endif // SHOPPINGMALL_H
```

5.16 Apartment.h

```
00001 #ifndef APARTMENT_H
00002 #define APARTMENT_H
00003
00004 #include "ResidentialBuilding.h"
00005
00014 class Apartment : public ResidentialBuilding
00015 {
00016 public:
00025     Apartment(EntityConfig ec, Size size, int xPos, int yPos);
```

```
00026
00034     Apartment(Apartment* entity);
00035
00039     virtual ~Apartment();
00040
00046     Entity* clone();
00047 };
00048
00049 #endif // APARTMENT_H
```

5.17 House.h

```
00001 #ifndef HOUSE_H
00002 #define HOUSE_H
00003
00004 #include "ResidentialBuilding.h"
00005
00013 class House : public ResidentialBuilding
00014 {
00015 public:
00024     House(EntityConfig ec, Size size, int xPos, int yPos);
00025
00033     House(House* entity);
00034
00038     virtual ~House();
00039
00045     Entity* clone();
00046 };
00047
00048 #endif // HOUSE_H
```

5.18 ResidentialBuilding.h

```
00001 #ifndef RESIDENTIALBUILDING_H
00002 #define RESIDENTIALBUILDING_H
00003
00004 #include "entities/building/base/Building.h"
00005 #include "utils/ConfigManager.h"
00006 #include <cmath>
00007
00016 class ResidentialBuilding : public Building
00017 {
00018 private:
00019     const float RATE_OF_CHANGE = 0.5f;
00020     float globalAirport;
00021     float localAirport;
00022     float globalBusStop;
00023     float localBusStop;
00024     float globalTrainStation;
00025     float localTrainStation;
00026     float globalFactory;
00027     float localFactory;
00028     float globalShoppingMall;
00029     float localShoppingMall;
00030     float globalOffice;
00031     float localOffice;
00032     float globalHospital;
00033     float localHospital;
00034     float globalPoliceStation;
00035     float localPoliceStation;
00036     float globalSchool;
00037     float localSchool;
00038     float globalAmenity;
00039     float localAmenity;
00040     float globalUtility;
00041     float localUtility;
00042     float globalIndustry;
00043     float localIndustry;
00044     float satisfaction;
00045     int capacity;
00046
00055     void updateEntity(SatisfactionConfig sc, float &local, float &global, Entity *entity);
00056
00062     void reduceByChange(float &value);
00063
00070     void reduceByChangeWithNegativeExtreme(SatisfactionConfig sc, float &value);
00071
00072 public:
00081     ResidentialBuilding(EntityConfig ec, Size size, int xPos, int yPos);
```

```
00082
00090     ResidentialBuilding(ResidentialBuilding *entity);
00091
00095     virtual ~ResidentialBuilding();
00096
00100     void update();
00101
00107     virtual Entity *clone() = 0;
00108
00112     void reset();
00113
00117     void calculateSatisfaction();
00118
00124     float getSatisfaction();
00125
00131     void updateAirport(Entity *entity);
00132
00138     void updateBusStop(Entity *entity);
00139
00145     void updateTrainStation(Entity *entity);
00146
00152     void updateFactory(Entity *entity);
00153
00159     void updateShoppingMall(Entity *entity);
00160
00166     void updateOffice(Entity *entity);
00167
00173     void updateHospital(Entity *entity);
00174
00180     void updatePoliceStation(Entity *entity);
00181
00187     void updateSchool(Entity *entity);
00188
00194     void updateAmenity(Entity *entity);
00195
00201     void updateUtility(Entity *entity);
00202
00208     void updateIndustry(Entity *entity);
00209
00215     int getCapacity();
00216
00222     void setCapacity(int capacity);
00223 };
00224
00225 #endif // RESIDENTIALBUILDING_H
```

5.19 Hospital.h

```
00001 #ifndef HOSPITAL_H
00002 #define HOSPITAL_H
00003
00004 #include "ServiceBuilding.h"
00005
00012 class Hospital : public ServiceBuilding
00013 {
00014 public:
00023     Hospital(EntityConfig ec, Size size, int xPos, int yPos);
00024
00030     Hospital(Hospital* hospital);
00031
00037     ~Hospital();
00038
00044     void update();
00045
00051     Entity* clone();
00052 };
00053
00054 #endif // HOSPITAL_H
```

5.20 PoliceStation.h

```
00001 #ifndef POLICESTATION_H
00002 #define POLICESTATION_H
00003
00004 #include "ServiceBuilding.h"
00005
00012 class PoliceStation : public ServiceBuilding
00013 {
00014 public:
```

```
00023     PoliceStation(EntityConfig ec, Size size, int xPos, int yPos);
00024
00030     PoliceStation(PoliceStation* police);
00031
00037     ~PoliceStation();
00038
00044     void update();
00045
00051     Entity* clone();
00052 };
00053
00054 #endif // POLICESTATION_H
```

5.21 School.h

```
00001 #ifndef SCHOOL_H
00002 #define SCHOOL_H
00003
00004 #include "ServiceBuilding.h"
00005
00012 class School : public ServiceBuilding
00013 {
00014 public:
00023     School(EntityConfig ec, Size size, int xPos, int yPos);
00024
00030     School(School* school);
00031
00037     ~School();
00038
00044     void update();
00045
00051     Entity* clone();
00052 };
00053
00054 #endif // SCHOOL_H
```

5.22 ServiceBuilding.h

```
00001 #ifndef SERVICEBUILDING_H
00002 #define SERVICEBUILDING_H
00003
00004 #include "entities/building/base/Building.h"
00005 #include "entities/building/residential/ResidentialBuilding.h"
00006
00013 class ServiceBuilding : public Building
00014 {
00015 public:
00024     ServiceBuilding(EntityConfig ec, Size size, int xPos, int yPos);
00025
00031     ServiceBuilding(ServiceBuilding* service);
00032
00038     virtual ~ServiceBuilding();
00039
00045     virtual void update() = 0;
00046
00052     virtual Entity* clone() = 0;
00053 };
00054
00055 #endif // SERVICEBUILDING_H
```

5.23 Industry.h

```
00001 #ifndef INDUSTRY_H
00002 #define INDUSTRY_H
00003
00004 #include "entities/base/Entity.h"
00005 #include "entities/building/residential/ResidentialBuilding.h"
00006
00015 class Industry : public Entity
00016 {
00017 private:
00018     int output;
00019     Cost cost;
00020 public:
00029     Industry(EntityConfig ec, Size size, int xPos, int yPos);
```

```
00030
00038     Industry(Industry *industry);
00039
00043     virtual ~Industry();
00044
00050     virtual void update() = 0;
00051
00059     virtual Entity *clone() = 0;
00060
00066     virtual int getOutput();
00067
00073     void setOutput(int output);
00074
00080     virtual int getLevel();
00081
00087     virtual Cost getCost();
00088
00094     virtual Entity* upgrade() = 0;
00095 };
00096
00097 #endif // INDUSTRY_H
```

5.24 ConcreteProducer.h

```
00001 #ifndef CONCRETEPRODUCER_H
00002 #define CONCRETEPRODUCER_H
00003
00004 #include "entities/industry/base/Industry.h"
00005
00006 class ConcreteProducerLevelOneUpgrade;
00007
00015 class ConcreteProducer : public Industry
00016 {
00017 public:
00026     ConcreteProducer(EntityConfig ec, Size size, int xPos, int yPos);
00027
00035     ConcreteProducer(ConcreteProducer *concreteProducer);
00036
00040     virtual ~ConcreteProducer();
00041
00047     void update() override;
00048
00054     Entity *clone() override;
00055
00061     Entity *upgrade() override;
00062 };
00063
00064 #endif // CONCRETEPRODUCER_H
```

5.25 ConcreteProducerLevelOneUpgrade.h

```
00001 #ifndef CONCRETEPRODUCERLEVELONEUPGRADE_H
00002 #define CONCRETEPRODUCERLEVELONEUPGRADE_H
00003
00004 #include "entities/industry/concreteproducer/ConcreteProducerUpgrade.h"
00005
00013 class ConcreteProducerLevelOneUpgrade : public ConcreteProducerUpgrade
00014 {
00015 public:
00021     ConcreteProducerLevelOneUpgrade(ConcreteProducer *concreteProd);
00022
00030     ConcreteProducerLevelOneUpgrade(ConcreteProducerLevelOneUpgrade *concreteProd);
00031
00035     ~ConcreteProducerLevelOneUpgrade();
00036
00042     void update() override;
00043
00049     int getOutput() override;
00050
00056     int getLevel() override;
00057
00063     Entity *clone() override;
00064
00070     Entity *upgrade() override;
00071
00077     Cost getCost() override;
00078
00079 private:
00080     const int UPGRADE = 2;
```

```
00081 };
00082
00083 #endif // CONCRETEPRODUCERLEVELONEUPGRADE_H
```

5.26 ConcreteProducerLevelThreeUpgrade.h

```
00001 #ifndef CONCRETEPRODUCERLEVELTHREEUPGRADE_H
00002 #define CONCRETEPRODUCERLEVELTHREEUPGRADE_H
00003
00004 #include "entities/industry/concreteproducer/ConcreteProducerUpgrade.h"
00005
00013 class ConcreteProducerLevelThreeUpgrade : public ConcreteProducerUpgrade
00014 {
00015 public:
00021     ConcreteProducerLevelThreeUpgrade(ConcreteProducer *concreteProd);
00022
00030     ConcreteProducerLevelThreeUpgrade(ConcreteProducerLevelThreeUpgrade *concreteProd);
00031
00035     ~ConcreteProducerLevelThreeUpgrade();
00036
00042     void update() override;
00043
00049     int getOutput() override;
00050
00056     int getLevel() override;
00057
00063     Entity *clone() override;
00064
00070     Cost getCost() override;
00071
00077     Entity *upgrade() override;
00078
00079 private:
00080     const int UPGRADE = 6;
00081 };
00082
00083 #endif // CONCRETEPRODUCERLEVELTHREEUPGRADE_H
```

5.27 ConcreteProducerLevelTwoUpgrade.h

```
00001 #ifndef CONCRETEPRODUCERLEVELTWOUPGRADE_H
00002 #define CONCRETEPRODUCERLEVELTWOUPGRADE_H
00003
00004 #include "entities/industry/concreteproducer/ConcreteProducerUpgrade.h"
00005
00013 class ConcreteProducerLevelTwoUpgrade : public ConcreteProducerUpgrade
00014 {
00015 public:
00021     ConcreteProducerLevelTwoUpgrade(ConcreteProducer *concreteProd);
00022
00030     ConcreteProducerLevelTwoUpgrade(ConcreteProducerLevelTwoUpgrade *concreteProd);
00031
00035     ~ConcreteProducerLevelTwoUpgrade();
00036
00042     void update() override;
00043
00049     int getOutput() override;
00050
00056     int getLevel() override;
00057
00063     Entity *clone() override;
00064
00070     Entity *upgrade() override;
00071
00077     Cost getCost() override;
00078
00079 private:
00080     const int UPGRADE = 4;
00081 };
00082
00083 #endif // CONCRETEPRODUCERLEVELTWOUPGRADE_H
```

5.28 ConcreteProducerUpgrade.h

```
00001 #ifndef CONCRETEPRODUCERUPGRADE_H
```

```
00002 #define CONCRETEPRODUCERUPGRADE_H
00003
00004 class ConcreteProducer;
00005
00006 #include "ConcreteProducer.h"
00007
00015 class ConcreteProducerUpgrade : public ConcreteProducer
00016 {
00017 public:
00023     ConcreteProducerUpgrade(ConcreteProducer *concreteProd);
00024
00032     ConcreteProducerUpgrade(ConcreteProducerUpgrade *concreteProd);
00033
00037     virtual ~ConcreteProducerUpgrade();
00038
00044     virtual int getOutput() = 0;
00045
00051     virtual Entity *clone() = 0;
00052
00058     virtual void update() = 0;
00059
00065     virtual Cost getCost() = 0;
00066
00072     virtual Entity* upgrade() = 0;
00073
00074 protected:
00075     ConcreteProducer *concreteProducer;
00076 };
00077
00078 #endif // CONCRETEPRODUCERUPGRADE_H
```

5.29 StoneProducer.h

```
00001 #ifndef STONEPRODUCER_H
00002 #define STONEPRODUCER_H
00003
00004 #include "entities/industry/base/Industry.h"
00005
00012 class StoneProducer : public Industry
00013 {
00014 public:
00020     StoneProducer(StoneProducer *stoneProducer);
00021
00030     StoneProducer(EntityConfig ec, Size size, int xPos, int yPos);
00031
00035     virtual ~StoneProducer();
00036
00042     void update() override;
00043
00049     Entity *clone() override;
00050
00056     Entity *upgrade() override;
00057 };
00058
00059 #endif // STONEPRODUCER_H
```

5.30 StoneProducerLevelOneUpgrade.h

```
00001 #ifndef STONEPRODUCERLEVELONEUPGRADE_H
00002 #define STONEPRODUCERLEVELONEUPGRADE_H
00003
00004 #include "StoneProducerUpgrade.h"
00005
00013 class StoneProducerLevelTwoUgprade; // Typo: "Ugprade" should be "Upgrade."
00014
00015 class StoneProducerLevelOneUpgrade : public StoneProducerUpgrade
00016 {
00017 public:
00023     StoneProducerLevelOneUpgrade(StoneProducer *stoneProd);
00024
00030     StoneProducerLevelOneUpgrade(StoneProducerLevelOneUpgrade *stoneProd);
00031
00035     ~StoneProducerLevelOneUpgrade();
00036
00042     int getOutput() override;
00043
00049     int getLevel() override;
00050
00056     Entity *clone() override;
```

```
00057
00061     void update() override;
00062
00068     Entity *upgrade() override;
00069
00075     Cost getCost() override;
00076
00077 private:
00078     const int UPGRADE = 2;
00079 };
00080
00081 #endif // STONEPRODUCERLEVELONEUPGRADE_H
```

5.31 StoneProducerLevelThreeUpgrade.h

```
00001 #ifndef STONEPRODUCERLEVELTHREEUPGRADE_H
00002 #define STONEPRODUCERLEVELTHREEUPGRADE_H
00003
00004 #include "StoneProducerUpgrade.h"
00005
00013 class StoneProducerLevelThreeUpgrade : public StoneProducerUpgrade
00014 {
00015 public:
00021     StoneProducerLevelThreeUpgrade(StoneProducer *stoneProd);
00022
00028     StoneProducerLevelThreeUpgrade(StoneProducerLevelThreeUpgrade *stoneProd);
00029
00033     ~StoneProducerLevelThreeUpgrade();
00034
00038     void update() override;
00039
00045     int getOutput() override;
00046
00052     int getLevel() override;
00053
00059     Entity *clone() override;
00060
00066     Entity *upgrade() override;
00067
00073     Cost getCost() override;
00074
00075 private:
00076     const int UPGRADE = 6;
00077 };
00078
00079 #endif // STONEPRODUCERLEVELTHREEUPGRADE_H
```

5.32 StoneProducerLevelTwoUpgrade.h

```
00001 #ifndef STONEPRODUCERLEVELTWOUPGRADE_H
00002 #define STONEPRODUCERLEVELTWOUPGRADE_H
00003
00004 #include "StoneProducerUpgrade.h"
00005
00013 class StoneProducerLevelThreeUpgrade; // Forward declaration for level three upgrade.
00014
00015 class StoneProducerLevelTwoUpgrade : public StoneProducerUpgrade
00016 {
00017 public:
00023     StoneProducerLevelTwoUpgrade(StoneProducer *stoneProd);
00024
00030     StoneProducerLevelTwoUpgrade(StoneProducerLevelTwoUpgrade *stoneProd);
00031
00035     ~StoneProducerLevelTwoUpgrade();
00036
00042     Entity *clone() override;
00043
00047     void update() override;
00048
00054     int getOutput() override;
00055
00061     int getLevel() override;
00062
00068     Entity *upgrade() override;
00069
00075     Cost getCost() override;
00076
00077 private:
00078     const int UPGRADE = 4;
```

```
00079 };
00080
00081 #endif // STONEPRODUCERLEVELTWOUPGRADE_H
```

5.33 StoneProducerUpgrade.h

```
00001 #ifndef STONEPRODUCERUPGRADE_H
00002 #define STONEPRODUCERUPGRADE_H
00003
00004 #include "StoneProducer.h"
00005
00012 class StoneProducerUpgrade : public StoneProducer
00013 {
00014 public:
00020     StoneProducerUpgrade(StoneProducer *stoneProd);
00021
00027     StoneProducerUpgrade(StoneProducerUpgrade *stoneProd);
00028
00032     virtual ~StoneProducerUpgrade();
00033
00039     virtual void update() = 0;
00040
00046     virtual Entity *clone() = 0;
00047
00053     virtual int getOutput() = 0;
00054
00060     virtual Entity* upgrade() = 0;
00061
00067     virtual Cost getCost() = 0;
00068
00069 protected:
00070     StoneProducer *stoneProducer;
00071 };
00072
00073 #endif // STONEPRODUCERUPGRADE_H
```

5.34 WoodProducer.h

```
00001 #ifndef WOODPRODUCER_H
00002 #define WOODPRODUCER_H
00003
00004 #include "entities/industry/base/Industry.h"
00005
00013 class WoodProducer : public Industry
00014 {
00015 public:
00024     WoodProducer(EntityConfig ec, Size size, int xPos, int yPos);
00025
00031     WoodProducer(WoodProducer *woodProducer);
00032
00036     virtual ~WoodProducer();
00037
00041     void update() override;
00042
00048     Entity *clone() override;
00049
00055     Entity *upgrade() override;
00056 };
00057
00058 #endif // WOODPRODUCER_H
```

5.35 WoodProducerLevelOneUpgrade.h

```
00001 #ifndef WOODPRODUCERLEVELONEUPGRADE_H
00002 #define WOODPRODUCERLEVELONEUPGRADE_H
00003
00004 #include "WoodProducerUpgrade.h"
00005
00014 class WoodProducerLevelOneUpgrade : public WoodProducerUpgrade
00015 {
00016 public:
00022     WoodProducerLevelOneUpgrade(WoodProducer *woodProducer);
00023
00029     WoodProducerLevelOneUpgrade(WoodProducerLevelOneUpgrade *woodprod);
00030
```

```
00034     ~WoodProducerLevelOneUpgrade();
00035
00039     void update() override;
00040
00046     Entity *clone() override;
00047
00053     int getOutput() override;
00054
00060     int getLevel() override;
00061
00067     Entity *upgrade() override;
00068
00074     Cost getCost() override;
00075
00076 private:
00077     const int UPGRADE = 2;
00078 };
00079
00080 #endif // WOODPRODUCERLEVELONEUPGRADE_H
```

5.36 WoodProducerLevelThreeUpgrade.h

```
00001 #ifndef WOODPRODUCERLEVELTHREEUPGRADE_H
00002 #define WOODPRODUCERLEVELTHREEUPGRADE_H
00003
00004 #include "WoodProducerUpgrade.h"
00005
00014 class WoodProducerLevelThreeUpgrade : public WoodProducerUpgrade
00015 {
00016 public:
00022     WoodProducerLevelThreeUpgrade(WoodProducer *woodProd);
00023
00029     WoodProducerLevelThreeUpgrade(WoodProducerLevelThreeUpgrade *woodProd);
00030
00034     ~WoodProducerLevelThreeUpgrade();
00035
00039     void update() override;
00040
00046     Entity *clone() override;
00047
00053     int getOutput() override;
00054
00060     int getLevel() override;
00061
00067     Entity *upgrade() override;
00068
00074     Cost getCost() override;
00075
00076 private:
00077     const int UPGRADE = 6;
00078 };
00079
00080 #endif // WOODPRODUCERLEVELTHREEUPGRADE_H
```

5.37 WoodProducerLevelTwoUpgrade.h

```
00001 #ifndef WOODPRODUCERLEVELTWOUPGRADE_H
00002 #define WOODPRODUCERLEVELTWOUPGRADE_H
00003
00004 #include "WoodProducerUpgrade.h"
00005
00014 class WoodProducerLevelTwoUpgrade : public WoodProducerUpgrade
00015 {
00016 public:
00022     WoodProducerLevelTwoUpgrade(WoodProducer *woodProducer);
00023
00029     WoodProducerLevelTwoUpgrade(WoodProducerLevelTwoUpgrade *woodProd);
00030
00034     ~WoodProducerLevelTwoUpgrade();
00035
00039     void update() override;
00040
00046     Entity *clone() override;
00047
00053     int getOutput() override;
00054
00060     int getLevel() override;
00061
00067     Entity *upgrade() override;
```

```
00068
00074     Cost getCost() override;
00075
00076 private:
00077     const int UPGRADE = 4;
00078 };
00079
00080 #endif // WOODPRODUCERLEVELTWOUPGRADE_H
```

5.38 WoodProducerUpgrade.h

```
00001 #ifndef WOODPRODUCERUPGRADE_H
00002 #define WOODPRODUCERUPGRADE_H
00003
00004 #include "WoodProducer.h"
00005
00013 class WoodProducerUpgrade : public WoodProducer
00014 {
00015 public:
00021     WoodProducerUpgrade(WoodProducer *woodProducer);
00022
00026     virtual ~WoodProducerUpgrade();
00027
00033     WoodProducerUpgrade(WoodProducerUpgrade *woodProducerUpgrade);
00034
00040     virtual void update() = 0;
00041
00049     virtual int getOutput() = 0;
00050
00058     virtual Cost getCost() = 0;
00059
00067     virtual Entity *upgrade() = 0;
00068
00076     virtual Entity *clone() = 0;
00077
00078 protected:
00079     WoodProducer *woodProducer;
00080 };
00081
00082 #endif // WOODPRODUCERUPGRADE_H
```

5.39 Road.h

```
00001 #ifndef ROAD_H
00002 #define ROAD_H
00003
00004 #include "entities/base/Entity.h"
00005
00014 class Road : public Entity
00015 {
00016 public:
00025     Road(EntityConfig ec, Size size, int xPos, int yPos);
00026
00034     Road(Road *road);
00035
00039     ~Road();
00040
00044     void update();
00045
00051     Entity *clone();
00052 };
00053
00054 #endif // ROAD_H
```

5.40 Built.h

```
00001 #ifndef BUILT_H
00002 #define BUILT_H
00003
00004 #include "State.h"
00005
00014 class Built : public State
00015 {
00016 public:
00021     Built(int buildTime);
```

```
00022
00030     Built(Built* built);
00031
00035     ~Built();
00036
00041     State* update();
00042
00051     State* clone();
00052 };
00053
00054 #endif // BUILT_H
```

5.41 State.h

```
00001 #ifndef STATE_H
00002 #define STATE_H
00003
00012 class State
00013 {
00014 private:
00015     int gameLoopCounter;
00016     int buildTime;
00017
00018 public:
00023     State(int buildTime);
00024
00032     State(State* state);
00033
00037     virtual ~State();
00038
00043     virtual State* update() = 0;
00044
00053     virtual State* clone() = 0;
00054
00059     int getGameLoopCounter();
00060
00065     int getBuildTime();
00066
00070     void incrementGameLoopCounter();
00071 };
00072
00073 #endif // STATE_H
```

5.42 UnderConstruction.h

```
00001 #ifndef UNDERCONSTRUCTION_H
00002 #define UNDERCONSTRUCTION_H
00003
00004 #include "State.h"
00005
00013 class UnderConstruction : public State
00014 {
00015 public:
00020     UnderConstruction(int buildTime);
00021
00029     UnderConstruction(UnderConstruction* underConstruction);
00030
00034     ~UnderConstruction();
00035
00040     State* update();
00041
00050     State* clone();
00051 };
00052
00053 #endif // UNDERCONSTRUCTION_H
```

5.43 Airport.h

```
00001 #ifndef AIRPORT_H
00002 #define AIRPORT_H
00003
00004 #include "Transport.h"
00005
00013 class Airport : public Transport
00014 {
```

```
00015 public:
00024     Airport(EntityConfig ec, Size size, int xPos, int yPos);
00025
00033     Airport(Airport* airport);
00034
00038     virtual ~Airport();
00039
00043     void update();
00044
00050     Entity* clone();
00051 };
00052
00053 #endif // AIRPORT_H
```

5.44 BusStop.h

```
00001 #ifndef BUSSTOP_H
00002 #define BUSSTOP_H
00003
00004 #include "Transport.h"
00005
00013 class BusStop : public Transport
00014 {
00015 public:
00024     BusStop(EntityConfig ec, Size size, int xPos, int yPos);
00025
00033     BusStop(BusStop* busStop);
00034
00038     virtual ~BusStop();
00039
00043     void update();
00044
00050     Entity* clone();
00051 };
00052
00053 #endif // BUSSTOP_H
```

5.45 TrainStation.h

```
00001 #ifndef TRAINSTATION_H
00002 #define TRAINSTATION_H
00003
00004 #include "Transport.h"
00005
00013 class TrainStation : public Transport
00014 {
00015 public:
00024     TrainStation(EntityConfig ec, Size size, int xPos, int yPos);
00025
00033     TrainStation(TrainStation* trainStation);
00034
00038     virtual ~TrainStation();
00039
00043     void update();
00044
00050     Entity* clone();
00051 };
00052
00053 #endif // TRAINSTATION_H
```

5.46 Transport.h

```
00001 #ifndef TRANSPORT_H
00002 #define TRANSPORT_H
00003
00004 #include "entities/base/Entity.h"
00005 #include "entities/building/residential/ResidentialBuilding.h"
00006
00015 class Transport : public Entity
00016 {
00017 public:
00026     Transport(EntityConfig ec, Size size, int xPos, int yPos);
00027
00035     Transport(Transport* transport);
00036
```

```
00040     virtual ~Transport();
00041
00047     virtual void update() = 0;
00048
00056     virtual Entity* clone() = 0;
00057 };
00058
00059 #endif // TRANSPORT_H
```

5.47 Utility.h

```
00001 #ifndef UTILITY_H
00002 #define UTILITY_H
00003
00004 #include "entities/base/Entity.h"
00005 #include "entities/building/residential/ResidentialBuilding.h"
00006 #include <string>
00007
00014 class Utility : public Entity
00015 {
00016 public:
00028     Utility(EntityConfig ec, Size size, int xPos, int yPos);
00029
00037     Utility(Utility *utility);
00038
00042     virtual ~Utility();
00043
00049     virtual void update() = 0;
00050
00058     virtual Entity *clone() = 0;
00059
00064     virtual Entity *upgrade() = 0;
00065
00071     virtual int getOutput();
00072
00078     void setOutput(int output);
00079
00084     virtual Cost getCost();
00085
00091     virtual int getLevel();
00092
00093 private:
00094     int output;
00095     Cost cost;
00096 };
00097
00098 #endif // UTILITY_H
```

5.48 PowerPlant.h

```
00001 #ifndef POWERPLANT_H
00002 #define POWERPLANT_H
00003
00004 #include "entities/utility/base/Utility.h"
00005
00006 class PowerPlantLevelOneUpgrade;
00007
00014 class PowerPlant : public Utility
00015 {
00016 public:
00027     PowerPlant(EntityConfig ec, Size size, int xPos, int yPos);
00028
00036     PowerPlant(PowerPlant *power);
00037
00041     virtual ~PowerPlant();
00042
00048     void update() override;
00049
00057     Entity *clone() override;
00058
00063     Entity *upgrade() override;
00064 };
00065
00066 #endif // POWERPLANT_H
```

5.49 PowerPlantLevelOneUpgrade.h

```
00001 #ifndef POWERPLANTLEVELONEUPGRADE_H
00002 #define POWERPLANTLEVELONEUPGRADE_H
00003
00004 #include "PowerPlantUpgrade.h"
00005
00006 class PowerPlantLevelTwo;
00007
00015 class PowerPlantLevelOneUpgrade : public PowerPlantUpgrade
00016 {
00017 public:
00025     PowerPlantLevelOneUpgrade(PowerPlant *power);
00026
00035     PowerPlantLevelOneUpgrade(PowerPlantLevelOneUpgrade *pPLOU);
00036
00042     ~PowerPlantLevelOneUpgrade();
00043
00049     void update() override;
00050
00058     Entity *clone() override;
00059
00064     Entity *upgrade() override;
00065
00073     int getOutput() override;
00074
00079     Cost getCost() override;
00080
00086     int getLevel() override;
00087
00088 private:
00089     const int UPGRADE = 2;
00090 };
00091
00092 #endif // POWERPLANTLEVELONEUPGRADE_H
```

5.50 PowerPlantLevelThreeUpgrade.h

```
00001 #ifndef POWERPLANTLEVELTHREEUPGRADE_H
00002 #define POWERPLANTLEVELTHREEUPGRADE_H
00003
00004 #include "PowerPlantUpgrade.h"
00005
00013 class PowerPlantLevelThreeUpgrade : public PowerPlantUpgrade
00014 {
00015 public:
00023     PowerPlantLevelThreeUpgrade(PowerPlant *power);
00024
00033     PowerPlantLevelThreeUpgrade(PowerPlantLevelThreeUpgrade *pPLTU);
00034
00040     ~PowerPlantLevelThreeUpgrade();
00041
00047     void update() override;
00048
00056     Entity *clone() override;
00057
00062     Entity *upgrade() override;
00063
00071     int getOutput() override;
00072
00077     Cost getCost() override;
00078
00084     int getLevel() override;
00085
00086 private:
00087     const int UPGRADE = 6;
00088 };
00089
00090 #endif // POWERPLANTLEVELTHREEUPGRADE_H
```

5.51 PowerPlantLevelTwoUpgrade.h

```
00001 #ifndef POWERPLANTLEVELTWOUPGRADE_H
00002 #define POWERPLANTLEVELTWOUPGRADE_H
00003
00004 #include "PowerPlantUpgrade.h"
00005
00006 class PowerPlantLevelThree;
00007
```

```
00015 class PowerPlantLevelTwoUpgrade : public PowerPlantUpgrade
00016 {
00017 public:
00025     PowerPlantLevelTwoUpgrade(PowerPlant *power);
00026
00035     PowerPlantLevelTwoUpgrade(PowerPlantLevelTwoUpgrade *pPLTU);
00036
00042     ~PowerPlantLevelTwoUpgrade();
00043
00049     void update() override;
00050
00058     Entity *clone() override;
00059
00064     Entity *upgrade() override;
00065
00073     int getOutput() override;
00074
00079     Cost getCost() override;
00080
00086     int getLevel() override;
00087
00088 private:
00089     const int UPGRADE = 4;
00090 };
00091
00092 #endif // POWERPLANTLEVELTWOUPGRADE_H
```

5.52 PowerPlantUpgrade.h

```
00001 #ifndef POWERPLANTUPGRADE_H
00002 #define POWERPLANTUPGRADE_H
00003
00004 #include "entities/utility/powerplant/PowerPlant.h"
00005
00013 class PowerPlantUpgrade : public PowerPlant
00014 {
00015 public:
00023     PowerPlantUpgrade(PowerPlant *power);
00024
00032     PowerPlantUpgrade(PowerPlantUpgrade *pPU);
00033
00037     virtual ~PowerPlantUpgrade();
00038
00044     virtual void update() = 0;
00045
00053     virtual Entity *clone() = 0;
00054
00059     virtual Entity *upgrade() = 0;
00060
00066     virtual int getOutput() = 0;
00067
00072     virtual Cost getCost() = 0;
00073
00074 protected:
00075     PowerPlant *powerPlant;
00076 };
00077
00078 #endif // POWERPLANTUPGRADE_H
```

5.53 SewageSystem.h

```
00001 #ifndef SEWAGESYSTEM_H
00002 #define SEWAGESYSTEM_H
00003
00004 #include "entities/utility/base/Utility.h"
00005
00006 class SewageSystemLevelOneUpgrade;
00007
00014 class SewageSystem : public Utility
00015 {
00016 public:
00027     SewageSystem(EntityConfig ec, Size size, int xPos, int yPos);
00028
00036     SewageSystem(SewageSystem *sewage);
00037
00041     virtual ~SewageSystem();
00042
00048     void update() override;
00049
```

```
00057     Entity *clone() override;
00058
00063     Entity *upgrade() override;
00064 };
00065
00066 #endif // SEWAGESYSTEM_H
```

5.54 SewageSystemLevelOneUpgrade.h

```
00001 #ifndef SEWAGESYSTEMLEVELONEUPGRADE_H
00002 #define SEWAGESYSTEMLEVELONEUPGRADE_H
00003
00004 #include "SewageSystemUpgrade.h"
00005
00006 class SewageSystemLevelTwoUpgrade;
00007
00015 class SewageSystemLevelOneUpgrade : public SewageSystemUpgrade
00016 {
00017 public:
00025     SewageSystemLevelOneUpgrade(SewageSystem *sewage);
00026
00035     SewageSystemLevelOneUpgrade(SewageSystemLevelOneUpgrade *sSLOU);
00036
00042     ~SewageSystemLevelOneUpgrade();
00043
00049     void update() override;
00050
00058     Entity *clone() override;
00059
00064     Entity *upgrade() override;
00065
00073     int getOutput() override;
00074
00079     Cost getCost() override;
00080
00086     int getLevel() override;
00087
00088 private:
00089     const int UPGRADE = 2;
00090 };
00091
00092 #endif // SEWAGESYSTEMLEVELONEUPGRADE_H
```

5.55 SewageSystemLevelThreeUpgrade.h

```
00001 #ifndef SEWAGESYSTEMLEVELTHREEUPGRADE_H
00002 #define SEWAGESYSTEMLEVELTHREEUPGRADE_H
00003
00004 #include "SewageSystemUpgrade.h"
00005
00013 class SewageSystemLevelThreeUpgrade : public SewageSystemUpgrade
00014 {
00015 public:
00023     SewageSystemLevelThreeUpgrade(SewageSystem *sewage);
00024
00033     SewageSystemLevelThreeUpgrade(SewageSystemLevelThreeUpgrade *sSLTU);
00034
00040     ~SewageSystemLevelThreeUpgrade();
00041
00047     void update() override;
00048
00056     Entity *clone() override;
00057
00062     Entity *upgrade() override;
00063
00071     int getOutput() override;
00072
00077     Cost getCost() override;
00078
00084     int getLevel() override;
00085
00086 private:
00087     const int UPGRADE = 6;
00088 };
00089
00090 #endif // SEWAGESYSTEMLEVELTHREEUPGRADE_H
```

5.56 SewageSystemLevelTwoUpgrade.h

```
00001 #ifndef SEWAGESYSTEMLEVELTWOUPGRADE_H
00002 #define SEWAGESYSTEMLEVELTWOUPGRADE_H
00003
00004 #include "SewageSystemUpgrade.h"
00005
00006 class SewageSystemLevelThreeUpgrade;
00007
00015 class SewageSystemLevelTwoUpgrade : public SewageSystemUpgrade
00016 {
00017 public:
00025     SewageSystemLevelTwoUpgrade(SewageSystem *sewage);
00026
00035     SewageSystemLevelTwoUpgrade(SewageSystemLevelTwoUpgrade *sSLTU);
00036
00042     ~SewageSystemLevelTwoUpgrade();
00043
00049     void update() override;
00050
00058     Entity *clone() override;
00059
00064     Entity *upgrade() override;
00065
00073     int getOutput() override;
00074
00079     Cost getCost() override;
00080
00086     int getLevel() override;
00087
00088 private:
00089     const int UPGRADE = 4;
00090 };
00091
00092 #endif // SEWAGESYSTEMLEVELTWOUPGRADE_H
```

5.57 SewageSystemUpgrade.h

```
00001 #ifndef SEWAGESYSTEMUPGRADE_H
00002 #define SEWAGESYSTEMUPGRADE_H
00003
00004 #include "entities/utility/sewagesystem/SewageSystem.h"
00005
00013 class SewageSystemUpgrade : public SewageSystem
00014 {
00015 public:
00023     SewageSystemUpgrade(SewageSystem *sewage);
00024
00032     SewageSystemUpgrade(SewageSystemUpgrade *sSU);
00033
00037     virtual ~SewageSystemUpgrade();
00038
00042     virtual void update() = 0;
00043
00049     virtual Entity *clone() = 0;
00050
00055     virtual Entity *upgrade() = 0;
00056
00062     virtual int getOutput() = 0;
00063
00068     virtual Cost getCost() = 0;
00069
00070 protected:
00071     SewageSystem *sewageSystem;
00072 };
00073
00074 #endif // SEWAGESYSTEMUPGRADE_H
```

5.58 WasteManagement.h

```
00001 #ifndef WASTEMANAGEMENT_H
00002 #define WASTEMANAGEMENT_H
00003
00004 #include "entities/utility/base/Utility.h"
00005
00006 class WasteManagementLevelOneUpgrade;
00007
00014 class WasteManagement : public Utility
00015 {
```

```
00016 public:
00027     WasteManagement(EntityConfig ec, Size size, int xPos, int yPos);
00028
00036     WasteManagement(WasteManagement *waste);
00037
00041     virtual ~WasteManagement();
00042
00048     void update() override;
00049
00057     Entity *clone() override;
00058
00063     Entity *upgrade() override;
00064 };
00065
00066 #endif // WASTEMANAGEMENT_H
```

5.59 WasteManagementLevelOneUpgrade.h

```
00001 #ifndef WASTEMANAGEMENTLEVELONEUPGRADE_H
00002 #define WASTEMANAGEMENTLEVELONEUPGRADE_H
00003
00004 #include "WasteManagementUpgrade.h"
00005
00006 class WasteManagementLevelTwoUpgrade;
00007
00015 class WasteManagementLevelOneUpgrade : public WasteManagementUpgrade
00016 {
00017 public:
00025     WasteManagementLevelOneUpgrade(WasteManagement *waste);
00026
00035     WasteManagementLevelOneUpgrade(WasteManagementLevelOneUpgrade *wMLOU);
00036
00042     ~WasteManagementLevelOneUpgrade();
00043
00049     void update() override;
00050
00058     Entity *clone() override;
00059
00064     Entity *upgrade() override;
00065
00073     int getOutput() override;
00074
00079     Cost getCost() override;
00080
00086     int getLevel() override;
00087
00088 private:
00089     const int UPGRADE = 2;
00090 };
00091
00092 #endif // WASTEMANAGEMENTLEVELONEUPGRADE_H
```

5.60 WasteManagementLevelThreeUpgrade.h

```
00001 #ifndef WASTEMANAGEMENTLEVELTHREEUPGRADE_H
00002 #define WASTEMANAGEMENTLEVELTHREEUPGRADE_H
00003
00004 #include "WasteManagementUpgrade.h"
00005
00013 class WasteManagementLevelThreeUpgrade : public WasteManagementUpgrade
00014 {
00015 public:
00023     WasteManagementLevelThreeUpgrade(WasteManagement *waste);
00024
00033     WasteManagementLevelThreeUpgrade(WasteManagementLevelThreeUpgrade *wMLTU);
00034
00040     ~WasteManagementLevelThreeUpgrade();
00041
00047     void update() override;
00048
00056     Entity *clone() override;
00057
00062     Entity *upgrade() override;
00063
00071     int getOutput() override;
00072
00077     Cost getCost() override;
00078
00084     int getLevel() override;
```

```
00085
00086 private:
00087     const int UPGRADE = 6;
00088 };
00089
00090 #endif // WASTEMANAGEMENTLEVELTHREEUPGRADE_H
```

5.61 WasteManagementLevelTwoUpgrade.h

```
00001 #ifndef WASTEMANAGEMENTLEVELTWOUPGRADE_H
00002 #define WASTEMANAGEMENTLEVELTWOUPGRADE_H
00003
00004 #include "WasteManagementUpgrade.h"
00005
00006 class WasteManagementLevelThreeUpgrade;
00007
00015 class WasteManagementLevelTwoUpgrade : public WasteManagementUpgrade
00016 {
00017 public:
00025     WasteManagementLevelTwoUpgrade(WasteManagement *waste);
00026
00034     WasteManagementLevelTwoUpgrade(WasteManagementLevelTwoUpgrade *wMLTU);
00035
00039     ~WasteManagementLevelTwoUpgrade();
00040
00044     void update() override;
00045
00051     Entity *clone() override;
00052
00057     Entity *upgrade() override;
00058
00066     int getOutput() override;
00067
00072     Cost getCost() override;
00073
00079     int getLevel() override;
00080
00081 private:
00082     const int UPGRADE = 4;
00083 };
00084
00085 #endif // WASTEMANAGEMENTLEVELTWOUPGRADE_H
```

5.62 WasteManagementUpgrade.h

```
00001 #ifndef WASTEMANAGEMENTUPGRADE_H
00002 #define WASTEMANAGEMENTUPGRADE_H
00003
00004 #include "entities/utility/wastemanagement/WasteManagement.h"
00005
00013 class WasteManagementUpgrade : public WasteManagement
00014 {
00015 public:
00023     WasteManagementUpgrade(WasteManagement *waste);
00024
00032     WasteManagementUpgrade(WasteManagementUpgrade *wMU);
00033
00037     virtual ~WasteManagementUpgrade();
00038
00042     virtual void update() = 0;
00043
00049     virtual Entity *clone() = 0;
00050
00055     virtual Entity *upgrade() = 0;
00056
00062     virtual int getOutput() = 0;
00063
00068     virtual Cost getCost() = 0;
00069
00070 protected:
00071     WasteManagement *wasteManagement;
00072 };
00073
00074 #endif // WASTEMANAGEMENTUPGRADE_H
```

5.63 WaterSupply.h

```
00001 #ifndef WATERSUPPLY_H
00002 #define WATERSUPPLY_H
00003
00004 #include "entities/utility/base/Utility.h"
00005
00006 class WaterSupplyLevelOneUpgrade;
00007
00014 class WaterSupply : public Utility
00015 {
00016 public:
00027     WaterSupply(EntityConfig ec, Size size, int xPos, int yPos);
00028
00036     WaterSupply(WaterSupply *water);
00037
00041     virtual ~WaterSupply();
00042
00048     void update() override;
00049
00057     Entity *clone() override;
00058
00063     Entity *upgrade() override;
00064 };
00065
00066 #endif // WATERSUPPLY_H
```

5.64 WaterSupplyLevelOneUpgrade.h

```
00001 #ifndef WATERSUPPLYLEVELONEUPGRADE_H
00002 #define WATERSUPPLYLEVELONEUPGRADE_H
00003
00004 #include "WaterSupplyUpgrade.h"
00005
00006 class WaterSupplyLevelTwoUpgrade;
00007
00015 class WaterSupplyLevelOneUpgrade : public WaterSupplyUpgrade
00016 {
00017 public:
00025     WaterSupplyLevelOneUpgrade(WaterSupply *water);
00026
00035     WaterSupplyLevelOneUpgrade(WaterSupplyLevelOneUpgrade *wSLOU);
00036
00042     ~WaterSupplyLevelOneUpgrade();
00043
00049     void update() override;
00050
00058     Entity *clone() override;
00059
00064     Entity *upgrade() override;
00065
00073     int getOutput() override;
00074
00079     Cost getCost() override;
00080
00086     int getLevel() override;
00087
00088 private:
00089     const int UPGRADE = 2;
00090 };
00091
00092 #endif // WATERSUPPLYLEVELONEUPGRADE_H
```

5.65 WaterSupplyLevelThreeUpgrade.h

```
00001 #ifndef WATERSUPPLYLEVELTHREEUPGRADE_H
00002 #define WATERSUPPLYLEVELTHREEUPGRADE_H
00003
00004 #include "WaterSupplyUpgrade.h"
00005
00013 class WaterSupplyLevelThreeUpgrade : public WaterSupplyUpgrade
00014 {
00015 public:
00023     WaterSupplyLevelThreeUpgrade(WaterSupply *water);
00024
00033     WaterSupplyLevelThreeUpgrade(WaterSupplyLevelThreeUpgrade *wSLTU);
00034
00040     ~WaterSupplyLevelThreeUpgrade();
00041
```

```
00047     void update() override;
00048
00056     Entity *clone() override;
00057
00062     Entity *upgrade() override;
00063
00071     int getOutput() override;
00072
00077     Cost getCost() override;
00078
00084     int getLevel() override;
00085
00086 private:
00087     const int UPGRADE = 6;
00088 };
00089
00090 #endif // WATERSUPPLYLEVELTHREEUPGRADE_H
```

5.66 WaterSupplyLevelTwoUpgrade.h

```
00001 #ifndef WATERSUPPLYLEVELTWOUPGRADE_H
00002 #define WATERSUPPLYLEVELTWOUPGRADE_H
00003
00004 #include "WaterSupplyUpgrade.h"
00005
00006 class WaterSupplyLevelThreeUpgrade;
00007
00015 class WaterSupplyLevelTwoUpgrade : public WaterSupplyUpgrade
00016 {
00017 public:
00025     WaterSupplyLevelTwoUpgrade(WaterSupply *water);
00026
00034     WaterSupplyLevelTwoUpgrade(WaterSupplyLevelTwoUpgrade *wSLTU);
00035
00039     ~WaterSupplyLevelTwoUpgrade();
00040
00044     void update() override;
00045
00051     Entity *clone() override;
00052
00057     Entity *upgrade() override;
00058
00066     int getOutput() override;
00067
00072     Cost getCost() override;
00073
00079     int getLevel() override;
00080
00081 private:
00082     const int UPGRADE = 4;
00083 };
00084
00085 #endif // WATERSUPPLYLEVELTWOUPGRADE_H
```

5.67 WaterSupplyUpgrade.h

```
00001 #ifndef WATERSUPPLYUPGRADE_H
00002 #define WATERSUPPLYUPGRADE_H
00003
00004 #include "entities/utility/watersupply/WaterSupply.h"
00005
00013 class WaterSupplyUpgrade : public WaterSupply
00014 {
00015 public:
00023     WaterSupplyUpgrade(WaterSupply *water);
00024
00032     WaterSupplyUpgrade(WaterSupplyUpgrade *wSU);
00033
00037     virtual ~WaterSupplyUpgrade();
00038
00042     virtual void update() = 0;
00043
00049     virtual Entity *clone() = 0;
00050
00055     virtual Entity *upgrade() = 0;
00056
00062     virtual int getOutput() = 0;
00063
00068     virtual Cost getCost() = 0;
```

```
00069
00070 protected:
00071     WaterSupply *waterSupply;
00072 };
00073
00074 #endif // WATERSUPPLYUPGRADE_H
```

5.68 EntityFactory.h

```
00001 #ifndef ENTITYFACTORY_H
00002 #define ENTITYFACTORY_H
00003
00004 #include "entities/base/Entity.h"
00005 #include "utils/EntityType.h"
00006 #include "utils/ConfigManager.h"
00007 #include "utils/Size.h"
00008
00020 class EntityFactory
00021 {
00022 public:
00026     EntityFactory();
00027
00031     virtual ~EntityFactory();
00032
00045     virtual Entity* createEntity(EntityType type, Size size, int xPos, int yPos) = 0;
00046 };
00047
00048 #endif // ENTITYFACTORY_H
```

5.69 AmenityFactory.h

```
00001 #ifndef AMENITYFACTORY_H
00002 #define AMENITYFACTORY_H
00003
00004 #include "factory/base/EntityFactory.h"
00005 #include "entities/building/amenity/Park.h"
00006 #include "entities/building/amenity/Theater.h"
00007 #include "entities/building/amenity/Monument.h"
00008
00016 class AmenityFactory : public EntityFactory {
00017 public:
00021     AmenityFactory();
00022
00026     ~AmenityFactory();
00027
00037     virtual Entity* createEntity(EntityType type, Size size, int xPos, int yPos);
00038
00039 private:
00048     Entity* createPark(Size size, int xPos, int yPos);
00049
00058     Entity* createTheater(Size size, int xPos, int yPos);
00059
00068     Entity* createMonument(Size size, int xPos, int yPos);
00069 };
00070
00071 #endif // AMENITYFACTORY_H
```

5.70 EconomicBuildingFactory.h

```
00001 #ifndef ECONOMICBUILDINGFACTORY_H
00002 #define ECONOMICBUILDINGFACTORY_H
00003
00004 #include "factory/base/EntityFactory.h"
00005 #include "entities/building/economic/Factory.h"
00006 #include "entities/building/economic/ShoppingMall.h"
00007 #include "entities/building/economic/Office.h"
00008
00016 class EconomicBuildingFactory : public EntityFactory {
00017 public:
00021     EconomicBuildingFactory();
00022
00026     ~EconomicBuildingFactory();
00027
00037     virtual Entity* createEntity(EntityType type, Size size, int xPos, int yPos);
00038
```

```
00039 private:
00048     Entity* createFactory(Size size, int xPos, int yPos);
00049
00058     Entity* createShoppingMall(Size size, int xPos, int yPos);
00059
00068     Entity* createOffice(Size size, int xPos, int yPos);
00069 };
00070
00071 #endif // ECONOMICBUILDINGFACTORY_H
```

5.71 ResidentialBuildingFactory.h

```
00001 #ifndef RESIDENTIALBUILDINGFACTORY_H
00002 #define RESIDENTIALBUILDINGFACTORY_H
00003
00004 #include "factory/base/EntityFactory.h"
00005 #include "entities/building/residential/House.h"
00006 #include "entities/building/residential/Apartment.h"
00007
00015 class ResidentialBuildingFactory : public EntityFactory {
00016 public:
00020     ResidentialBuildingFactory();
00021
00025     ~ResidentialBuildingFactory();
00026
00036     virtual Entity* createEntity(EntityType type, Size size, int xPos, int yPos);
00037
00038 private:
00047     Entity* createHouse(Size size, int xPos, int yPos);
00048
00057     Entity* createApartment(Size size, int xPos, int yPos);
00058 };
00059
00060 #endif // RESIDENTIALBUILDINGFACTORY_H
```

5.72 ServiceBuildingFactory.h

```
00001 #ifndef SERVICEBUILDINGFACTORY_H
00002 #define SERVICEBUILDINGFACTORY_H
00003
00004 #include "entities/building/service/Hospital.h"
00005 #include "entities/building/service/PoliceStation.h"
00006 #include "entities/building/service/School.h"
00007 #include "factory/base/EntityFactory.h"
00008
00016 class ServiceBuildingFactory : public EntityFactory {
00017    public:
00021     ServiceBuildingFactory();
00022
00026     ~ServiceBuildingFactory();
00027
00037     virtual Entity* createEntity(EntityType type, Size size, int xPos, int yPos) override;
00038
00039    private:
00048     Entity* createHospital(Size size, int xPos, int yPos);
00049
00058     Entity* createPoliceStation(Size size, int xPos, int yPos);
00059
00068     Entity* createSchool(Size size, int xPos, int yPos);
00069 };
00070
00071 #endif // SERVICEBUILDINGFACTORY_H
```

5.73 IndustryFactory.h

```
00001 #ifndef INDUSTRYFACTORY_H
00002 #define INDUSTRYFACTORY_H
00003
00004 #include "entities/industry/concreteproducer/ConcreteProducer.h"
00005 #include "entities/industry/stoneproducer/StoneProducer.h"
00006 #include "entities/industry/woodproducer/WoodProducer.h"
00007 #include "factory/base/EntityFactory.h"
00008
00016 class IndustryFactory : public EntityFactory {
00017    public:
```

```
00021     IndustryFactory();
00022
00026     ~IndustryFactory();
00027
00037     virtual Entity* createEntity(EntityType type, Size size, int xPos, int yPos);
00038
00039    private:
00048     Entity* createConcreteProducer(Size size, int xPos, int yPos);
00049
00058     Entity* createStoneProducer(Size size, int xPos, int yPos);
00059
00068     Entity* createWoodProducer(Size size, int xPos, int yPos);
00069 };
00070
00071 #endif // INDUSTRYFACTORY_H
```

5.74 TransportFactory.h

```
00001 #ifndef TRANSPORTFACTORY_H
00002 #define TRANSPORTFACTORY_H
00003
00004 #include "entities/transport/Airport.h"
00005 #include "entities/transport/BusStop.h"
00006 #include "entities/transport/TrainStation.h"
00007 #include "factory/base/EntityFactory.h"
00008
00016 class TransportFactory : public EntityFactory {
00017    public:
00021     TransportFactory();
00022
00026     ~TransportFactory();
00027
00037     virtual Entity* createEntity(EntityType type, Size size, int xPos, int yPos);
00038
00039    private:
00048     Entity* createBusStop(Size size, int xPos, int yPos);
00049
00058     Entity* createTrainStation(Size size, int xPos, int yPos);
00059
00068     Entity* createAirport(Size size, int xPos, int yPos);
00069 };
00070
00071 #endif // TRANSPORTFACTORY_H
```

5.75 UtilityFactory.h

```
00001 #ifndef UTILITYFACTORY_H
00002 #define UTILITYFACTORY_H
00003
00004 #include "factory/base/EntityFactory.h"
00005 #include "entities/base/Entity.h"
00006 #include "entities/utility/powerplant/PowerPlant.h"
00007 #include "entities/utility/watersupply/WaterSupply.h"
00008 #include "entities/utility/wastemanagement/WasteManagement.h"
00009 #include "entities/utility/sewagesystem/SewageSystem.h"
00010
00019 class UtilityFactory : public EntityFactory {
00020 public:
00024     UtilityFactory();
00025
00029     ~UtilityFactory();
00030
00040     virtual Entity* createEntity(EntityType type, Size size, int xPos, int yPos);
00041
00042 private:
00051     Entity* createPowerPlant(Size size, int xPos, int yPos);
00052
00061     Entity* createWaterSupply(Size size, int xPos, int yPos);
00062
00071     Entity* createWasteManagement(Size size, int xPos, int yPos);
00072
00081     Entity* createSewageSystem(Size size, int xPos, int yPos);
00082 };
00083
00084 #endif // UTILITYFACTORY_H
```

5.76 Iterator.h

```
00001 #ifndef ITERATOR_H
00002 #define ITERATOR_H
00003
00004 #include <vector>
00005 #include <unordered_set>
00006 #include "entities/base/Entity.h"
00007
00008 class Iterator
00009 {
00010 protected:
00011     std::vector<std::vector<Entity *» grid;
00012     std::vector<std::vector<Entity *»::iterator currRow;
00013     std::vector<Entity *>::iterator curr;
00014     int row;
00015     int col;
00016     std::unordered_set<Entity *> visitedEntities; // Tracks visited entities
00017
00018 public:
00019     Iterator();
00020     virtual ~Iterator();
00021
00022     Iterator(std::vector<std::vector<Entity *» &grid) : grid(grid) {};
00023     virtual void first() = 0;
00024     virtual void next() = 0;
00025     virtual bool hasNext() = 0;
00026     virtual Entity *current() = 0;
00027     virtual int getRow();
00028     virtual int getCol();
00029
00030 protected:
00031     bool isVisited(Entity *entity);
00032     void markVisited(Entity *entity);
00033 };
00034
00035 #endif // ITERATOR_H
```

5.77 AmenityIterator.h

```
00001 #ifndef AMENITYITERATOR_H
00002 #define AMENITYITERATOR_H
00003
00004 #include "iterators/base/Iterator.h"
00005 #include "entities/building/amenity/Amenity.h"
00006
00007 class AmenityIterator : public Iterator
00008 {
00009 public:
00010     AmenityIterator();
00011     ~AmenityIterator();
00012
00013     AmenityIterator(std::vector<std::vector<Entity *» &grid);
00014     void first() override;
00015     void next() override;
00016     bool hasNext() override;
00017     Entity *current() override;
00018 };
00019
00020 #endif // AMENITYITERATOR_H
```

5.78 BuildingIterator.h

```
00001 #ifndef BUILDINGITERATOR_H
00002 #define BUILDINGITERATOR_H
00003
00004 #include "iterators/base/Iterator.h"
00005 #include "entities/building/base/Building.h"
00006
00007 class BuildingIterator : public Iterator
00008 {
00009 public:
00010     BuildingIterator();
00011     ~BuildingIterator();
00012
00013     BuildingIterator(std::vector<std::vector<Entity *» &grid);
00014     void first();
00015     void next();
00016     bool hasNext();
```

```
00017     Entity *current();
00018 };
00019
00020 #endif // BUILDINGITERATOR_H
```

5.79 EconomicBuildingIterator.h

```
00001 #ifndef ECONOMICBUILDINGITERATOR_H
00002 #define ECONOMICBUILDINGITERATOR_H
00003
00004 #include "iterators/base/Iterator.h"
00005 #include "entities/building/economic/EconomicBuilding.h"
00006
00007 class EconomicBuildingIterator : public Iterator
00008 {
00009 public:
00010     EconomicBuildingIterator();
00011     ~EconomicBuildingIterator();
00012
00013     EconomicBuildingIterator(std::vector<std::vector<Entity *» &grid);
00014     void first();
00015     void next();
00016     bool hasNext();
00017     Entity *current();
00018 };
00019
00020 #endif // ECONOMICBUILDINGITERATOR_H
```

5.80 ResidentialBuildingIterator.h

```
00001 #ifndef RESIDENTIALBUILDINGITERATOR_H
00002 #define RESIDENTIALBUILDINGITERATOR_H
00003
00004 #include "iterators/base/Iterator.h"
00005 #include "entities/building/residential/ResidentialBuilding.h"
00006
00007 class ResidentialBuildingIterator : public Iterator
00008 {
00009 public:
00010     ResidentialBuildingIterator();
00011     ~ResidentialBuildingIterator();
00012
00013     ResidentialBuildingIterator(std::vector<std::vector<Entity *» &grid);
00014     void first();
00015     void next();
00016     bool hasNext();
00017     Entity *current();
00018 };
00019
00020 #endif // RESIDENTIALBUILDINGITERATOR_H
```

5.81 ServiceBuildingIterator.h

```
00001 #ifndef SERVICEBUILDINGITERATOR_H
00002 #define SERVICEBUILDINGITERATOR_H
00003
00004 #include "iterators/base/Iterator.h"
00005 #include "entities/building/service/ServiceBuilding.h"
00006
00007 class ServiceBuildingIterator : public Iterator
00008 {
00009 public:
00010     ServiceBuildingIterator();
00011     ~ServiceBuildingIterator();
00012
00013     ServiceBuildingIterator(std::vector<std::vector<Entity *» &grid);
00014     void first();
00015     void next();
00016     bool hasNext();
00017     Entity *current();
00018 };
00019
00020 #endif // SERVICEBUILDINGITERATOR_H
```

5.82 CityIterator.h

```
00001 #ifndef CITYITERATOR_H
00002 #define CITYITERATOR_H
00003
00004 #include "iterators/base/Iterator.h"
00005 #include "city/City.h"
00006
00007 class CityIterator : public Iterator
00008 {
00009 private:
00010     bool unique;
00011
00012 public:
00013     CityIterator();
00014     ~CityIterator();
00015
00016     // Constructor with unique iteration option
00017     CityIterator(std::vector<std::vector<Entity *» &grid, bool unique = true);
00018
00019     void first() override;
00020     void next() override;
00021     bool hasNext() override;
00022     Entity *current() override;
00023 };
00024
00025 #endif // CITYITERATOR_H
```

5.83 ConcreteProducerIterator.h

```
00001 #ifndef CONCRETEPRODUCERITERATOR_H
00002 #define CONCRETEPRODUCERITERATOR_H
00003
00004 #include "iterators/base/Iterator.h"
00005 #include "entities/industry/concreteproducer/ConcreteProducer.h"
00006
00007 class ConcreteProducerIterator : public Iterator
00008 {
00009 public:
00010     ConcreteProducerIterator();
00011     ~ConcreteProducerIterator();
00012
00013     ConcreteProducerIterator(std::vector<std::vector<Entity *» &grid);
00014     void first();
00015     void next();
00016     bool hasNext();
00017     Entity *current();
00018 };
00019
00020 #endif // CONCRETEPRODUCERITERATOR_H
```

5.84 IndustryIterator.h

```
00001 #ifndef INDUSTRYITERATOR_H
00002 #define INDUSTRYITERATOR_H
00003
00004 #include "iterators/base/Iterator.h"
00005 #include "entities/industry/base/Industry.h"
00006
00007 class IndustryIterator : public Iterator
00008 {
00009 public:
00010     IndustryIterator();
00011     ~IndustryIterator();
00012
00013     IndustryIterator(std::vector<std::vector<Entity *» &grid);
00014     void first();
00015     void next();
00016     bool hasNext();
00017     Entity *current();
00018 };
00019
00020 #endif // INDUSTRYITERATOR_H
```

5.85 StoneProducerIterator.h

```
00001 #ifndef STONEPRODUCERITERATOR_H
```

```
00002 #define STONEPRODUCERITERATOR_H
00003
00004 #include "iterators/base/Iterator.h"
00005 #include "entities/industry/stoneproducer/StoneProducer.h"
00006
00007 class StoneProducerIterator : public Iterator
00008 {
00009 public:
00010     StoneProducerIterator();
00011     ~StoneProducerIterator();
00012
00013     StoneProducerIterator(std::vector<std::vector<Entity *» &grid);
00014     void first();
00015     void next();
00016     bool hasNext();
00017     Entity *current();
00018 };
00019
00020 #endif // STONEPRODUCERITERATOR_H
```

5.86 WoodProducerIterator.h

```
00001 #ifndef WOODPRODUCERITERATOR_H
00002 #define WOODPRODUCERITERATOR_H
00003
00004 #include "iterators/base/Iterator.h"
00005 #include "entities/industry/woodproducer/WoodProducer.h"
00006
00007 class WoodProducerIterator : public Iterator
00008 {
00009 public:
00010     WoodProducerIterator();
00011     ~WoodProducerIterator();
00012
00013     WoodProducerIterator(std::vector<std::vector<Entity *» &grid);
00014     void first();
00015     void next();
00016     bool hasNext();
00017     Entity *current();
00018 };
00019
00020 #endif // WOODPRODUCERITERATOR_H
```

5.87 RoadIterator.h

```
00001 #ifndef ROADITERATOR_H
00002 #define ROADITERATOR_H
00003
00004 #include "iterators/base/Iterator.h"
00005 #include "entities/road/Road.h"
00006
00007 class RoadIterator : public Iterator
00008 {
00009 public:
00010     RoadIterator();
00011     ~RoadIterator();
00012
00013     RoadIterator(std::vector<std::vector<Entity *» &grid);
00014     void first();
00015     void next();
00016     bool hasNext();
00017     Entity *current();
00018 };
00019
00020 #endif // ROADITERATOR_H
```

5.88 TransportIterator.h

```
00001 #ifndef TRANSPORTITERATOR_H
00002 #define TRANSPORTITERATOR_H
00003
00004 #include "iterators/base/Iterator.h"
00005 #include "entities/transport/Transport.h"
00006
00007 class TransportIterator : public Iterator
```

```
00008 {
00009 public:
00010     TransportIterator();
00011     ~TransportIterator();
00012
00013     TransportIterator(std::vector<std::vector<Entity *» &grid);
00014     void first();
00015     void next();
00016     bool hasNext();
00017     Entity *current();
00018 };
00019
00020 #endif // TRANSPORTITERATOR_H
```

5.89 PowerPlantIterator.h

```
00001 #ifndef POWERPLANTITERATOR_H
00002 #define POWERPLANTITERATOR_H
00003
00004 #include "iterators/base/Iterator.h"
00005 #include "entities/utility/powerplant/PowerPlant.h"
00006
00007 class PowerPlantIterator : public Iterator
00008 {
00009 public:
00010     PowerPlantIterator();
00011     ~PowerPlantIterator();
00012
00013     PowerPlantIterator(std::vector<std::vector<Entity *» &grid);
00014     void first();
00015     void next();
00016     bool hasNext();
00017     Entity *current();
00018 };
00019
00020 #endif // POWERPLANTITERATOR_H
```

5.90 SewageSystemIterator.h

```
00001 #ifndef SEWAGESYSTEMITERATOR_H
00002 #define SEWAGESYSTEMITERATOR_H
00003
00004 #include "iterators/base/Iterator.h"
00005 #include "entities/utility/sewagesystem/SewageSystem.h"
00006
00007 class SewageSystemIterator : public Iterator
00008 {
00009 public:
00010     SewageSystemIterator();
00011     ~SewageSystemIterator();
00012
00013     SewageSystemIterator(std::vector<std::vector<Entity *» &grid);
00014     void first();
00015     void next();
00016     bool hasNext();
00017     Entity *current();
00018 };
00019
00020 #endif // SEWAGESYSTEMITERATOR_H
```

5.91 UtilityIterator.h

```
00001 #ifndef UTILITYITERATOR_H
00002 #define UTILITYITERATOR_H
00003
00004 #include "iterators/base/Iterator.h"
00005 #include "entities/utility/base/Utility.h"
00006
00007 class UtilityIterator : public Iterator
00008 {
00009 public:
00010     UtilityIterator();
00011     ~UtilityIterator();
00012
00013     UtilityIterator(std::vector<std::vector<Entity *» &grid);
```

```
00014     void first();
00015     void next();
00016     bool hasNext();
00017     Entity *current();
00018 };
00019
00020 #endif // UTILITYITERATOR_H
```

5.92 WasteManagementIterator.h

```
00001 #ifndef WASTEMANAGEMENTITERATOR_H
00002 #define WASTEMANAGEMENTITERATOR_H
00003
00004 #include "iterators/base/Iterator.h"
00005 #include "entities/utility/wastemanagement/WasteManagement.h"
00006
00007 class WasteManagementIterator : public Iterator
00008 {
00009 public:
00010     WasteManagementIterator();
00011     ~WasteManagementIterator();
00012
00013     WasteManagementIterator(std::vector<std::vector<Entity *> &grid);
00014     void first();
00015     void next();
00016     bool hasNext();
00017     Entity *current();
00018 };
00019
00020 #endif // WASTEMANAGEMENTITERATOR_H
```

5.93 WaterSupplyIterator.h

```
00001 #ifndef WATERSUPPLYITERATOR_H
00002 #define WATERSUPPLYITERATOR_H
00003
00004 #include "iterators/base/Iterator.h"
00005 #include "entities/utility/watersupply/WaterSupply.h"
00006
00007 class WaterSupplyIterator : public Iterator
00008 {
00009 public:
00010     WaterSupplyIterator();
00011     ~WaterSupplyIterator();
00012
00013     WaterSupplyIterator(std::vector<std::vector<Entity *> &grid);
00014     void first();
00015     void next();
00016     bool hasNext();
00017     Entity *current();
00018 };
00019
00020 #endif // WATERSUPPLYITERATOR_H
```

5.94 src/managers/AmenityManager.h File Reference

Manages the creation and handling of amenities within the application.

```
#include "entities/building/amenity/Amenity.h"
#include "factory/building/AmenityFactory.h"
#include "utils/EntityType.h"
#include "utils/Size.h"
#include "city/City.h"
```

Classes

- class AmenityManager

  Responsible for managing amenities by creating and configuring them based on specified types and sizes.

5.94.1 Detailed Description

Manages the creation and handling of amenities within the application.

5.95 AmenityManager.h

Go to the documentation of this file.

```
00001
00006 #ifndef AMENITYMANAGER_H
00007 #define AMENITYMANAGER_H
00008
00009 #include "entities/building/amenity/Amenity.h"
00010 #include "factory/building/AmenityFactory.h"
00011 #include "utils/EntityType.h"
00012 #include "utils/Size.h"
00013 #include "city/City.h"
00014
00020 class AmenityManager
00021 {
00022 public:
00026     AmenityManager();
00027
00031     ~AmenityManager();
00032
00042     void buildAmenity(EntityType type, Size size, int xPos, int yPos);
00043 };
00044
00045 #endif // AMENITYMANAGER_H
```

5.96 src/managers/BuildingManager.h File Reference

Header file for the BuildingManager class.

```
#include <string>
#include "utils/ConfigManager.h"
```

Classes

- class BuildingManager

  Manages the construction of residential buildings in the city.

5.96.1 Detailed Description

Header file for the BuildingManager class.

5.97 BuildingManager.h

Go to the documentation of this file.

```
00001
00006 #ifndef BUILDINGMANAGER_H
00007 #define BUILDINGMANAGER_H
00008
00009 #include <string>
00010 #include "utils/ConfigManager.h"
00011
00016 class BuildingManager
00017 {
00018 public:
00022     BuildingManager();
00023
00027     ~BuildingManager();
00028
00039     bool buildBuilding(EntityType type, Size size, int x, int y);
00040 };
00041
00042 #endif // BUILDINGMANAGER_H
```

5.98 src/managers/CityManager.h File Reference

Manages city operations, including initialization, updating entities, and managing building purchases and sales.

```
#include "entities/base/Entity.h"
#include <vector>
#include "city/City.h"
```

Classes

- class CityManager

  Manages and maintains city entities and provides functions for updating city states.

5.98.1 Detailed Description

Manages city operations, including initialization, updating entities, and managing building purchases and sales.

5.99 CityManager.h

Go to the documentation of this file.

```
00001
00006 #ifndef CITYMANAGER_H
00007 #define CITYMANAGER_H
00008
00009 #include "entities/base/Entity.h"
00010 #include <vector>
00011 #include "city/City.h"
00012
00017 class CityManager
00018 {
00019 public:
00023     CityManager();
00024
00028     ~CityManager();
00029
00033     void initializeCity();
00034
00038     void updateCity();
00039
00046     Entity *getEntity(int x, int y);
00047
00053     void sellBuilding(int xPos, int yPos);
00054
00059     void sellAllBuildingsOfType(EntityType type);
00060
00067     std::vector<std::vector<int» getAvailiablePositions(EntityType type, Size size);
00068
00075     bool canAffordToBuy(EntityType type, Size size);
00076
00085     bool canBuyAt(int xPos, int yPos, EntityType type, Size size);
00086
00093     bool buyEntity(EntityType type, Size size);
00094
00104     void generateCity(std::optional<unsigned int> seed = std::nullopt);
00105
00118     void generateRandomRoads(int gridWidth, int gridHeight, int minWidth, int minHeight, int roadGap);
00119
00128     void generateRandomBuildings(int placementProbability);
00129 };
00130
00131 #endif // CITYMANAGER_H
```

5.100 GovernmentManager.h

```
00001 #ifndef GOVERNMENTMANAGER_H
00002 #define GOVERNMENTMANAGER_H
00003
00004 #include <vector>
00005 #include "utils/Caretaker.h"
00006 #include "utils/PolicyType.h"
00007
00015 class GovernmentManager {
00016 public:
00020     GovernmentManager();
00021
00025     ~GovernmentManager();
00026
00031     void setResidentialTaxRate(float rate);
00032
00037     void setEconomicTaxRate(float rate);
00038
00043     int getResidentialTax();
00044
00049     int getEconomicTax();
00050
00055     int getResidentialTaxRate();
00056
00061     int getEconomicTaxRate();
00062
00067     void enactWaterUsagePolicy(PolicyType policy);
00068
00073     void enactElectricityPolicy(PolicyType policy);
00074
00079     std::vector<Memento*> getAllPastPolicies();
00080
00081 private:
00082     Caretaker* caretaker;
00083 };
00084
00085 #endif // GOVERNMENTMANAGER_H
```

5.101 src/managers/PopulationManager.h File Reference

Manages population growth, capacity, and satisfaction for the City.

```
#include "visitors/population/PopulationVisitor.h"
#include "visitors/satisfaction/SatisfactionVisitor.h"
#include "visitors/utility/UtilityVisitor.h"
#include "city/City.h"
```

Classes

- class PopulationManager

  Responsible for managing the population growth, decrease, capacity calculations, and satisfaction levels within a City.

5.101.1 Detailed Description

Manages population growth, capacity, and satisfaction for the City.

5.102 PopulationManager.h

Go to the documentation of this file.

```
00001
00006 #ifndef POPULATIONMANAGER_H
00007 #define POPULATIONMANAGER_H
00008
00009 #include "visitors/population/PopulationVisitor.h"
00010 #include "visitors/satisfaction/SatisfactionVisitor.h"
00011 #include "visitors/utility/UtilityVisitor.h"
00012 #include "city/City.h"
00013
00019 class PopulationManager
00020 {
00021 private:
00022     int minimumIncrease;
00023     int maximumIncrease;
00025 public:
00031     PopulationManager(int minimumIncrease, int maximumIncrease);
00032
00036     ~PopulationManager();
00037
00041     void calculatePopulationCapacity();
00042
00047     void growPopulation();
00048
00053     void decreasePopulation();
00054
00059     void calculateSatisfaction();
00060 };
00061
00062 #endif // POPULATIONMANAGER_H
```

5.103 ResourceManager.h

```
00001 #ifndef RESOURCEMANAGER_H
00002 #define RESOURCEMANAGER_H
00003
00004 #include "entities/industry/base/Industry.h"
00005 #include "iterators/industry/ConcreteProducerIterator.h"
00006 #include "iterators/industry/StoneProducerIterator.h"
00007 #include "iterators/industry/IndustryIterator.h"
00008 #include "iterators/industry/WoodProducerIterator.h"
00009 #include "city/City.h"
00010 #include "factory/industry/IndustryFactory.h"
00011 #include "visitors/resource/ResourceVisitor.h"
00012 #include "visitors/tax/TaxCalculationVisitor.h"
00013
00014 class ResourceManager
00015 {
00016 public:
00018     ResourceManager();
00019
00021     ~ResourceManager();
00022
00030     void buildIndustry(EntityType type, Size size, int x, int y);
00031
00033     int calculateMoneyMade();
00034
00036     int calculateWoodMade();
00037
00039     int calculateStoneMade();
00040
00042     int calculateConcreteMade();
00043
00045     std::vector<Industry*> getAllIndustryBuildings();
00046
00048     std::vector<Industry*> getAllConcreteProducers();
00049
00051     std::vector<Industry*> getAllStoneProducers();
00052
00054     std::vector<Industry*> getAllWoodProducers();
00055
00061     bool canAffordUpgrade(Industry* industry);
00062
00068     bool upgrade(Industry*& industry);
00069 };
00070
00071 #endif // RESOURCEMANAGER_H
```

5.104 ServiceManager.h

```
00001 #ifndef SERVICEMANAGER_H
00002 #define SERVICEMANAGER_H
00003
00004 #include "utils/EntityType.h"
00005 #include "utils/Size.h"
00006 #include "entities/building/service/ServiceBuilding.h"
00007 #include "factory/building/ServiceBuildingFactory.h"
00008 #include "city/City.h"
00034 class ServiceManager
00035 {
00036 public:
00040     ServiceManager();
00041
00045     ~ServiceManager();
00046
00054     bool buildService(EntityType type, Size size, int xPos, int yPos);
00055 };
00056
00057 #endif // SERVICEMANAGER_H
```

5.105 TransportManager.h

```
00001 #ifndef TRANSPORTMANAGER_H
00002 #define TRANSPORTMANAGER_H
00003 #include "utils/EntityType.h"
00004 #include "utils/Size.h"
00005 #include "entities//road/Road.h"
00006 #include "factory/transport/TransportFactory.h"
00007
00015 class TransportManager
00016 {
00017 public:
00021     TransportManager();
00022
00026     ~TransportManager();
00027
00033     bool canAffordRoad();
00034
00042     bool buildRoad(int x, int y);
00043
00053     bool buildPublicTransit(EntityType type, Size size, int x, int y);
00061     bool canAffordPublicTransit(EntityType type, Size size);
00062 };
00063
00064 #endif // TRANSPORTMANAGER_H
```

5.106 src/managers/UtilityManager.h File Reference

Manages utility creation, upgrades, and statistics within the city.

```
#include "utils/EntityType.h"
#include "utils/Size.h"
#include "entities/utility/base/Utility.h"
#include "city/City.h"
#include "factory/utility/UtilityFactory.h"
#include "iterators/utility/UtilityIterator.h"
#include "iterators/utility/WaterSupplyIterator.h"
#include "iterators/utility/PowerPlantIterator.h"
#include "iterators/utility/WasteManagementIterator.h"
#include "iterators/utility/SewageSystemIterator.h"
#include <vector>
```

Classes

- class UtilityManager

  Responsible for creating and managing utilities within the city, handling utility upgrades, and gathering utility-related statistics.

5.106.1 Detailed Description

Manages utility creation, upgrades, and statistics within the city.

5.107 UtilityManager.h

Go to the documentation of this file.

```
00001
00006 #ifndef UTILITYMANAGER_H
00007 #define UTILITYMANAGER_H
00008
00009 #include "utils/EntityType.h"
00010 #include "utils/Size.h"
00011 #include "entities/utility/base/Utility.h"
00012 #include "city/City.h"
00013 #include "factory/utility/UtilityFactory.h"
00014 #include "iterators/utility/UtilityIterator.h"
00015 #include "iterators/utility/WaterSupplyIterator.h"
00016 #include "iterators/utility/PowerPlantIterator.h"
00017 #include "iterators/utility/WasteManagementIterator.h"
00018 #include "iterators/utility/SewageSystemIterator.h"
00019 #include <vector>
00020
00026 class UtilityManager
00027 {
00028 public:
00032     UtilityManager();
00033
00037     ~UtilityManager();
00038
00046     bool buildUtility(EntityType type, Size size, int x, int y);
00047
00052     int getElectricityProduction();
00053
00058     int getElectricityConsumption();
00059
00064     int getWaterProduction();
00065
00070     int getWaterConsumption();
00071
00076     int getWasteProduction();
00077
00082     int getWasteConsumption();
00083
00088     int getSewageProduction();
00089
00094     int getSewageConsumption();
00095
00100     std::vector<Utility*> getAllUtilities();
00101
00106     std::vector<Utility*> getAllWaterSupplies();
00107
00112     std::vector<Utility*> getAllPowerPlants();
00113
00118     std::vector<Utility*> getAllWasteManagements();
00119
00124     std::vector<Utility*> getAllSewageSystems();
00125
00131     bool canAffordUpgrade(Utility* utility);
00132
00138     bool upgrade(Utility*& utility);
00139 };
00140
00141 #endif // UTILITYMANAGER_H
```

5.108 src/menus/base/BuyMenu.h File Reference

Defines the BuyMenu class, an abstract menu for handling building purchases.

```
#include "menus/base/IMenu.h"
#include "utils/EntityType.h"
#include "utils/Size.h"
```

Classes

- class BuyMenu

  Abstract class representing the Buy Menu in the game.

5.108.1 Detailed Description

Defines the BuyMenu class, an abstract menu for handling building purchases.

5.109 BuyMenu.h

Go to the documentation of this file.

```
00001
00006 #ifndef BUYMENU_H
00007 #define BUYMENU_H
00008
00009 #include "menus/base/IMenu.h"
00010 #include "utils/EntityType.h"
00011 #include "utils/Size.h"
00012
00020 class BuyMenu : public IMenu
00021 {
00022 public:
00028     BuyMenu();
00029
00035     virtual ~BuyMenu();
00036
00042     void display() const override;
00043
00050     void handleInput() override;
00051
00052 protected:
00053     EntityType selectedType;
00054     Size selectedSize;
00055
00062     virtual EntityType chooseEntityType() = 0;
00063
00072     Size chooseBuildingSize(EntityType type);
00073
00084     void chooseBuildingPosition(int &xPos, int &yPos, EntityType type, Size size);
00085
00096     void confirmPurchase(EntityType type, Size size, int xPos, int yPos);
00097
00108     virtual void buildEntity(EntityType type, Size size, int xPos, int yPos) = 0;
00109
00116     void displayAvailablePositions(const std::vector<std::vector<int» &positions) const override;
00117 };
00118
00119 #endif // BUYMENU_H
```

5.110 IMenu.h

```
00001 #ifndef IMENU_H
00002 #define IMENU_H
00003
00004 #include <string>
00005 #include <vector>
00006 #include <iostream>
00007 #include <algorithm>
00008 #include <limits>
00009 #include <variant>
00010 #include "managers/CityManager.h"
00011
00015 struct Option
00016 {
00017     std::variant<char, int> key;
00018     std::string icon;
00019     std::string text;
00020 };
00021
```

```
00025 struct Section
00026 {
00027     std::string heading;
00028     std::vector<Option> options;
00029 };
00030
00037 class IMenu
00038 {
00039 public:
00043     IMenu() = default;
00044
00049     IMenu(std::string heading);
00050
00054     virtual ~IMenu() = default;
00055
00061     virtual void display() const = 0;
00062
00068     virtual void handleInput() = 0;
00069
00074     void setHeading(const std::string &heading);
00075
00076 protected:
00077     std::vector<Section> sections;
00078     std::string menuHeading;
00079     bool hasExited;
00080     CityManager cityManager;
00081
00082     bool displayResources;
00083     bool isInfoMenu;
00084
00085     // Utility functions and color constants for inherited classes.
00086
00090     static const char *RESET;
00091     static const char *BOLD_WHITE;
00092     static const char *NORMAL_WHITE;
00093     static const char *DARK_GRAY;
00094     static const char *BOLD_YELLOW;
00095     static const char *BOLD_GREEN;
00096     static const char *BOLD_RED;
00097     static const char *BOLD_CYAN;
00098     static const char *BLUE;
00099
00107     static char indexToExtendedChar(int index);
00108
00115     std::string repeat(const std::string &str, int times) const;
00116
00125     int calculateMaxWidth(const std::string &menuHeading, const std::vector<Section> &sections) const;
00126
00131     void printTopBorder(int width) const;
00132
00137     void printBottomBorder(int width) const;
00138
00143     void printSectionDivider(int width) const;
00144
00149     void printDoubleLineDivider(int width) const;
00150
00157     std::string centerText(const std::string &text, int width) const;
00158
00166     std::string centerTextWithChar(const std::string &text, int width, const std::string &padChar)
      const;
00167
00171     void displayMenu() const;
00172
00176     void displayChoicePrompt() const;
00177
00182     void displayChoiceMessagePrompt(const std::string &message) const;
00183
00187     void displayInvalidChoice() const;
00188
00193     void displayErrorMessage(const std::string &message) const;
00194
00199     void displaySuccessMessage(const std::string &message) const;
00200
00204     void displayPressEnterToContinue() const;
00205
00209     void clearScreen() const { system("clear"); }
00210
00216     std::string stripColorCodes(const std::string &input) const;
00217
00225     static std::string coordinatesToLabel(int x, int y);
00226
00233     virtual void displayAvailablePositions(const std::vector<std::vector<int» &positions) const;
00234 };
00235
00236 #endif // IMENU_H
```

5.111 src/menus/base/MenuManager.h File Reference

Defines the MenuManager class for handling different game menus and switching between them.

```
#include "IMenu.h"
#include "city/City.h"
#include "utils/Menu.h"
#include "menus/main/MainMenu.h"
#include "menus/policy/PolicyMenu.h"
#include "menus/upgrades/UpgradesMenu.h"
#include "menus/tax/TaxMenu.h"
#include "menus/buildings/BuildingsMenu.h"
#include "menus/main/DisplayCityMenu.h"
#include "menus/buildings/amenity/BuyAmenityMenu.h"
#include "menus/buildings/economic/BuyEconomicBuildingMenu.h"
#include "menus/buildings/residential/BuyResidentialBuildingMenu.h"
#include "menus/buildings/transport/BuyTransportMenu.h"
#include "menus/buildings/utility/BuyUtilityMenu.h"
#include "menus/buildings/resource/BuyResourceMenu.h"
#include "menus/buildings/service/BuyServiceMenu.h"
#include "menus/buildings/demolish/DemolishMenu.h"
#include "menus/stats/StatsMenu.h"
#include "menus/road/BuyRoadMenu.h"
#include <memory>
#include <unordered_map>
#include <string>
```

Classes

- class MenuManager

  Manages the different menus in the game and allows switching between them.

5.111.1 Detailed Description

Defines the MenuManager class for handling different game menus and switching between them.

5.112 MenuManager.h

Go to the documentation of this file.

```
00001
00006 #ifndef MENUMANAGER_H
00007 #define MENUMANAGER_H
00008
00009 #include "IMenu.h"
00010 #include "city/City.h"
00011 #include "utils/Menu.h"
00012 #include "menus/main/MainMenu.h"
00013 #include "menus/policy/PolicyMenu.h"
00014 #include "menus/upgrades/UpgradesMenu.h"
00015 #include "menus/tax/TaxMenu.h"
00016 #include "menus/buildings/BuildingsMenu.h"
00017 #include "menus/main/DisplayCityMenu.h"
00018 #include "menus/buildings/amenity/BuyAmenityMenu.h"
00019 #include "menus/buildings/economic/BuyEconomicBuildingMenu.h"
00020 #include "menus/buildings/residential/BuyResidentialBuildingMenu.h"
00021 #include "menus/buildings/transport/BuyTransportMenu.h"
```

```
00022 #include "menus/buildings/utility/BuyUtilityMenu.h"
00023 #include "menus/buildings/resource/BuyResourceMenu.h"
00024 #include "menus/buildings/service/BuyServiceMenu.h"
00025 #include "menus/buildings/demolish/DemolishMenu.h"
00026 #include "menus/stats/StatsMenu.h"
00027 #include "menus/road/BuyRoadMenu.h"
00028 #include <memory>
00029 #include <unordered_map>
00030 #include <string>
00031
00039 class MenuManager
00040 {
00041 public:
00048     static MenuManager &instance();
00049
00056     void setCurrentMenu(Menu menuType);
00057
00064     void setCurrentMenu(std::shared_ptr<IMenu> menu);
00065
00071     void displayCurrentMenu();
00072
00078     void handleCurrentMenuInput();
00079
00086     void setCity(City *city);
00087
00094     City *getCity() const;
00095
00101     void clearScreen() const;
00102
00103 private:
00109     MenuManager();
00110
00116     ~MenuManager();
00117
00118     std::shared_ptr<IMenu> currentMenu;
00119     City *city = nullptr;
00120
00121     std::unordered_map<Menu, std::shared_ptr<IMenu>> menus;
00122
00128     MenuManager(const MenuManager &) = delete;
00129
00135     MenuManager &operator=(const MenuManager &) = delete;
00136 };
00137
00138 #endif // MENUMANAGER_H
```

5.113 src/menus/buildings/amenity/BuyAmenityMenu.h File Reference

Declares the BuyAmenityMenu class for purchasing amenities within the game.

```
#include "menus/base/BuyMenu.h"
#include "managers/AmenityManager.h"
```

Classes

- class BuyAmenityMenu

  Provides a menu interface for purchasing various types of amenities.

5.113.1 Detailed Description

Declares the BuyAmenityMenu class for purchasing amenities within the game.

5.114 BuyAmenityMenu.h

Go to the documentation of this file.

```
00001
00006 #ifndef BUYAMENITYMENU_H
00007 #define BUYAMENITYMENU_H
00008
00009 #include "menus/base/BuyMenu.h"
00010 #include "managers/AmenityManager.h"
00011
00020 class BuyAmenityMenu : public BuyMenu
00021 {
00022 protected:
00030     EntityType chooseEntityType() override;
00031
00042     void buildEntity(EntityType type, Size size, int xPos, int yPos) override;
00043
00044 private:
00045     AmenityManager amenityManager;
00046 };
00047
00048 #endif // BUYAMENITYMENU_H
```

5.115 src/menus/buildings/BuildingsMenu.h File Reference

Declares the BuildingsMenu class for managing building-related options in the game.

```
#include <vector>
#include "menus/base/IMenu.h"
#include "utils/EntityType.h"
```

Classes

- class BuildingsMenu

  Provides a menu interface for managing buildings in the game.

5.115.1 Detailed Description

Declares the BuildingsMenu class for managing building-related options in the game.

5.116 BuildingsMenu.h

Go to the documentation of this file.

```
00001
00006 #ifndef BUILDINGSMENU_H
00007 #define BUILDINGSMENU_H
00008
00009 #include <vector>
00010 #include "menus/base/IMenu.h"
00011 #include "utils/EntityType.h"
00012
00021 class BuildingsMenu : public IMenu
00022 {
00023 public:
00030     BuildingsMenu();
00031
00037     ~BuildingsMenu();
00038
00045     void display() const override;
00046
00054     void handleInput() override;
00055 };
00056
00057 #endif // BUILDINGSMENU_H
```

5.117 src/menus/buildings/BuildingsStatMenu.h File Reference

Declares the BuildingsStatMenu class for displaying statistics of various building types.

```
#include "menus/base/IMenu.h"
#include "utils/EntityType.h"
#include <vector>
```

Classes

- class BuildingsStatMenu

  Provides a menu interface for displaying detailed statistics of various building types.

5.117.1 Detailed Description

Declares the BuildingsStatMenu class for displaying statistics of various building types.

5.118 BuildingsStatMenu.h

Go to the documentation of this file.

```
00001
00006 #ifndef BUILDINGSSTATMENU_H
00007 #define BUILDINGSSTATMENU_H
00008
00009 #include "menus/base/IMenu.h"
00010 #include "utils/EntityType.h"
00011 #include <vector>
00012
00021 class BuildingsStatMenu : public IMenu
00022 {
00023 public:
00030     BuildingsStatMenu(std::vector<EntityType> types);
00031
00037     ~BuildingsStatMenu();
00038
00045     void display() const override;
00046
00052     void handleInput() override;
00053
00054 private:
00055     std::vector<EntityType> entityTypes;
00056
00064     void displayEntityStats(EntityType type) const;
00065
00074     void displayStatRow(const std::string &label, const std::string &value, int boxWidth) const;
00075 };
00076
00077 #endif // BUILDINGSSTATMENU_H
```

5.119 src/menus/buildings/demolish/DemolishMenu.h File Reference

Declares the DemolishMenu class for handling building demolition within the game.

```
#include "menus/base/IMenu.h"
#include <vector>
```

Classes

- class DemolishMenu

  Provides a menu interface for demolishing buildings in the game.

5.119.1 Detailed Description

Declares the DemolishMenu class for handling building demolition within the game.

5.120 DemolishMenu.h

Go to the documentation of this file.

```
00001
00006 #ifndef DEMOLISHMENU_H
00007 #define DEMOLISHMENU_H
00008
00009 #include "menus/base/IMenu.h"
00010 #include <vector>
00011
00020 class DemolishMenu : public IMenu
00021 {
00022 public:
00028     DemolishMenu();
00029
00033     ~DemolishMenu();
00034
00040     void display() const override;
00041
00047     void handleInput() override;
00048
00049 private:
00055     void demolishSpecificBuilding();
00056
00062     void demolishAllBuildingsOfType();
00063
00070     void confirmDemolish(const std::vector<std::pair<int, int» &positionsToDemolish);
00071 };
00072
00073 #endif // DEMOLISHMENU_H
```

5.121 BuyEconomicBuildingMenu.h

```
00001 #ifndef BUYECONOMICBUILDINGMENU_H
00002 #define BUYECONOMICBUILDINGMENU_H
00003
00004 #include "menus/base/BuyMenu.h"
00005 #include "managers/BuildingManager.h"
00006
00014 class BuyEconomicBuildingMenu : public BuyMenu
00015 {
00016 public:
00021     BuyEconomicBuildingMenu();
00022
00027     ~BuyEconomicBuildingMenu();
00028
00029 protected:
00038     EntityType chooseEntityType() override;
00039
00051     void buildEntity(EntityType type, Size size, int xPos, int yPos) override;
00052
00053 private:
00054     BuildingManager buildingManager;
00055 };
00056
00057 #endif // BUYECONOMICBUILDINGMENU_H
```

5.122 BuyResidentialBuildingMenu.h

```
00001 #ifndef BUYRESIDENTIALBUILDINGMENU_H
00002 #define BUYRESIDENTIALBUILDINGMENU_H
00003
00004 #include "menus/base/BuyMenu.h"
00005 #include "managers/BuildingManager.h"
00006
00014 class BuyResidentialBuildingMenu : public BuyMenu
00015 {
00016 public:
00021     BuyResidentialBuildingMenu();
00022
00027     ~BuyResidentialBuildingMenu();
00028
00029 protected:
00038     EntityType chooseEntityType() override;
00039
00051     void buildEntity(EntityType type, Size size, int xPos, int yPos) override;
00052
00053 private:
00054     BuildingManager buildingManager;
00055 };
00056
00057 #endif // BUYRESIDENTIALBUILDINGMENU_H
```

5.123 BuyResourceMenu.h

```
00001 #ifndef BUYRESOURCEMENU_H
00002 #define BUYRESOURCEMENU_H
00003
00004 #include "menus/base/BuyMenu.h"
00005 #include "managers/ResourceManager.h"
00006
00014 class BuyResourceMenu : public BuyMenu
00015 {
00016 public:
00021     BuyResourceMenu();
00022
00027     ~BuyResourceMenu();
00028
00029 protected:
00038     EntityType chooseEntityType() override;
00039
00050     void buildEntity(EntityType type, Size size, int xPos, int yPos) override;
00051
00052 private:
00053     ResourceManager resourceManager;
00054 };
00055
00056 #endif // BUYRESOURCEMENU_H
```

5.124 BuyServiceMenu.h

```
00001 #ifndef BUYSERVICEMENU_H
00002 #define BUYSERVICEMENU_H
00003
00004 #include "menus/base/BuyMenu.h"
00005 #include "managers/ServiceManager.h"
00006
00014 class BuyServiceMenu : public BuyMenu
00015 {
00016 public:
00021     BuyServiceMenu();
00022
00027     ~BuyServiceMenu();
00028
00029 protected:
00038     EntityType chooseEntityType() override;
00039
00050     void buildEntity(EntityType type, Size size, int xPos, int yPos) override;
00051
00052 private:
00053     ServiceManager serviceManager;
00054 };
00055
00056 #endif // BUYSERVICEMENU_H
```

5.125 BuyTransportMenu.h

```
00001 #ifndef BUYTRANSPORTMENU_H
00002 #define BUYTRANSPORTMENU_H
00003
00004 #include "menus/base/BuyMenu.h"
00005 #include "managers/TransportManager.h"
00006
00014 class BuyTransportMenu : public BuyMenu
00015 {
00016 public:
00021     BuyTransportMenu();
00022
00027     ~BuyTransportMenu();
00028
00029 protected:
00038     EntityType chooseEntityType() override;
00039
00050     void buildEntity(EntityType type, Size size, int xPos, int yPos) override;
00051
00052 private:
00053     TransportManager transportManager;
00054 };
00055
00056 #endif // BUYTRANSPORTMENU_H
```

5.126 BuyUtilityMenu.h

```
00001 #ifndef BUYUTILITYMENU_H
00002 #define BUYUTILITYMENU_H
00003
00004 #include "menus/base/BuyMenu.h"
00005 #include "managers/UtilityManager.h"
00006
00014 class BuyUtilityMenu : public BuyMenu
00015 {
00016 public:
00021     BuyUtilityMenu();
00022
00027     ~BuyUtilityMenu();
00028
00029 protected:
00038     EntityType chooseEntityType() override;
00039
00050     void buildEntity(EntityType type, Size size, int xPos, int yPos) override;
00051
00052 private:
00053     UtilityManager utilityManager;
00054 };
00055
00056 #endif // BUYUTILITYMENU_H
```

5.127 src/menus/main/DisplayCityMenu.h File Reference

Declares the DisplayCityMenu class for showing various city views in the game.

```
#include "menus/base/IMenu.h"
```

Classes

- class DisplayCityMenu

  Provides functionality to display the city and filter views by entity type.

5.127.1 Detailed Description

Declares the DisplayCityMenu class for showing various city views in the game.

5.128 DisplayCityMenu.h

Go to the documentation of this file.

```
00001
00006 #ifndef DISPLAYCITYMENU_H
00007 #define DISPLAYCITYMENU_H
00008
00009 #include "menus/base/IMenu.h"
00010
00019 class DisplayCityMenu : public IMenu
00020 {
00021 public:
00027     DisplayCityMenu();
00028
00034     ~DisplayCityMenu();
00035
00042     void display() const override;
00043
00050     void handleInput() override;
00051
00052 private:
00057     enum class DisplayMode
00058     {
00059         WHOLE_CITY,
00060         RESIDENTIAL,
00061         ECONOMIC,
00062         SERVICE,
00063         UTILITY,
00064         INDUSTRY,
00065         TRANSPORT
00066     };
00067
00068     DisplayMode currentDisplayMode = DisplayMode::WHOLE_CITY;
00069
00075     void displayCity() const;
00076
00083     template <typename T>
00084     void displayCityByType() const;
00085
00091     void displayFilteredCity() const;
00092 };
00093
00094 #endif // DISPLAYCITYMENU_H
```

5.129 src/menus/main/MainMenu.h File Reference

Defines the MainMenu class, representing the main interface of the game.

```
#include "menus/base/IMenu.h"
```

Classes

- class MainMenu

  Represents the main menu of the game, providing primary navigation options.

5.129.1 Detailed Description

Defines the MainMenu class, representing the main interface of the game.

5.130 MainMenu.h

Go to the documentation of this file.

```
00001
00006 #ifndef MAINMENU_H
00007 #define MAINMENU_H
00008
00009 #include "menus/base/IMenu.h"
00010
00018 class MainMenu : public IMenu
00019 {
00020 public:
00026     MainMenu();
00027
00033     ~MainMenu();
00034
00040     void display() const override;
00041
00048     void handleInput() override;
00049 };
00050
00051 #endif // MAINMENU_H
```

5.131 src/menus/policy/PolicyMenu.h File Reference

Declares the PolicyMenu class for managing policy-related interactions in the game.

```
#include "menus/base/IMenu.h"
#include "managers/GovernmentManager.h"
#include <vector>
#include <string>
```

Classes

- class PolicyMenu

  Provides functionality for players to apply and review city policies.

5.131.1 Detailed Description

Declares the PolicyMenu class for managing policy-related interactions in the game.

5.132 PolicyMenu.h

Go to the documentation of this file.

```
00001
00006 #ifndef POLICYMENU_H
00007 #define POLICYMENU_H
00008
00009 #include "menus/base/IMenu.h"
00010 #include "managers/GovernmentManager.h"
00011 #include <vector>
00012 #include <string>
00013
00021 class PolicyMenu : public IMenu
00022 {
00023 public:
00029     PolicyMenu();
00030
00036     ~PolicyMenu();
```

```
00037
00043     void display() const override;
00044
00050     void handleInput() override;
00051
00052 private:
00053     GovernmentManager governmentManager;
00054
00060     void selectWaterPolicy();
00061
00067     void selectElectricityPolicy();
00068
00074     void showPolicyHistory();
00075 };
00076
00077 #endif // POLICYMENU_H
```

5.133 src/menus/road/BuyRoadMenu.h File Reference

Declares the BuyRoadMenu class for managing road purchases in the game.

```
#include "menus/base/IMenu.h"
#include "managers/TransportManager.h"
```

Classes

- class BuyRoadMenu

  Provides functionality for players to purchase roads and place them in the city.

5.133.1 Detailed Description

Declares the BuyRoadMenu class for managing road purchases in the game.

5.134 BuyRoadMenu.h

Go to the documentation of this file.

```
00001
00006 #ifndef BUYROADMENU_H
00007 #define BUYROADMENU_H
00008
00009 #include "menus/base/IMenu.h"
00010 #include "managers/TransportManager.h"
00011
00019 class BuyRoadMenu : public IMenu
00020 {
00021 public:
00027     BuyRoadMenu();
00028
00034     virtual ~BuyRoadMenu();
00035
00041     void display() const override;
00042
00048     void handleInput() override;
00049
00050 private:
00051     TransportManager transportManager;
00052
00060     void chooseRoadPosition(int &xPos, int &yPos);
00061
00069     void confirmPurchase(int xPos, int yPos);
00070 };
00071
00072 #endif // BUYROADMENU_H
```

5.135 src/menus/stats/StatsMenu.h File Reference

Declares the StatsMenu class for managing and displaying city statistics and entity lists.

```
#include "menus/base/IMenu.h"
#include "city/City.h"
```

Classes

- class StatsMenu

  Provides functionality for displaying city statistics and various entity listings.

5.135.1 Detailed Description

Declares the StatsMenu class for managing and displaying city statistics and entity lists.

5.136 StatsMenu.h

Go to the documentation of this file.

```
00001
00006 #ifndef STATSMENU_H
00007 #define STATSMENU_H
00008
00009 #include "menus/base/IMenu.h"
00010 #include "city/City.h"
00011
00019 class StatsMenu : public IMenu
00020 {
00021 public:
00027     StatsMenu();
00028
00034     ~StatsMenu();
00035
00041     void display() const override;
00042
00048     void handleInput() override;
00049
00050 private:
00058     void listEntities(Iterator *it, std::string (*labelGenerator)(Entity *), std::string heading);
00059
00066     static std::string labelGenerator(Entity *entity);
00067
00074     static std::string residentialLabelGenerator(Entity *entity);
00075
00082     static std::string industrialLabelGenerator(Entity *entity);
00083
00090     static std::string utilityLabelGenerator(Entity *entity);
00091
00095     void showCityStats();
00096
00103     std::string formatSatisfaction(float satisfaction) const;
00104 };
00105
00106 #endif // STATSMENU_H
```

5.137 src/menus/tax/TaxMenu.h File Reference

Declares the TaxMenu class for managing tax adjustments in the game.

```
#include "menus/base/IMenu.h"
```

Classes

- class TaxMenu

  Provides functionality for managing and adjusting tax rates in the game.

5.137.1 Detailed Description

Declares the TaxMenu class for managing tax adjustments in the game.

5.138 TaxMenu.h

Go to the documentation of this file.

```
00001
00006 #ifndef TAXMENU_H
00007 #define TAXMENU_H
00008
00009 #include "menus/base/IMenu.h"
00010
00018 class TaxMenu : public IMenu
00019 {
00020 public:
00026     TaxMenu();
00027
00033     ~TaxMenu();
00034
00040     void display() const override;
00041
00048     void handleInput() override;
00049 };
00050
00051 #endif // TAXMENU_H
```

5.139 src/menus/upgrades/UpgradesMenu.h File Reference

Declares the UpgradesMenu class for upgrading various systems in the game.

```
#include "menus/base/IMenu.h"
#include "managers/UtilityManager.h"
#include "managers/ResourceManager.h"
#include <vector>
```

Classes

- class UpgradesMenu

  Provides a menu interface for upgrading utilities and industries in the game.

5.139.1 Detailed Description

Declares the UpgradesMenu class for upgrading various systems in the game.

5.140 UpgradesMenu.h

Go to the documentation of this file.

```
00001
00006 #ifndef UPGRADESMENU_H
00007 #define UPGRADESMENU_H
00008
00009 #include "menus/base/IMenu.h"
00010 #include "managers/UtilityManager.h"
00011 #include "managers/ResourceManager.h"
00012 #include <vector>
00013
00022 class UpgradesMenu : public IMenu
00023 {
00024 public:
00030     UpgradesMenu();
00031
00037     ~UpgradesMenu();
00038
00044     void display() const override;
00045
00051     void handleInput() override;
00052
00053 private:
00059     void upgradeUtilities();
00060
00066     void upgradeIndustries();
00067
00074     void selectSpecificUtility(const std::string &type, const std::vector<Utility *> &options);
00075
00082     void selectSpecificIndustry(const std::string &type, const std::vector<Industry *> &options);
00083
00091     void confirmUpgrade(const std::string &entityName, int currentLevel, int upgradeCost);
00092
00093 private:
00094     UtilityManager utilityManager;
00095     ResourceManager resourceManager;
00096 };
00097
00098 #endif // UPGRADESMENU_H
```

5.141 Policy.h

```
00001 #ifndef POLICY_H
00002 #define POLICY_H
00003
00004 #include "utils/Memento.h"
00005 #include <string>
00006
00012 class Policy {
00013 private:
00014     std::string name;
00015     std::string detail;
00016
00017 public:
00024     Policy(const std::string& name, const std::string& detail);
00025
00031     Memento* createMemento() const;
00032
00038     void setMemento(const Memento* memento);
00039
00045     std::string getName() const;
00046
00052     std::string getDetail() const;
00053 };
00054
00055 #endif
```

5.142 ElectricityPolicy.h

```
00001 #ifndef ELECTRICITYPOLICY_H
00002 #define ELECTRICITYPOLICY_H
00003
00004 #include "policies/base/Policy.h"
00005
00011 class ElectricityPolicy : public Policy {
00012 public:
```

```
00019     ElectricityPolicy(const std::string& name, const std::string& detail) : Policy(name, detail) {}
00020
00027     virtual int calculateElectricityUsage(int electricityUsage) = 0;
00028
00032     virtual ~ElectricityPolicy() {}
00033 };
00034
00035 #endif
```

5.143 HighElectricityPolicy.h

```
00001 #ifndef HIGHELECTRICITYPOLICY_H
00002 #define HIGHELECTRICITYPOLICY_H
00003
00004 #include "ElectricityPolicy.h"
00005
00011 class HighElectricityPolicy : public ElectricityPolicy {
00012 public:
00019     int calculateElectricityUsage(int electricityUsage) override;
00020
00025     HighElectricityPolicy() : ElectricityPolicy("HighElectricityPolicy", "This policy allows for high
      electricity usage.") {}
00026 };
00027
00028 #endif
```

5.144 LowElectricityPolicy.h

```
00001 #ifndef LOWELECTRICITYPOLICY_H
00002 #define LOWELECTRICITYPOLICY_H
00003
00004 #include "ElectricityPolicy.h"
00005
00011 class LowElectricityPolicy : public ElectricityPolicy {
00012 public:
00019     int calculateElectricityUsage(int electricityUsage) override;
00020
00025     LowElectricityPolicy() : ElectricityPolicy("LowElectricityPolicy", "This policy minimises
      electricity usage.") {}
00026 };
00027
00028 #endif
```

5.145 NormalElectricityPolicy.h

```
00001 #ifndef NORMALELECTRICITYPOLICY_H
00002 #define NORMALELECTRICITYPOLICY_H
00003
00004 #include "ElectricityPolicy.h"
00005
00011 class NormalElectricityPolicy : public ElectricityPolicy {
00012 public:
00019     int calculateElectricityUsage(int electricityUsage) override;
00020
00025     NormalElectricityPolicy() : ElectricityPolicy("NormalElectricityPolicy", "This policy maintains
      average electricity usage.") {}
00026 };
00027
00028 #endif
```

5.146 HighWaterPolicy.h

```
00001 #ifndef HIGHWATERPOLICY_H
00002 #define HIGHWATERPOLICY_H
00003
00004 #include "WaterPolicy.h"
00005
00011 class HighWaterPolicy : public WaterPolicy {
00012 public:
00019     int calculateWaterUsage(int waterUsage) override;
00024     HighWaterPolicy() : WaterPolicy("HighWaterPolicy", "This policy allows for high water usage.") {}
00025 };
00026
00027 #endif
```

5.147 LowWaterPolicy.h

```
00001 #ifndef LOWWATERPOLICY_H
00002 #define LOWWATERPOLICY_H
00003
00004 #include "WaterPolicy.h"
00005
00011 class LowWaterPolicy : public WaterPolicy {
00012 public:
00019     int calculateWaterUsage(int waterUsage) override;
00024     LowWaterPolicy() : WaterPolicy("LowWaterPolicy", "This policy minimises water usage.") {}
00025 };
00026
00027 #endif
```

5.148 NormalWaterPolicy.h

```
00001 #ifndef NORMALWATERPOLICY_H
00002 #define NORMALWATERPOLICY_H
00003
00004 #include "WaterPolicy.h"
00005
00011 class NormalWaterPolicy : public WaterPolicy {
00012 public:
00019     int calculateWaterUsage(int waterUsage) override;
00024     NormalWaterPolicy() : WaterPolicy("NormalWaterPolicy", "This policy maintains average water
      usage.") {}
00025 };
00026
00027 #endif
```

5.149 WaterPolicy.h

```
00001 #ifndef WATERPOLICY_H
00002 #define WATERPOLICY_H
00003
00004 #include "policies/base/Policy.h"
00005
00011 class WaterPolicy : public Policy {
00012 public:
00019     WaterPolicy(const std::string& name, const std::string& detail) : Policy(name, detail) {}
00026     virtual int calculateWaterUsage(int waterUsage) = 0;
00027
00031     virtual ~WaterPolicy() {}
00032 };
00033
00034 #endif
```

5.150 BSPPartitioner.h

```
00001 #ifndef BSP_PARTITIONER_H
00002 #define BSP_PARTITIONER_H
00003
00004 #include <vector>
00005
00009 class Rectangle
00010 {
00011 public:
00012     int x;
00013     int y;
00014     int width;
00015     int height;
00016
00024     Rectangle(int x, int y, int width, int height) : x(x), y(y), width(width), height(height) {}
00025 };
00026
00031 class BSPPartitioner
00032 {
00033 private:
00034     int gridWidth;
00035     int gridHeight;
00036     int minWidth;
00037     int minHeight;
00038     int gap;
00039     std::vector<Rectangle> rooms;
```

```
00040     std::vector<Rectangle> gaps;
00041
00046     void split(Rectangle rect);
00047
00048 public:
00057     BSPPartitioner(int gridWidth, int gridHeight, int minWidth, int minHeight, int gap);
00058
00062     void partition();
00063
00068     const std::vector<Rectangle> &getRooms() const;
00069
00074     const std::vector<Rectangle> &getGaps() const;
00075 };
00076
00077 #endif // BSP_PARTITIONER_H
```

5.151 Caretaker.h

```
00001 #ifndef CARETAKER_H
00002 #define CARETAKER_H
00003
00004 #include "Memento.h"
00005 #include <vector>
00006
00012 class Caretaker {
00013 private:
00014     std::vector<Memento*> pastStrategies;
00015
00016 public:
00022     void setMemento(Memento* memento);
00023
00029     Memento* getMemento() const;
00030
00034     ~Caretaker();
00035
00041     std::vector<Memento*> getPastPolicies() const;
00042 };
00043
00044 #endif
```

5.152 src/utils/ConfigManager.h File Reference

Manages entity and satisfaction configurations for different entity types and sizes.

```
#include <map>
#include <optional>
#include "EntityType.h"
#include "Size.h"
#include "EntityConfig.h"
#include "SatisfactionConfig.h"
```

Classes

- class ConfigManager

  Singleton class to manage configurations for various entities.

5.152.1 Detailed Description

Manages entity and satisfaction configurations for different entity types and sizes.

5.153 ConfigManager.h

Go to the documentation of this file.

```
00001
00006 #ifndef CONFIGMANAGER_H
00007 #define CONFIGMANAGER_H
00008
00009 #include <map>
00010 #include <optional>
00011 #include "EntityType.h"
00012 #include "Size.h"
00013 #include "EntityConfig.h"
00014 #include "SatisfactionConfig.h"
00015
00020 class ConfigManager
00021 {
00022 public:
00029     static EntityConfig getEntityConfig(EntityType entityType, Size size)
00030     {
00031         static ConfigManager instance;
00032         auto key = std::make_pair(entityType, size);
00033         return instance.entityConfigTable.at(key); // Use at to safely access the map
00034     }
00035
00041     static SatisfactionConfig getSatisfactionConfig(EntityType entityType)
00042     {
00043         static ConfigManager instance;
00044         return instance.satisfactionConfigTable.at(entityType); // Use `at` to safely access the map
00045     }
00046
00047 private:
00048     std::map<std::pair<EntityType, Size>, EntityConfig> entityConfigTable;
00049     std::map<EntityType, SatisfactionConfig> satisfactionConfigTable;
00050
00054     ConfigManager()
00055     {
00056         initializeEntityConfig();
00057         initializeSatisfactionConfig();
00058     }
00059
00063     void initializeEntityConfig()
00064     {
00065         // EntityConfig(Cost(money, wood, stone, concrete), electricity, water, symbol, radius,
    localEffect, globalEffect, width, height, revenue, buildTime)
00066
00067         // Houses
00068         entityConfigTable.emplace(std::make_pair(EntityType::HOUSE, Size::SMALL),
00069                                   EntityConfig(Cost(100, 50, 20, 10), 2, 1, "H", 3, 5, 1, 2, 2, 500,
    3, EntityType::HOUSE, Size::SMALL));
00070         entityConfigTable.emplace(std::make_pair(EntityType::HOUSE, Size::MEDIUM),
00071                                   EntityConfig(Cost(200, 80, 30, 20), 5, 2, "H", 5, 10, 2, 3, 3, 1000,
    3, EntityType::HOUSE, Size::MEDIUM));
00072         entityConfigTable.emplace(std::make_pair(EntityType::HOUSE, Size::LARGE),
00073                                   EntityConfig(Cost(300, 120, 50, 30), 7, 3, "H", 7, 15, 3, 4, 4,
    1500, 3, EntityType::HOUSE, Size::LARGE));
00074
00075         // Apartments
00076         entityConfigTable.emplace(std::make_pair(EntityType::APARTMENT, Size::SMALL),
00077                                   EntityConfig(Cost(150, 70, 40, 15), 3, 1, "A", 4, 6, 2, 3, 3, 600,
    3, EntityType::APARTMENT, Size::SMALL));
00078         entityConfigTable.emplace(std::make_pair(EntityType::APARTMENT, Size::MEDIUM),
00079                                   EntityConfig(Cost(300, 140, 60, 30), 6, 3, "A", 6, 12, 4, 4, 4,
    1200, 3, EntityType::APARTMENT, Size::MEDIUM));
00080         entityConfigTable.emplace(std::make_pair(EntityType::APARTMENT, Size::LARGE),
00081                                   EntityConfig(Cost(450, 210, 90, 45), 9, 4, "A", 9, 18, 6, 5, 5,
    1800, 3, EntityType::APARTMENT, Size::LARGE));
00082
00083         // Factories
00084         entityConfigTable.emplace(std::make_pair(EntityType::FACTORY, Size::SMALL),
00085                                   EntityConfig(Cost(500, 200, 100, 50), 20, 10, "", 10, -5, -10, 4, 4,
    3000, 3, EntityType::FACTORY, Size::SMALL));
00086         entityConfigTable.emplace(std::make_pair(EntityType::FACTORY, Size::MEDIUM),
00087                                   EntityConfig(Cost(700, 300, 150, 80), 30, 15, "", 15, -10, -15, 6,
    6, 5000, 3, EntityType::FACTORY, Size::MEDIUM));
00088         entityConfigTable.emplace(std::make_pair(EntityType::FACTORY, Size::LARGE),
00089                                   EntityConfig(Cost(1000, 400, 200, 100), 50, 20, "", 20, -20, -25, 8,
    8, 10000, 3, EntityType::FACTORY, Size::LARGE));
00090
00091         // Offices
00092         entityConfigTable.emplace(std::make_pair(EntityType::OFFICE, Size::SMALL),
00093                                   EntityConfig(Cost(400, 100, 60, 20), 10, 4, "", 5, 5, 2, 3, 3, 2000,
    3, EntityType::OFFICE, Size::SMALL));
00094         entityConfigTable.emplace(std::make_pair(EntityType::OFFICE, Size::MEDIUM),
00095                                   EntityConfig(Cost(600, 150, 90, 30), 20, 8, "", 7, 10, 4, 4, 4,
    4000, 3, EntityType::OFFICE, Size::MEDIUM));
```

```
00096         entityConfigTable.emplace(std::make_pair(EntityType::OFFICE, Size::LARGE),
00097                                   EntityConfig(Cost(900, 250, 120, 40), 30, 12, "", 10, 15, 6, 5, 5,
      6000, 3, EntityType::OFFICE, Size::LARGE));
00098
00099         // Shopping Malls
00100         entityConfigTable.emplace(std::make_pair(EntityType::SHOPPINGMALL, Size::SMALL),
00101                                   EntityConfig(Cost(800, 300, 150, 100), 40, 25, "", 8, 15, 10, 5, 5,
      8000, 3, EntityType::SHOPPINGMALL, Size::SMALL));
00102         entityConfigTable.emplace(std::make_pair(EntityType::SHOPPINGMALL, Size::MEDIUM),
00103                                   EntityConfig(Cost(1500, 500, 300, 200), 60, 35, "", 12, 20, 15, 8,
      8, 15000, 3, EntityType::SHOPPINGMALL, Size::MEDIUM));
00104         entityConfigTable.emplace(std::make_pair(EntityType::SHOPPINGMALL, Size::LARGE),
00105                                   EntityConfig(Cost(3000, 1000, 500, 400), 100, 50, "", 15, 30, 20,
      10, 10, 30000, 3, EntityType::SHOPPINGMALL, Size::LARGE));
00106
00107         // Schools
00108         entityConfigTable.emplace(std::make_pair(EntityType::SCHOOL, Size::SMALL),
00109                                   EntityConfig(Cost(600, 200, 80, 50), 10, 5, "", 6, 8, 4, 4, 4,
      -2500, 3, EntityType::SCHOOL, Size::SMALL));
00110         entityConfigTable.emplace(std::make_pair(EntityType::SCHOOL, Size::MEDIUM),
00111                                   EntityConfig(Cost(1200, 400, 160, 100), 20, 10, "", 10, 12, 6, 6, 6,
      -5000, 3, EntityType::SCHOOL, Size::MEDIUM));
00112         entityConfigTable.emplace(std::make_pair(EntityType::SCHOOL, Size::LARGE),
00113                                   EntityConfig(Cost(1800, 600, 240, 150), 30, 15, "", 15, 18, 8, 8, 8,
      -7500, 3, EntityType::SCHOOL, Size::LARGE));
00114
00115         // Hospitals
00116         entityConfigTable.emplace(std::make_pair(EntityType::HOSPITAL, Size::SMALL),
00117                                   EntityConfig(Cost(1000, 400, 200, 150), 25, 10, "", 8, 10, 6, 6, 6,
      -10000, 3, EntityType::HOSPITAL, Size::SMALL));
00118         entityConfigTable.emplace(std::make_pair(EntityType::HOSPITAL, Size::MEDIUM),
00119                                   EntityConfig(Cost(2000, 600, 300, 200), 35, 15, "", 12, 15, 10, 8,
      8, -15000, 3, EntityType::HOSPITAL, Size::MEDIUM));
00120         entityConfigTable.emplace(std::make_pair(EntityType::HOSPITAL, Size::LARGE),
00121                                   EntityConfig(Cost(4000, 1000, 500, 300), 50, 20, "", 20, 25, 15, 10,
      10, -30000, 3, EntityType::HOSPITAL, Size::LARGE));
00122
00123         // Police Stations
00124         entityConfigTable.emplace(std::make_pair(EntityType::POLICESTATION, Size::SMALL),
00125                                   EntityConfig(Cost(700, 250, 120, 70), 8, 5, "", 6, 10, 4, 4, 4,
      -4000, 3, EntityType::POLICESTATION, Size::SMALL));
00126         entityConfigTable.emplace(std::make_pair(EntityType::POLICESTATION, Size::MEDIUM),
00127                                   EntityConfig(Cost(1200, 400, 200, 100), 12, 7, "", 10, 15, 6, 6, 6,
      -6000, 3, EntityType::POLICESTATION, Size::MEDIUM));
00128         entityConfigTable.emplace(std::make_pair(EntityType::POLICESTATION, Size::LARGE),
00129                                   EntityConfig(Cost(2000, 600, 300, 150), 18, 10, "", 15, 20, 8, 8, 8,
      -10000, 3, EntityType::POLICESTATION, Size::LARGE));
00130
00131         // Parks
00132         entityConfigTable.emplace(std::make_pair(EntityType::PARK, Size::SMALL),
00133                                   EntityConfig(Cost(300, 100, 50, 25), 0, 0, "", 3, 12, 8, 4, 4, -0,
      2, EntityType::PARK, Size::SMALL));
00134         entityConfigTable.emplace(std::make_pair(EntityType::PARK, Size::MEDIUM),
00135                                   EntityConfig(Cost(600, 200, 100, 50), 0, 0, "", 6, 18, 10, 6, 6, -0,
      2, EntityType::PARK, Size::MEDIUM));
00136         entityConfigTable.emplace(std::make_pair(EntityType::PARK, Size::LARGE),
00137                                   EntityConfig(Cost(1200, 400, 200, 100), 0, 0, "", 10, 25, 15, 8, 8,
      -0, 2, EntityType::PARK, Size::LARGE));
00138
00139         // Monuments
00140         entityConfigTable.emplace(std::make_pair(EntityType::MONUMENT, Size::SMALL),
00141                                   EntityConfig(Cost(2000, 700, 400, 300), 0, 0, "", 12, 25, 18, 5, 5,
      -0, 5, EntityType::MONUMENT, Size::SMALL));
00142         entityConfigTable.emplace(std::make_pair(EntityType::MONUMENT, Size::MEDIUM),
00143                                   EntityConfig(Cost(3500, 1000, 600, 400), 0, 0, "", 18, 30, 22, 7, 7,
      -0, 5, EntityType::MONUMENT, Size::MEDIUM));
00144         entityConfigTable.emplace(std::make_pair(EntityType::MONUMENT, Size::LARGE),
00145                                   EntityConfig(Cost(6000, 1500, 900, 600), 0, 0, "", 25, 35, 28, 10,
      10, -0, 5, EntityType::MONUMENT, Size::LARGE));
00146
00147         // Theater
00148         entityConfigTable.emplace(std::make_pair(EntityType::THEATER, Size::SMALL),
00149                                   EntityConfig(Cost(1200, 400, 200, 150), 20, 10, "", 8, 12, 10, 6, 6,
      -5000, 3, EntityType::THEATER, Size::SMALL));
00150         entityConfigTable.emplace(std::make_pair(EntityType::THEATER, Size::MEDIUM),
00151                                   EntityConfig(Cost(2500, 800, 500, 300), 30, 15, "", 12, 20, 15, 8,
      8, -10000, 3, EntityType::THEATER, Size::MEDIUM));
00152         entityConfigTable.emplace(std::make_pair(EntityType::THEATER, Size::LARGE),
00153                                   EntityConfig(Cost(4500, 1200, 700, 500), 45, 25, "", 18, 30, 20, 10,
      10, -15000, 3, EntityType::THEATER, Size::LARGE));
00154
00155         // Power Plants
00156         entityConfigTable.emplace(std::make_pair(EntityType::POWERPLANT, Size::SMALL),
00157                                   EntityConfig(Cost(3000, 1200, 700, 500), 200, 0, "", 15, -10, -15,
      6, 6, -0, 6, EntityType::POWERPLANT, Size::SMALL));
00158         entityConfigTable.emplace(std::make_pair(EntityType::POWERPLANT, Size::MEDIUM),
00159                                   EntityConfig(Cost(5500, 2000, 1200, 800), 400, 0, "", 20, -20, -30,
```

```
      8, 8, -0, 8, EntityType::POWERPLANT, Size::MEDIUM));
00160         entityConfigTable.emplace(std::make_pair(EntityType::POWERPLANT, Size::LARGE),
00161                                   EntityConfig(Cost(8000, 3000, 2000, 1200), 600, 0, "", 30, -30, -45,
      10, -10, 0, 10, EntityType::POWERPLANT, Size::LARGE));
00162
00163         // Water Supply Systems
00164         entityConfigTable.emplace(std::make_pair(EntityType::WATERSUPPLY, Size::SMALL),
00165                                   EntityConfig(Cost(2000, 900, 400, 250), 0, 50, "", 10, 8, 5, 5, 5,
      -0, 4, EntityType::WATERSUPPLY, Size::SMALL));
00166         entityConfigTable.emplace(std::make_pair(EntityType::WATERSUPPLY, Size::MEDIUM),
00167                                   EntityConfig(Cost(3500, 1200, 600, 400), 0, 100, "", 15, 15, 10, 7,
      7, -0, 6, EntityType::WATERSUPPLY, Size::MEDIUM));
00168         entityConfigTable.emplace(std::make_pair(EntityType::WATERSUPPLY, Size::LARGE),
00169                                   EntityConfig(Cost(5000, 2000, 1000, 700), 0, 200, "", 20, 25, 15,
      10, 10, -0, 8, EntityType::WATERSUPPLY, Size::LARGE));
00170
00171         // Waste Management
00172         entityConfigTable.emplace(std::make_pair(EntityType::WASTEMANAGEMENT, Size::SMALL),
00173                                   EntityConfig(Cost(1000, 500, 300, 150), 10, 5, "", 5, -5, -5, 5, 5,
      -0, 3, EntityType::WASTEMANAGEMENT, Size::SMALL));
00174         entityConfigTable.emplace(std::make_pair(EntityType::WASTEMANAGEMENT, Size::MEDIUM),
00175                                   EntityConfig(Cost(2000, 900, 500, 300), 15, 10, "", 8, -10, -10, 7,
      7, -0, 5, EntityType::WASTEMANAGEMENT, Size::MEDIUM));
00176         entityConfigTable.emplace(std::make_pair(EntityType::WASTEMANAGEMENT, Size::LARGE),
00177                                   EntityConfig(Cost(3500, 1500, 800, 600), 25, 15, "", 12, -15, -15,
      9, 9, -0, 7, EntityType::WASTEMANAGEMENT, Size::LARGE));
00178
00179         // Sewage Systems
00180         entityConfigTable.emplace(std::make_pair(EntityType::SEWAGESYSTEM, Size::SMALL),
00181                                   EntityConfig(Cost(1500, 700, 400, 250), 5, 10, "", 6, -6, -8, 5, 5,
      -0, 4, EntityType::SEWAGESYSTEM, Size::SMALL));
00182         entityConfigTable.emplace(std::make_pair(EntityType::SEWAGESYSTEM, Size::MEDIUM),
00183                                   EntityConfig(Cost(2500, 1100, 600, 400), 10, 15, "", 8, -10, -12, 7,
      7, -0, 6, EntityType::SEWAGESYSTEM, Size::MEDIUM));
00184         entityConfigTable.emplace(std::make_pair(EntityType::SEWAGESYSTEM, Size::LARGE),
00185                                   EntityConfig(Cost(4000, 1500, 1000, 800), 20, 25, "", 12, -15, -20,
      10, 10, -0, 8, EntityType::SEWAGESYSTEM, Size::LARGE));
00186
00187         // Stone Producer
00188         entityConfigTable.emplace(std::make_pair(EntityType::STONEPRODUCER, Size::SMALL),
00189                                   EntityConfig(Cost(1200, 600, 300, 200), 15, 0, "", 6, 8, 5, 4, 4,
      -200, 3, EntityType::STONEPRODUCER, Size::SMALL));
00190         entityConfigTable.emplace(std::make_pair(EntityType::STONEPRODUCER, Size::MEDIUM),
00191                                   EntityConfig(Cost(2000, 1000, 500, 400), 30, 0, "", 10, 15, 10, 6,
      6, -400, 3, EntityType::STONEPRODUCER, Size::MEDIUM));
00192         entityConfigTable.emplace(std::make_pair(EntityType::STONEPRODUCER, Size::LARGE),
00193                                   EntityConfig(Cost(3500, 1500, 800, 600), 50, 0, "", 15, 25, 20, 8,
      8, -700, 3, EntityType::STONEPRODUCER, Size::LARGE));
00194
00195         // Wood Producer
00196         entityConfigTable.emplace(std::make_pair(EntityType::WOODPRODUCER, Size::SMALL),
00197                                   EntityConfig(Cost(1000, 500, 250, 150), 10, 5, "", 5, 8, 5, 4, 4,
      -300, 3, EntityType::WOODPRODUCER, Size::SMALL));
00198         entityConfigTable.emplace(std::make_pair(EntityType::WOODPRODUCER, Size::MEDIUM),
00199                                   EntityConfig(Cost(1800, 900, 450, 300), 20, 10, "", 8, 12, 10, 6, 6,
      -500, 3, EntityType::WOODPRODUCER, Size::MEDIUM));
00200         entityConfigTable.emplace(std::make_pair(EntityType::WOODPRODUCER, Size::LARGE),
00201                                   EntityConfig(Cost(3000, 1500, 750, 500), 40, 15, "", 12, 18, 15, 8,
      8, -800, 3, EntityType::WOODPRODUCER, Size::LARGE));
00202
00203         // Concrete Producer
00204         entityConfigTable.emplace(std::make_pair(EntityType::CONCRETEPRODUCER, Size::SMALL),
00205                                   EntityConfig(Cost(1400, 700, 350, 200), 20, 5, "", 6, 10, 5, 4, 4,
      -250, 4, EntityType::CONCRETEPRODUCER, Size::SMALL));
00206         entityConfigTable.emplace(std::make_pair(EntityType::CONCRETEPRODUCER, Size::MEDIUM),
00207                                   EntityConfig(Cost(2500, 1200, 600, 400), 35, 10, "", 9, 18, 10, 6,
      6, -600, 4, EntityType::CONCRETEPRODUCER, Size::MEDIUM));
00208         entityConfigTable.emplace(std::make_pair(EntityType::CONCRETEPRODUCER, Size::LARGE),
00209                                   EntityConfig(Cost(4000, 2000, 1000, 700), 60, 15, "", 12, 25, 15, 8,
      8, -1000, 4, EntityType::CONCRETEPRODUCER, Size::LARGE));
00210
00211         // Road
00212         entityConfigTable.emplace(std::make_pair(EntityType::ROAD, Size::SMALL),
00213                                   EntityConfig(Cost(50, 10, 5, 0), 0, 0, "", 1, 0, 0, 1, 1, 0, 1,
      EntityType::ROAD, Size::SMALL));
00214         entityConfigTable.emplace(std::make_pair(EntityType::ROAD, Size::MEDIUM),
00215                                   EntityConfig(Cost(100, 20, 10, 0), 0, 0, "", 1, 0, 0, 1, 1, 0, 1,
      EntityType::ROAD, Size::MEDIUM));
00216         entityConfigTable.emplace(std::make_pair(EntityType::ROAD, Size::LARGE),
00217                                   EntityConfig(Cost(200, 30, 15, 0), 0, 0, "", 1, 0, 0, 1, 1, 0, 1,
      EntityType::ROAD, Size::LARGE));
00218
00219         // Airport
00220         entityConfigTable.emplace(std::make_pair(EntityType::AIRPORT, Size::SMALL),
00221                                   EntityConfig(Cost(10000, 4000, 2000, 1000), 100, 50, "", 20, -15,
      -10, 10, 10, -0, 10, EntityType::AIRPORT, Size::SMALL));
00222         entityConfigTable.emplace(std::make_pair(EntityType::AIRPORT, Size::MEDIUM),
```

```
00223                                   EntityConfig(Cost(15000, 6000, 3000, 1500), 200, 100, "", 30, -20,
      -15, 12, 12, -0, 15, EntityType::AIRPORT, Size::MEDIUM));
00224         entityConfigTable.emplace(std::make_pair(EntityType::AIRPORT, Size::LARGE),
00225                                   EntityConfig(Cost(25000, 10000, 5000, 3000), 400, 150, "", 40, -30,
      -20, 15, 15, -0, 20, EntityType::AIRPORT, Size::LARGE));
00226
00227         // BusStop
00228         entityConfigTable.emplace(std::make_pair(EntityType::BUSSTOP, Size::SMALL),
00229                                   EntityConfig(Cost(100, 20, 10, 5), 0, 0, "", 3, 5, 3, 2, 2, -0, 1,
      EntityType::BUSSTOP, Size::SMALL));
00230         entityConfigTable.emplace(std::make_pair(EntityType::BUSSTOP, Size::MEDIUM),
00231                                   EntityConfig(Cost(200, 40, 20, 10), 0, 0, "", 4, 8, 5, 3, 3, -0, 1,
      EntityType::BUSSTOP, Size::MEDIUM));
00232         entityConfigTable.emplace(std::make_pair(EntityType::BUSSTOP, Size::LARGE),
00233                                   EntityConfig(Cost(500, 100, 50, 25), 0, 0, "", 5, 12, 8, 4, 4, -0,
      1, EntityType::BUSSTOP, Size::LARGE));
00234
00235         // TrainStation
00236         entityConfigTable.emplace(std::make_pair(EntityType::TRAINSTATION, Size::SMALL),
00237                                   EntityConfig(Cost(1500, 600, 300, 200), 15, 0, "", 10, 8, 5, 5, 5,
      -0, 4, EntityType::TRAINSTATION, Size::SMALL));
00238         entityConfigTable.emplace(std::make_pair(EntityType::TRAINSTATION, Size::MEDIUM),
00239                                   EntityConfig(Cost(3000, 1200, 600, 400), 30, 0, "", 15, 15, 10, 7,
      7, -0, 6, EntityType::TRAINSTATION, Size::MEDIUM));
00240         entityConfigTable.emplace(std::make_pair(EntityType::TRAINSTATION, Size::LARGE),
00241                                   EntityConfig(Cost(6000, 2400, 1200, 800), 50, 0, "", 20, 25, 15, 10,
      10, -0, 8, EntityType::TRAINSTATION, Size::LARGE));
00242     }
00243
00247     void initializeSatisfactionConfig()
00248     {
00249         // SatisfactionConfig(localRate, globalRate, localExtreme, globalExtreme)
00250
00251         // Hospital
00252         satisfactionConfigTable.emplace(EntityType::HOSPITAL, SatisfactionConfig(2.0f, 2.0f, 10.0f,
      5.0f));
00253
00254         // Police Station
00255         satisfactionConfigTable.emplace(EntityType::POLICESTATION, SatisfactionConfig(2.0f, 2.0f,
      10.0f, 2.0f));
00256
00257         // School
00258         satisfactionConfigTable.emplace(EntityType::SCHOOL, SatisfactionConfig(2.0f, 2.0f, 10.0f,
      5.0f));
00259
00260         // Park
00261         satisfactionConfigTable.emplace(EntityType::PARK, SatisfactionConfig(2.0f, 2.0f, 15.0f,
      0.0f));
00262
00263         // Monument
00264         satisfactionConfigTable.emplace(EntityType::MONUMENT, SatisfactionConfig(2.0f, 2.0f, 15.0f,
      0.0f));
00265
00266         // Theater
00267         satisfactionConfigTable.emplace(EntityType::THEATER, SatisfactionConfig(2.0f, 2.0f, 15.0f,
      0.0f));
00268
00269         // Power Plant
00270         satisfactionConfigTable.emplace(EntityType::POWERPLANT, SatisfactionConfig(-2.0f, 2.0f,
      -25.0f, 0.0f));
00271
00272         // Water Supply
00273         satisfactionConfigTable.emplace(EntityType::WATERSUPPLY, SatisfactionConfig(-2.0f, -2.0f,
      -25.0f, 0.0f));
00274
00275         // Waste Managment
00276         satisfactionConfigTable.emplace(EntityType::WASTEMANAGEMENT, SatisfactionConfig(-2.0f, 2.0f,
      -25.0f, 0.0f));
00277
00278         // Sewage System
00279         satisfactionConfigTable.emplace(EntityType::SEWAGESYSTEM, SatisfactionConfig(-2.0f, 2.0f,
      -25.0f, 0.0f));
00280
00281         // Wood Producer
00282         satisfactionConfigTable.emplace(EntityType::WOODPRODUCER, SatisfactionConfig(-2.0f, 2.0f,
      -25.0f, 0.0f));
00283
00284         // Stone Producer
00285         satisfactionConfigTable.emplace(EntityType::STONEPRODUCER, SatisfactionConfig(-2.0f, 2.0f,
      -25.0f, 0.0f));
00286
00287         // Concrete Producer
00288         satisfactionConfigTable.emplace(EntityType::CONCRETEPRODUCER, SatisfactionConfig(-2.0f, 2.0f,
      -25.0f, 0.0f));
00289
00290         // Airport
00291         satisfactionConfigTable.emplace(EntityType::AIRPORT, SatisfactionConfig(-2.0f, 2.0f, -15.0f,
```

```
      5.0f));
00292
00293         // Bus Stop
00294         satisfactionConfigTable.emplace(EntityType::BUSSTOP, SatisfactionConfig(2.0f, 2.0f, 10.0f,
      2.0f));
00295
00296         // Train Station
00297         satisfactionConfigTable.emplace(EntityType::TRAINSTATION, SatisfactionConfig(2.0f, 2.0f,
      10.0f, 2.0f));
00298
00299         // Factory
00300         satisfactionConfigTable.emplace(EntityType::FACTORY, SatisfactionConfig(-2.0f, 2.0f, -15.0f,
      10.0f));
00301
00302         // Shopping Mall
00303         satisfactionConfigTable.emplace(EntityType::SHOPPINGMALL, SatisfactionConfig(2.0f, 2.0f, 3.0f,
      10.0f));
00304
00305         // Office
00306         satisfactionConfigTable.emplace(EntityType::OFFICE, SatisfactionConfig(-2.0f, 2.0f, -10.0f,
      10.0f));
00307     }
00308 };
00309
00310 #endif // CONFIGMANAGER_H
```

5.154 Cost.h

```
00001 #ifndef COST_H
00002 #define COST_H
00003
00011 struct Cost
00012 {
00013     int moneyCost;
00014     int woodCost;
00015     int stoneCost;
00016     int concreteCost;
00025     Cost(int money = 0, int wood = 0, int stone = 0, int concrete = 0)
00026         : moneyCost(money), woodCost(wood), stoneCost(stone), concreteCost(concrete) {}
00027
00033     bool operator==(const Cost& other) const {
00034         return moneyCost == other.moneyCost &&
00035                woodCost == other.woodCost &&
00036                stoneCost == other.stoneCost &&
00037                concreteCost == other.concreteCost;
00038     }
00039 };
00040
00041 #endif // COST_H
```

5.155 EntityConfig.h

```
00001 #ifndef ENTITYCONFIG_H
00002 #define ENTITYCONFIG_H
00003
00004 #include "Cost.h"
00005 #include "EntityType.h" // Include EntityType
00006 #include "Size.h"       // Include Size
00007 #include <string>
00008
00010 struct EntityConfig
00011 {
00012   Cost cost;
00013   int electricityConsumption;
00014   int waterConsumption;
00015   std::string symbol;
00016   int effectRadius;
00017   int localEffectStrength;
00018   int globalEffectStrength;
00019   int width;
00020   int height;
00021   int revenue;
00022   int buildTime;
00023   EntityType entityType;
00024   Size size;
00025
00027   EntityConfig()
00028       : cost(), electricityConsumption(0), waterConsumption(0), symbol(""),
00029         effectRadius(0), localEffectStrength(0), globalEffectStrength(0),
```

```
00030         width(1), height(1), revenue(0), buildTime(0),
00031         entityType(EntityType::UNKNOWN), size(Size::SMALL) {}
00032
00047   EntityConfig(const Cost &cost, int electricity, int water, const std::string &symbol,
00048                int radius, int localEffect, int globalEffect, int width, int height,
00049                int revenue, int buildTime, EntityType entityType, Size size)
00050       : cost(cost), electricityConsumption(electricity), waterConsumption(water), symbol(symbol),
00051         effectRadius(radius), localEffectStrength(localEffect), globalEffectStrength(globalEffect),
00052         width(width), height(height), revenue(revenue), buildTime(buildTime),
00053         entityType(entityType), size(size) {}
00054 };
00055
00056 #endif // ENTITYCONFIG_H
```

5.156 EntityType.h

```
00001 #ifndef ENTITYTYPE_H
00002 #define ENTITYTYPE_H
00003
00004 #include <string>
00005 #include <stdexcept> // For std::invalid_argument
00006
00007 enum class EntityType
00008 {
00009     BUSSTOP,
00010     TRAINSTATION,
00011     AIRPORT,
00012     OFFICE,
00013     SHOPPINGMALL,
00014     FACTORY,
00015     HOUSE,
00016     APARTMENT,
00017     HOSPITAL,
00018     POLICESTATION,
00019     SCHOOL,
00020     PARK,
00021     THEATER,
00022     MONUMENT,
00023     POWERPLANT,
00024     WATERSUPPLY,
00025     WASTEMANAGEMENT,
00026     SEWAGESYSTEM,
00027     ROAD,
00028     WOODPRODUCER,
00029     STONEPRODUCER,
00030     CONCRETEPRODUCER,
00031     UNKNOWN,
00032 };
00033
00039 inline std::string entityTypeToString(EntityType type)
00040 {
00041     switch (type)
00042     {
00043     case EntityType::BUSSTOP:
00044         return "BUSSTOP";
00045     case EntityType::TRAINSTATION:
00046         return "TRAINSTATION";
00047     case EntityType::AIRPORT:
00048         return "AIRPORT";
00049     case EntityType::OFFICE:
00050         return "OFFICE";
00051     case EntityType::SHOPPINGMALL:
00052         return "SHOPPINGMALL";
00053     case EntityType::FACTORY:
00054         return "FACTORY";
00055     case EntityType::HOUSE:
00056         return "HOUSE";
00057     case EntityType::APARTMENT:
00058         return "APARTMENT";
00059     case EntityType::HOSPITAL:
00060         return "HOSPITAL";
00061     case EntityType::POLICESTATION:
00062         return "POLICESTATION";
00063     case EntityType::SCHOOL:
00064         return "SCHOOL";
00065     case EntityType::PARK:
00066         return "PARK";
00067     case EntityType::THEATER:
00068         return "THEATER";
00069     case EntityType::MONUMENT:
00070         return "MONUMENT";
00071     case EntityType::POWERPLANT:
00072         return "POWERPLANT";
```

```
00073     case EntityType::WATERSUPPLY:
00074         return "WATERSUPPLY";
00075     case EntityType::WASTEMANAGEMENT:
00076         return "WASTEMANAGEMENT";
00077     case EntityType::SEWAGESYSTEM:
00078         return "SEWAGESYSTEM";
00079     case EntityType::ROAD:
00080         return "ROAD";
00081     case EntityType::WOODPRODUCER:
00082         return "WOODPRODUCER";
00083     case EntityType::STONEPRODUCER:
00084         return "STONEPRODUCER";
00085     case EntityType::CONCRETEPRODUCER:
00086         return "CONCRETEPRODUCER";
00087     default:
00088         return "Unknown Entity";
00089     }
00090 }
00091
00098 inline EntityType stringToEntityType(const std::string &typeStr)
00099 {
00100     if (typeStr == "BUSSTOP")
00101         return EntityType::BUSSTOP;
00102     else if (typeStr == "TRAINSTATION")
00103         return EntityType::TRAINSTATION;
00104     else if (typeStr == "AIRPORT")
00105         return EntityType::AIRPORT;
00106     else if (typeStr == "OFFICE")
00107         return EntityType::OFFICE;
00108     else if (typeStr == "SHOPPINGMALL")
00109         return EntityType::SHOPPINGMALL;
00110     else if (typeStr == "FACTORY")
00111         return EntityType::FACTORY;
00112     else if (typeStr == "HOUSE")
00113         return EntityType::HOUSE;
00114     else if (typeStr == "APARTMENT")
00115         return EntityType::APARTMENT;
00116     else if (typeStr == "HOSPITAL")
00117         return EntityType::HOSPITAL;
00118     else if (typeStr == "POLICESTATION")
00119         return EntityType::POLICESTATION;
00120     else if (typeStr == "SCHOOL")
00121         return EntityType::SCHOOL;
00122     else if (typeStr == "PARK")
00123         return EntityType::PARK;
00124     else if (typeStr == "THEATER")
00125         return EntityType::THEATER;
00126     else if (typeStr == "MONUMENT")
00127         return EntityType::MONUMENT;
00128     else if (typeStr == "POWERPLANT")
00129         return EntityType::POWERPLANT;
00130     else if (typeStr == "WATERSUPPLY")
00131         return EntityType::WATERSUPPLY;
00132     else if (typeStr == "WASTEMANAGEMENT")
00133         return EntityType::WASTEMANAGEMENT;
00134     else if (typeStr == "SEWAGESYSTEM")
00135         return EntityType::SEWAGESYSTEM;
00136     else if (typeStr == "ROAD")
00137         return EntityType::ROAD;
00138     else if (typeStr == "WOODPRODUCER")
00139         return EntityType::WOODPRODUCER;
00140     else if (typeStr == "STONEPRODUCER")
00141         return EntityType::STONEPRODUCER;
00142     else if (typeStr == "CONCRETEPRODUCER")
00143         return EntityType::CONCRETEPRODUCER;
00144     else
00145         throw std::invalid_argument("Invalid entity type string: " + typeStr);
00146 }
00147
00148 #endif // ENTITYTYPE_H
```

5.157 Memento.h

```
00001 #ifndef MEMENTO_H
00002 #define MEMENTO_H
00003
00004 #include <string>
00005
00010 class Memento {
00011 private:
00012     std::string name;
00013     std::string detail;
00014
```

```
00015 public:
00022     Memento(const std::string& name, const std::string& detail);
00023
00029     std::string getName() const;
00030
00036     std::string getDetail() const;
00037
00043     void setName(const std::string& name);
00044
00050     void setDetail(const std::string& detail);
00051 };
00052
00053 #endif
```

5.158 Menu.h

```
00001 #ifndef MENU_H
00002 #define MENU_H
00003
00009 enum class Menu
00010 {
00011     MAIN,
00012     BUILDINGS,
00013     UPGRADES,
00014     POLICY,
00015     TAX,
00016     DISPLAYCITY,
00017     BUY_AMENITY,
00018     BUY_ECONOMIC_BUILDING,
00019     BUY_RESIDENTIAL_BUILDING,
00020     BUY_TRANSPORT,
00021     BUY_UTILITY,
00022     BUY_RESOURCE,
00023     BUY_SERVICE,
00024     BUY_ROAD,
00025     DEMOLISH,
00026     STATS,
00027
00028 };
00029
00030 #endif // MENU_H
```

5.159 PolicyType.h

```
00001 #ifndef POLICYTYPE_H
00002 #define POLICYTYPE_H
00003
00011 enum class PolicyType {
00012     LOW_WATER_POLICY,
00013     NORMAL_WATER_POLICY,
00014     HIGH_WATER_POLICY,
00015     LOW_ELECTRICITY_POLICY,
00016     NORMAL_ELECTRICITY_POLICY,
00017     HIGH_ELECTRICITY_POLICY
00018 };
00019
00020 #endif // POLICYTYPE_H
```

5.160 SatisfactionConfig.h

```
00001 #ifndef SATISFACTIONCONFIG_H
00002 #define SATISFACTIONCONFIG_H
00003
00004 struct SatisfactionConfig
00005 {
00006     float localRate;
00007     float globalRate;
00008     float localExtreme;
00009     float globalExtreme;
00010
00011     SatisfactionConfig()
00012         : localRate(0.0f), globalRate(0.0f), localExtreme(0.0f), globalExtreme(0.0f) {}
00013
00014     SatisfactionConfig(float localRate = 0.0f, float globalRate = 0.0f, float localExtreme = 0.0f,
      float globalExtreme = 0.0f)
```

```
00015         : localRate(localRate), globalRate(globalRate), localExtreme(localExtreme),
      globalExtreme(globalExtreme) {}
00016 };
00017
00018 #endif // SATISFACTIONCONFIG_H
```

5.161 Size.h

```
00001 #ifndef SIZE_H
00002 #define SIZE_H
00003
00004 #include <string>
00005 #include <stdexcept> // For std::invalid_argument
00006
00011 enum class Size
00012 {
00013     SMALL,
00014     MEDIUM,
00015     LARGE,
00016 };
00017
00024 inline std::string sizeToString(Size size)
00025 {
00026     switch (size)
00027     {
00028     case Size::SMALL:
00029         return "SMALL";
00030     case Size::MEDIUM:
00031         return "MEDIUM";
00032     case Size::LARGE:
00033         return "LARGE";
00034     default:
00035         throw std::invalid_argument("Invalid Size enum value");
00036     }
00037 }
00038
00045 inline Size stringToSize(const std::string &sizeStr)
00046 {
00047     if (sizeStr == "SMALL")
00048         return Size::SMALL;
00049     else if (sizeStr == "MEDIUM")
00050         return Size::MEDIUM;
00051     else if (sizeStr == "LARGE")
00052         return Size::LARGE;
00053     else
00054         throw std::invalid_argument("Invalid size string: " + sizeStr);
00055 }
00056
00057 #endif // SIZE_H
```

5.162 CityVisitor.h

```
00001 #ifndef CITYVISITOR_H
00002 #define CITYVISITOR_H
00003
00004 #include <vector>
00005 #include "entities/base/Entity.h"
00006
00007 class City; // Forward declaration to avoid circular dependency
00008
00012 class CityVisitor
00013 {
00014 public:
00018     CityVisitor() = default;
00019
00023     virtual ~CityVisitor() = default;
00024
00029     virtual void visit(City *city) = 0;
00030 };
00031
00032 #endif // CITYVISITOR_H
```

5.163 PopulationVisitor.h

```
00001 #ifndef POPULATIONVISITOR_H
```

```
00002 #define POPULATIONVISITOR_H
00003
00004 #include "visitors/base/CityVisitor.h"
00005 #include "city/City.h"
00006
00010 class PopulationVisitor : public CityVisitor
00011 {
00012 private:
00013     int totalPopulationCapacity;
00014     int totalWaterConsumption;
00015     int totalElectricityConsumption;
00016     int housePopulationCapacity;
00017     int apartmentPopulationCapacity;
00019 public:
00023     PopulationVisitor();
00024
00028     ~PopulationVisitor() {}
00029
00034     void visit(City *city) override;
00035
00040     int getTotalPopulationCapacity() const;
00041
00046     int getHousePopulationCapacity() const;
00047
00052     int getTotalWaterConsumption() const;
00053
00058     int getTotalElectricityConsumption() const;
00059
00064     int getApartmentPopulationCapacity() const;
00065 };
00066
00067 #endif // POPULATIONVISITOR_H
```

5.164 ResourceVisitor.h

```
00001 #ifndef RESOURCEVISITOR_H
00002 #define RESOURCEVISITOR_H
00003
00004 #include "visitors/base/CityVisitor.h"
00005 #include "entities/industry/concreteproducer/ConcreteProducer.h"
00006 #include "entities/industry/stoneproducer/StoneProducer.h"
00007 #include "entities/industry/woodproducer/WoodProducer.h"
00008
00012 class ResourceVisitor : public CityVisitor
00013 {
00014 private:
00015     int totalWood;
00016     int totalConcrete;
00017     int totalStone;
00019 public:
00023     ResourceVisitor();
00024
00028     ~ResourceVisitor();
00029
00034     void visit(City *city) override;
00035
00040     int getTotalWood() const { return totalWood; }
00041
00046     int getTotalConcrete() const { return totalConcrete; }
00047
00052     int getTotalStone() const { return totalStone; }
00053 };
00054
00055 #endif // RESOURCEVISITOR_H
```

5.165 SatisfactionVisitor.h

```
00001 #ifndef SATISFACTIONVISITOR_H
00002 #define SATISFACTIONVISITOR_H
00003
00004 #include "visitors/base/CityVisitor.h"
00005 #include "city/City.h"
00006
00010 class SatisfactionVisitor : public CityVisitor
00011 {
00012 private:
00013     float totalSatisfaction;
00014     int residentialCount;
00016 public:
```

```
00020     SatisfactionVisitor();
00021
00025     ~SatisfactionVisitor() {}
00026
00031     void visit(City *city) override;
00032
00037     float getAverageSatisfaction() const;
00038
00043     int getResidentialCount() const;
00044 };
00045
00046 #endif // SATISFACTIONVISITOR_H
```

5.166 TaxCalculationVisitor.h

```
00001 #ifndef TAXCALCULATIONVISITOR_H
00002 #define TAXCALCULATIONVISITOR_H
00003
00004 #include "visitors/base/CityVisitor.h"
00005 #include "city/City.h"
00006 #include "entities/building/residential/ResidentialBuilding.h"
00007 #include "entities/building/economic/EconomicBuilding.h"
00008
00012 class TaxCalculationVisitor : public CityVisitor
00013 {
00014 private:
00015     int totalResidentialTax;
00016     int totalEconomicTax;
00018 public:
00022     TaxCalculationVisitor();
00023
00027     ~TaxCalculationVisitor();
00028
00033     void visit(City *city) override;
00034
00039     int getTotalResidentialTax() const { return totalResidentialTax; }
00040
00045     int getTotalEconomicTax() const { return totalEconomicTax; }
00046
00051     int getTotalTax() const { return totalResidentialTax + totalEconomicTax; }
00052 };
00053
00054 #endif // TAXCALCULATIONVISITOR_H
```

5.167 UtilityVisitor.h

```
00001 #ifndef UTILITYVISITOR_H
00002 #define UTILITYVISITOR_H
00003
00004 #include "visitors/base/CityVisitor.h"
00005 #include "entities/utility/base/Utility.h"
00006
00011 class UtilityVisitor : public CityVisitor
00012 {
00013 private:
00014     int totalElectricity;
00015     int totalWater;
00016     int totalSewageHandled;
00017     int totalWasteHandled;
00019 public:
00023     UtilityVisitor();
00024
00028     virtual ~UtilityVisitor();
00029
00034     void visit(City *city) override;
00035
00040     int getTotalElectricity() const;
00041
00046     int getTotalWater() const;
00047
00052     int getTotalSewageHandled() const;
00053
00058     int getTotalWasteHandled() const;
00059 };
00060
00061 #endif // UTILITYVISITOR_H
```