Politechnika Śląska
Wydział Informatyki, Elektroniki i Informatyki

Podstawy Programowania Komputerów

temat projektu:

Symulator Miasta

autor	Eustachy Lisiński
prowadzący	mgr inż. Maciej Długosz
rok akademicki	2021/2022
kierunek	informatyka
rodzaj studiów	SSI
semestr	2
termin laboratorium	środa, 15:30– 17:00
sekcja	12
termin oddania sprawozdania	2022-06-22

1 Treść zadania

Proszę napisać program będący symulatorem miasta. Zadaniem gracza jest wytyczanie dróg, wyznaczanie stref mieszkalnych oraz stref do budowy zakładów pracy, a także specjalnych budynków komunalnych (np. szkół). Różne strefy posiadają różne parametry (np. różna wartość ziemi w strefach mieszkalnych (typu liczba rzeczywista) albo liczba osób mogących zostać zatrudnionych w danej strefie (liczba naturalna)). Program powinien implementować prosty budżet (obejmujący np. podatki, koszty budowy i utrzymania infrastruktury) oraz kilka (wybieranych przez użytkownika albo losowo) algorytmów wyznaczających zainteresowanie mieszkańców zamieszkaniem w mieście.

2 Opis projektu

Zrealizowany program realizuje większość z podanych wytycznych w mniej bądź większej mierze. Główne założenia zostały zrealizowane, symulator pozwala na tworzenie różnego rodzaju budowli, drogi pozwalają budynkom komunalnym na wpływanie na tereny mieszkalne, miejsca pracy zatrudniają ograniczoną ilość osób oraz zawiera w pełni funkcjonujący budżet.

Natomiast jedynym algorytmem zainteresowania jest system „jakości życia", który zwiększa ilość ludności w budynkach które mają dostęp do budynków komunalnych. Jednakże jest to niedopracowany system i wymagał by przebudowania.

Dodatkowym problemem jest fakt że gra nie posiada warunków ukończenia.

2.1 Dodatkowe Pliki

Program do działania wymaga pliku Settings.ini który zawiera różne dane wejściowe konfigurujące działanie programu oraz folderów z plikami związanymi z SDL2.

3 Wnioski

Ten projekt był bardzo wymagający i gdyby nie brak czasu mógłbym go bardziej dopracować. Był to pierwszy pełnowymiarowy program obiektowy jaki pisałem, przez co musiałem się dużo nauczyć na bieżąco, a poprawna implementacja często sprawiała duże problemy. W szczególności że błędy konsolidatora zdarzają się dużo częściej a są znacząco trudniejsze do rozwiązania.

Sama dynamika programu sprawiała że wykrycie niektórych błędów było wyjątkowo trudne, szczególnie że niektóre pojawiały się dopiero w specyficznych warunkach.

Uważam że ten projekt poszerzył moje umiejętności i pomógł mi utrwalić wiedzę związaną z programowaniem obiektowym. Dodatkowo wykorzystanie biblioteki graficznej było dla mnie nowym wyzwaniem.

Program w swoim aktualnym stanie ma duży potencjał na rozbudowę, implementacja nowych obiektów i mechanik jest prosta ponieważ obecna architektura zależności pozawala na łatwe dodawanie nowych elementów.

Kod zawiera wiele funkcji i linii kodu które mogły by zostać usprawnione,

4 Testowanie i błędy

Program był w miarę dogłębnie testowany i na moment pisania tego sprawozdania występują dwa znane i nierozwiązane błędy:

- Usunięcie obszaru przy pomocy ERASE i próba budowania na nim kończy się naruszeniem dostępu do odczytu.
- Program z biegiem czasu samoistnie zwiększa zapotrzebowanie na pamięć, przyczyna nie jest znana, ale podejrzewam że jakiś obiekt jest wielokrotnie tworzony i nie niszczony.

5 Bibliografia

[1] Poradnik tworzenia gier z użyciem SDL 2
https://www.youtube.com/watch?v=QQzAHcojEKg&list=PLhfAbcv9cehhkG7ZQK0nfIGJC_C-wSLrx

[2] Linki do pobrania biblioteki:

SDL2 https://www.libsdl.org/download-2.0.php

SDL2_image https://www.libsdl.org/projects/SDL_image/

SDL2_ttf https://github.com/libsdl-org/SDL_ttf/tree/main/VisualC (wymaga samodzielnego złożenia)

[3] Strony z których korzystałem do rozwiązywania błędów

https://www.geeksforgeeks.org/

https://stackoverflow.com

https://www.tutorialspoint.com/cplusplus/index.htm

https://refactoring.guru/pl/design-patterns/catalog

Symulator miasta

Wygenerowano przez Doxygen 1.9.3

Rozdział 1

Indeks hierarchiczny

1.1 Hierarchia klas

Ta lista dziedziczenia posortowana jest z grubsza, choć nie całkowicie, alfabetycznie:

Rozdział 2

Indeks klas

2.1 Lista klas

Tutaj znajdują się klasy, struktury, unie i interfejsy wraz z ich krótkimi opisami:

Rozdział 3

Indeks plików

3.1 Lista plików

Tutaj znajduje się lista wszystkich udokumentowanych plików z ich krótkimi opisami:

Rozdział 4

Dokumentacja klas

4.1 Dokumentacja klasy Budget

Klasa Budzetu wykorzystujaca Singleton.

`#include <Budget.h>`

Diagram dziedziczenia dla Budget

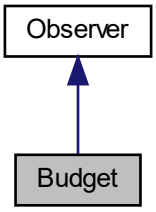
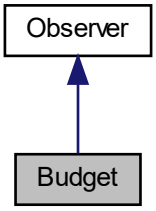


Diagram współpracy dla Budget:



Metody publiczne

- **Budget** (Budget &other)=delete
- void **operator=** (const Budget &)=delete
- void tickUpdate ()

  funkcja przyslaniajaca Observer::tickUpdate().
- void mTickUpdate ()

  funkcja przyslaniajaca Observer::mTickUpdate(). Dodaje saldo do skarbca, wywoluje recount() i recountPop().
- int addBuilding (Buildings ∗b)

  dodaje budynek do mapy rozliczeniowej, zwraca unikalny identyfikator.
- void removeBuilding (int id)

  Usuwa budynek z mapy rozliczeniowej.
- int addPop (Residential ∗b)

  dodaje obszar mieszkalny do mapy rozliczeniowej, zwraca unikalny identyfikator.
- void removePop (int id)

  Usuwa obszar mieszkalny z mapy rozliczeniowej.
- int addWork (Workspace ∗b)

  dodaje miejsce pracy do mapy rozliczeniowej, zwraca unikalny identyfikator.
- void removeWork (int id)

  Usuwa miejsce pracy z mapy rozliczeniowej.
- int getTreasury ()

  Zwraca stan skarbca.
- int getBalance ()

  Zwraca aktualne saldo.
- int getPop ()

  Zwraca liczbe ludnosci.

Statyczne metody publiczne

- static Budget & GetInstance ()

  Statyczna funkcja zwracajaca instancje budzetu.

4.1.1 Opis szczegółowy

Klasa Budzetu wykorzystujaca Singleton.

4.1.2 Dokumentacja funkcji składowych

4.1.2.1 addBuilding()

```
int Budget::addBuilding (
            Buildings * b )
```

dodaje budynek do mapy rozliczeniowej, zwraca unikalny identyfikator.

Zwraca

int

4.1.2.2 addPop()

```
int Budget::addPop (
            Residential * b )
```

dodaje obszar mieszkalny do mapy rozliczeniowej, zwraca unikalny identyfikator.

Zwraca

int

4.1.2.3 addWork()

```
int Budget::addWork (
            Workspace * b )
```

dodaje miejsce pracy do mapy rozliczeniowej, zwraca unikalny identyfikator.

Zwraca

int

4.1.2.4 getBalance()

```
int Budget::getBalance ( )
```

Zwraca aktualne saldo.

Zwraca

int

4.1.2.5 GetInstance()

```
Budget & Budget::GetInstance ( )  [static]
```

Statyczna funkcja zwracajaca instancje budzetu.

Zwraca

Budget&

4.1.2.6 getPop()

```
int Budget::getPop ( )
```

Zwraca liczbe ludnosci.

Zwraca

int

4.1.2.7 getTreasury()

```
int Budget::getTreasury ( )
```

Zwraca stan skarbca.

Zwraca

int

4.1.2.8 mTickUpdate()

```
void Budget::mTickUpdate ( )  [virtual]
```

funkcja przyslaniajaca Observer::mTickUpdate(). Dodaje saldo do skarbca, wywoluje recount() i recountPop().

Implementuje Observer.

4.1.2.9 removeBuilding()

```
void Budget::removeBuilding (
            int id )
```

Usuwa budynek z mapy rozliczeniowej.

Parametry

id	Identyfikator

4.1.2.10 removePop()

```
void Budget::removePop (
            int id )
```

Usuwa obszar mieszkalny z mapy rozliczeniowej.

Parametry

id	Identyfikator

4.1.2.11 removeWork()

```
void Budget::removeWork (
            int id )
```

Usuwa miejsce pracy z mapy rozliczeniowej.

Parametry

id	Identyfikator

4.1.2.12 tickUpdate()

```
void Budget::tickUpdate ( )  [virtual]
```

funkcja przyslaniajaca Observer::tickUpdate().

Implementuje Observer.

Dokumentacja dla tej klasy została wygenerowana z plików:

- Symulator miasta/Symulator miasta/Budget.h
- Symulator miasta/Symulator miasta/Budget.cpp

4.2 Dokumentacja szablonu klasy Builder< T >

Szablonowa klasa budowniczego. Decyduje gdzie i jakie pola nalezy postawic na podstawie dzialan uzytkownika T Moze byc dowolna niewirtualna klasa polimorfizowalna do Tile.

```
#include <Builder.h>
```

Metody publiczne

- Builder (Grid *grid, int mode)

  Konstruktor.
- bool mouseDown (std::pair< int, int > cords)

  funkcja zapisuje kordynaty poczatkowe. dodatkowo sprawdza czy gracz poprawnie zaznaczyl miejsce poczatkowe
- bool mouseUp (std::pair< int, int > cords)

  funkcja zapisuje kordynaty koncowe. sprawdza czy gracz poprawnie zaznaczyl miejsce koncowe wywoluje metode build
- std::map< int, Tile * > Adjecent (int i, int j)

  funkcja spisuje sasiednie kratki
- void build (std::pair< int, int > cords)

  funkcja Wyznacza obszar budowania i wywoluje Grid::build

4.2.1 Opis szczegółowy

template<class T>
class Builder< T >

Szablonowa klasa budowniczego. Decyduje gdzie i jakie pola nalezy postawic na podstawie dzialan uzytkownika T Moze byc dowolna niewirtualna klasa polimorfizowalna do Tile.

4.2.2 Dokumentacja konstruktora i destruktora

4.2.2.1 Builder()

```
template<class T >
Builder< T >::Builder (
            Grid * grid,
            int mode )  [inline]
```

Konstruktor.

Parametry

grid	wskaznik siatki
mode	tryb

4.2.3 Dokumentacja funkcji składowych

4.2.3.1 Adjecent()

```
template<class T >
std::map< int, Tile * > Builder< T >::Adjecent (
            int i,
            int j )  [inline]
```

funkcja spisuje sasiednie kratki

Parametry

i	wiersz kratki
j	kolumna kratki

Zwraca

std::map<int, Tile*>

4.2.3.2 build()

```
template<class T >
void Builder< T >::build (
            std::pair< int, int > cords )  [inline]
```

funkcja Wyznacza obszar budowania i wywoluje Grid::build

Parametry

cords	kordynaty

4.2.3.3 mouseDown()

```
template<class T >
bool Builder< T >::mouseDown (
            std::pair< int, int > cords )  [inline]
```

funkcja zapisuje kordynaty poczatkowe. dodatkowo sprawdza czy gracz poprawnie zaznaczyl miejsce poczatkowe

Parametry

cords	kordynaty

Zwraca

bool

4.2.3.4 mouseUp()

```
template<class T >
bool Builder< T >::mouseUp (
            std::pair< int, int > cords )  [inline]
```

funkcja zapisuje kordynaty koncowe. sprawdza czy gracz poprawnie zaznaczyl miejsce koncowe wywoluje metode build

Parametry

cords	kordynaty

Zwraca

bool

Dokumentacja dla tej klasy została wygenerowana z pliku:

- Symulator miasta/Symulator miasta/Builder.h

4.3 Dokumentacja klasy Buildings

Klasa posrednia reprezentujaca budynki.

```
#include <Buildings.h>
```

Diagram dziedziczenia dla Buildings

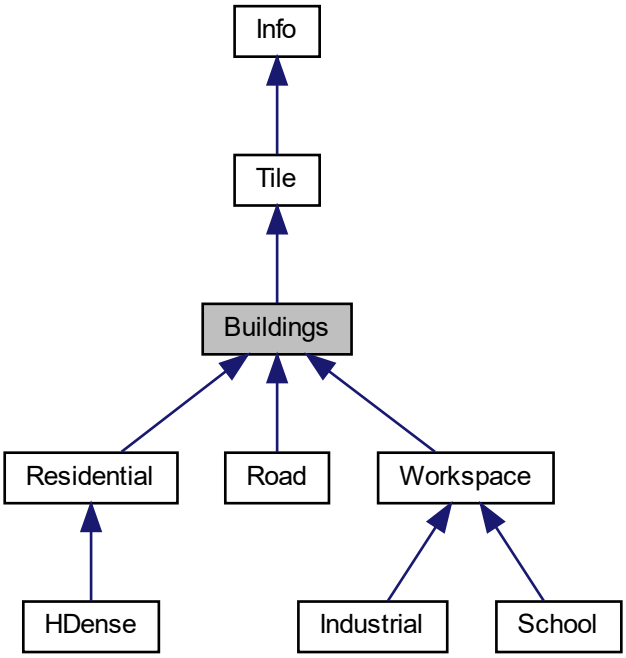
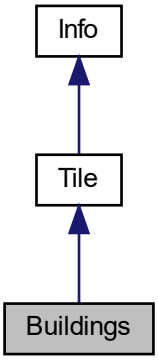


Diagram współpracy dla Buildings:



Metody publiczne

- **Buildings** (int maintenance, int cost, int tex, std::map< int, Tile * > adjacent)

- int getBalance ()

  Zwraca przychod - utrzymanie.
- int getCost ()

  Getter dla cost.
- void addObserving (Communal ∗b)

  dodaje obiekt do obserwowanych.
- void removeObserving (Communal ∗b)

  usuwa obiekt z obserwowanych.

Atrybuty chronione

- std::set< Communal ∗ > observing
- std::set< Buildings ∗ > observers
- int maintenance
- int revenue

4.3.1 Opis szczegółowy

Klasa posrednia reprezentujaca budynki.

4.3.2 Dokumentacja funkcji składowych

4.3.2.1 addObserving()

```
void Buildings::addObserving (
            Communal * b )
```

dodaje obiekt do obserwowanych.

Parametry

b	wskaznik obiektu

4.3.2.2 getBalance()

```
int Buildings::getBalance ( )
```

Zwraca przychod - utrzymanie.

Zwraca

int

4.3.2.3 getCost()

```
int Buildings::getCost ( )
```

Getter dla cost.

Zwraca

int

4.3.2.4 removeObserving()

```
void Buildings::removeObserving (
            Communal * b )
```

usuwa obiekt z obserwowanych.

Parametry

b	wskaznik obiektu

4.3.3 Dokumentacja atrybutów składowych

4.3.3.1 maintenance

```
int Buildings::maintenance  [protected]
```

koszt utrzymania

4.3.3.2 observers

```
std::set<Buildings*> Buildings::observers  [protected]
```

zbior obiektow obserwujacych

4.3.3.3 observing

```
std::set<Communal*> Buildings::observing  [protected]
```

zbior obiektow obserwowanych

4.3.3.4 revenue

```
int Buildings::revenue  [protected]
```

przychod

Dokumentacja dla tej klasy została wygenerowana z plików:

- Symulator miasta/Symulator miasta/Buildings.h
- Symulator miasta/Symulator miasta/Buildings.cpp

4.4 Dokumentacja szablonu klasy Button< T >

Szablonowa klasa reprezentujaca przycisk. posredniczy miedzy akcjami gracza, a budowniczym Wszystkie metody sa przeslaniajace.

```
#include <Button.h>
```

Diagram dziedziczenia dla Button< T >

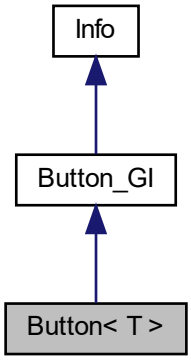
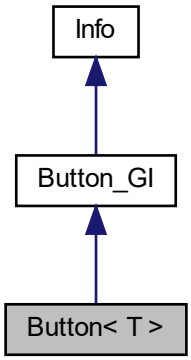


Diagram współpracy dla Button< T >:



Metody publiczne

- **Button** (int texId, std::pair< int, int > position)
- void activate (Grid *grid)

  Czysto wirtualna metoda wywolywana przy kliknieciu na przycisk.
- void deactivate ()

  Czysto wirtualna metoda wylaczajaca przycisk.
- bool mouseDown (std::pair< int, int > cords)

  Czysto wirtualna metoda wywolywujaca Builder< T >::mouseDown.
- bool mouseUp (std::pair< int, int > cords)

  Czysto wirtualna metoda wywolywujaca Builder< T >::mouseUp.

Dodatkowe Dziedziczone Składowe

4.4.1 Opis szczegółowy

template<class T>
class Button< T >

Szablonowa klasa reprezentujaca przycisk. posredniczy miedzy akcjami gracza, a budowniczym Wszystkie metody sa przeslaniajace.

4.4.2 Dokumentacja funkcji składowych

4.4.2.1 activate()

```
template<class T >
void Button< T >::activate (
            Grid * grid )  [inline], [virtual]
```

Czysto wirtualna metoda wywolywana przy kliknieciu na przycisk.

Parametry

grid	

Implementuje Button_GI.

4.4.2.2 deactivate()

```
template<class T >
void Button< T >::deactivate ( )  [inline], [virtual]
```

Czysto wirtualna metoda wylaczajaca przycisk.

Implementuje Button_GI.

4.4.2.3 mouseDown()

```
template<class T >
bool Button< T >::mouseDown (
            std::pair< int, int > cords )  [inline], [virtual]
```

Czysto wirtualna metoda wywolywujaca Builder<T>::mouseDown.

Parametry

cords	kordynaty

Zwraca

bool

Implementuje Button_GI.

4.4.2.4 mouseUp()

```
template<class T >
bool Button< T >::mouseUp (
            std::pair< int, int > cords )  [inline], [virtual]
```

Czysto wirtualna metoda wywolywujaca Builder<T>::mouseUp.

Parametry

cords	kordynaty

Zwraca

bool

Implementuje Button_GI.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- Symulator miasta/Symulator miasta/Button.h

4.5 Dokumentacja klasy Button_GI

Klasa opakowujaca szablonowa klase Button<T>.

`#include <Button_GI.h>`

Diagram dziedziczenia dla Button_GI

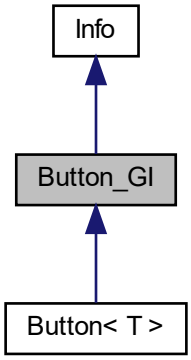
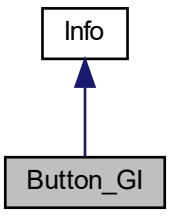


Diagram współpracy dla Button_GI:



Metody publiczne

- **Button_GI** (int texId, std::pair< int, int > position)
- std::pair< int, int > getPosition ()

  getter dla position.
- int getTex ()

  getter texId.
- virtual void activate (Grid *grid)=0

  Czysto wirtualna metoda wywolywana przy kliknieciu na przycisk.
- virtual void deactivate ()=0

  Czysto wirtualna metoda wylaczajaca przycisk.
- virtual bool mouseDown (std::pair< int, int > cords)=0

  Czysto wirtualna metoda wywolywujaca Builder< T >::mouseDown.
- virtual bool mouseUp (std::pair< int, int > cords)=0

  Czysto wirtualna metoda wywolywujaca Builder< T >::mouseUp.

Atrybuty chronione

- bool active
- std::pair< int, int > position

4.5.1 Opis szczegółowy

Klasa opakowujaca szablonowa klase Button<T>.

4.5.2 Dokumentacja funkcji składowych

4.5.2.1 activate()

```
virtual void Button_GI::activate (
            Grid * grid )  [pure virtual]
```

Czysto wirtualna metoda wywolywana przy kliknieciu na przycisk.

Parametry

grid	

Implementowany w Button< T >.

4.5.2.2 deactivate()

```
virtual void Button_GI::deactivate ( )  [pure virtual]
```

Czysto wirtualna metoda wylaczajaca przycisk.

Implementowany w Button< T >.

4.5.2.3 getPosition()

```
std::pair< int, int > Button_GI::getPosition ( )
```

getter dla position.

Zwraca

std::pair<int, int>

4.5.2.4 getTex()

```
int Button_GI::getTex ( )
```

getter texId.

Zwraca

int

4.5.2.5 mouseDown()

```
virtual bool Button_GI::mouseDown (
            std::pair< int, int > cords )  [pure virtual]
```

Czysto wirtualna metoda wywolywujaca Builder<T>::mouseDown.

Parametry

cords	kordynaty

Zwraca

bool

Implementowany w Button< T >.

4.5.2.6 mouseUp()

```
virtual bool Button_GI::mouseUp (
            std::pair< int, int > cords )  [pure virtual]
```

Czysto wirtualna metoda wywolywujaca Builder<T>::mouseUp.

Parametry

cords	kordynaty

Zwraca

bool

Implementowany w Button< T >.

4.5.3 Dokumentacja atrybutów składowych

4.5.3.1 active

```
bool Button_GI::active  [protected]
```

Status przycisku

4.5.3.2 position

```
std::pair<int, int> Button_GI::position  [protected]
```

Pozycja na ktorej znajduje sie przycisk

Dokumentacja dla tej klasy została wygenerowana z plików:

- Symulator miasta/Symulator miasta/Button_GI.h
- Symulator miasta/Symulator miasta/Button_GI.cpp

4.6 Dokumentacja klasy Clock

Zegar taktujacy dzialanie gry. laczy wzorzec Singletona oraz Observatora Zawiera dwa rozne rodzaje obserwatorow: czestych, ktorzy zostaja informowani co takt oraz rzadkich, informowani sa co takt znaczacy. Takty znaczace wystepuja co kilka taktow zwyklych i sluza glownie do obliczen ekonomicznych.

```
#include <Clock.h>
```

Metody publiczne

- **Clock** (Clock &other)=delete

  Destruktor.
- void **operator=** (const Clock &)=delete
- void **tick** ()

  Funkcja zawierajaca petle taktujaca.
- void newOb (Observer ∗ob)

  Dodanie nowego czestego obserwatora.
- void newMajorOb (Observer ∗ob)

  Dodanie nowego zadkiego obserwatora.
- void deleteOb (Observer ∗ob)

  Usuwa czestego obserwatora.
- void deleteMajorOb (Observer ∗ob)

  Usuwa zadkiego obserwatora.
- void tickInform ()

  Informuje zadkich obserwatorow.
- void **majorTickInform** ()

  Usuwa nowych obserwatorow.
- void **start** ()

  Rozpoczyna prace zegara.
- void **stop** ()

  Zatrzymuje prace zegara.

Statyczne metody publiczne

- static Clock & GetInstance ()

  Statyczna funkcja zwracajaca instancje zegara.

4.6.1 Opis szczegółowy

Zegar taktujacy dzialanie gry. laczy wzorzec Singletona oraz Observatora Zawiera dwa rozne rodzaje obserwatorow: czestych, ktorzy zostaja informowani co takt oraz rzadkich, informowani sa co takt znaczacy. Takty znaczace wystepuja co kilka taktow zwyklych i sluza glownie do obliczen ekonomicznych.

4.6.2 Dokumentacja funkcji składowych

4.6.2.1 deleteMajorOb()

```
void Clock::deleteMajorOb (
            Observer * ob )
```

Usuwa zadkiego obserwatora.

Parametry

ob	wskaznik obserwatora

4.6.2.2 deleteOb()

```
void Clock::deleteOb (
            Observer * ob )
```

Usuwa czestego obserwatora.

Parametry

ob	wskaznik obserwatora

4.6.2.3 GetInstance()

```
Clock & Clock::GetInstance ( )  [static]
```

Statyczna funkcja zwracajaca instancje zegara.

Zwraca

Clock&

4.6.2.4 newMajorOb()

```
void Clock::newMajorOb (
            Observer * ob )
```

Dodanie nowego zadkiego obserwatora.

Parametry

ob	wskaznik obserwatora

4.6.2.5 newOb()

```
void Clock::newOb (
```

```
Observer * ob )
```

Dodanie nowego czestego obserwatora.

Parametry

ob	wskaznik obserwatora

4.6.2.6 tickInform()

```
void Clock::tickInform ( )
```

Informuje zadkich obserwatorow.

Parametry

ob	wskaznik obserwatora

Dokumentacja dla tej klasy została wygenerowana z plików:

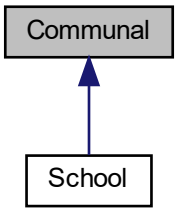
- Symulator miasta/Symulator miasta/Clock.h
- Symulator miasta/Symulator miasta/Clock.cpp

4.7 Dokumentacja klasy Communal

Klasa opakowujaca dla obiektow komunalnych.

```
#include <Communal.h>
```

Diagram dziedziczenia dla Communal



Metody publiczne

- **Communal** (float influance, unsigned short int range)
- float getInfluance ()

  Getter dla influance.
- unsigned short int getRange ()

  Getter dla zmiennej range.

4.7.1 Opis szczegółowy

Klasa opakowujaca dla obiektow komunalnych.

4.7.2 Dokumentacja funkcji składowych

4.7.2.1 getInfluance()

```
float Communal::getInfluance ( )
```

Getter dla influance.

Zwraca

float

4.7.2.2 getRange()

```
unsigned short int Communal::getRange ( )
```

Getter dla zmiennej range.

Zwraca

unsigned short int

Dokumentacja dla tej klasy została wygenerowana z plików:

- Symulator miasta/Symulator miasta/Communal.h
- Symulator miasta/Symulator miasta/Communal.cpp

4.8 Dokumentacja klasy Empty

Klasa pustej kratki. Wszystkie metody sa przyslaniajace dla metod klasy Tile posiada konstruktor bez tekstury, przyjmuje wtedy wartosc domyslna.

`#include <Empty.h>`

Diagram dziedziczenia dla Empty

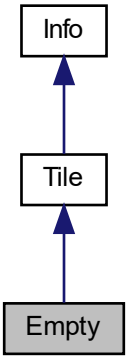
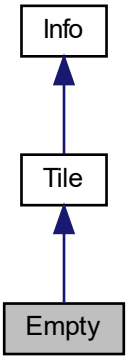


Diagram współpracy dla Empty:



Metody publiczne

- **Empty** (int texture, std::map< int, Tile * > adjacent)
- **Empty** (std::map< int, Tile * > adjacent)
- void changeTex (int tex)

setter dla texture.

- int getTex ()

  getter dla texture.

- void changeAdjecent (Tile ∗newAdj, int side)

  Dodaje sasiada.

- void removeAdjecent (int side)

  usuwa sasiada po konkretnej stronie.

- void update ()

  funkcja aktualizacji. dzialanie znaczaco rozni sie w zaleznosci od klasy potomnej

Dodatkowe Dziedziczone Składowe

4.8.1 Opis szczegółowy

Klasa pustej kratki. Wszystkie metody sa przyslaniajace dla metod klasy Tile posiada konstruktor bez tekstury, przyjmuje wtedy wartosc domyslna.

4.8.2 Dokumentacja funkcji składowych

4.8.2.1 changeAdjecent()

```
void Empty::changeAdjecent (
            Tile * newAdj,
            int side )  [virtual]
```

Dodaje sasiada.

Parametry

newAdj	wskaznik sasiad
side	strona

Reimplementowana z Tile.

4.8.2.2 changeTex()

```
void Empty::changeTex (
            int tex )  [virtual]
```

setter dla texture.

Parametry

tex	identyfikator nowej tekstury

Reimplementowana z Tile.

4.8.2.3 getTex()

```
int Empty::getTex ( )  [virtual]
```

getter dla texture.

Zwraca

int

Reimplementowana z Tile.

4.8.2.4 removeAdjecent()

```
void Empty::removeAdjecent (
            int side )  [virtual]
```

usuwa sasiada po konkretnej stronie.

Parametry

side	strona

Reimplementowana z Tile.

4.8.2.5 update()

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

funkcja aktualizacji. dzialanie znaczaco rozni sie w zaleznosci od klasy potomnej

Reimplementowana z Tile.

Dokumentacja dla tej klasy została wygenerowana z plików:

- Symulator miasta/Symulator miasta/Empty.h
- Symulator miasta/Symulator miasta/Empty.cpp

4.9 Dokumentacja klasy Game

Klasa inicjalizujaca wszystkie obiekty gry, i kontrolujaca ich dzialanie.

`#include <Game.h>`

Diagram dziedziczenia dla Game

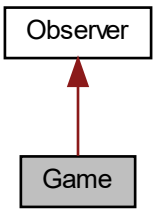
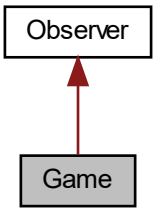


Diagram współpracy dla Game:



Metody publiczne

- **Game** ()
  Konstruktor.
- ∼**Game** ()
  Destruktor.
- void **events** ()
  Funkcja odpowiedzialna za obsluge interakcji uzytkownika.
- void **render** ()
  Funkcja przeprowadzajaca operacje zwiazane z wyswietlaniem.
- void **clean** ()
  Wywolywana przy konczeniu dzialania, zapewnia uporzadkowane niszczenie obiektow.
- SDL_Renderer ∗ **getRenderer** ()

Getter zmiennej "renderer".

- void renderVisibleArea ()

  Funkcja Wyznacza zakres pol do wyswietlenia.
- void mTickUpdate ()

  Funkcja przyslaniajaca Obserever::mTickUpdate().
- void tickUpdate ()

  Funkcja przyslaniajaca Obserever::tickUpdate().
- bool active ()

  Getter zmiennej "isActive".

4.9.1 Opis szczegółowy

Klasa inicjalizujaca wszystkie obiekty gry, i kontrolujaca ich dzialanie.

4.9.2 Dokumentacja funkcji składowych

4.9.2.1 active()

```
bool Game::active ( )
```

Getter zmiennej "isActive".

Zwraca

bool

4.9.2.2 mTickUpdate()

```
void Game::mTickUpdate ( )  [virtual]
```

Funkcja przyslaniajaca Obserever::mTickUpdate().

Implementuje Observer.

4.9.2.3 renderVisibleArea()

```
void Game::renderVisibleArea ( )
```

Funkcja Wyznacza zakres pol do wyswietlenia.

Zwraca

SDL_Renderer*

4.9.2.4 tickUpdate()

```
void Game::tickUpdate ( )  [virtual]
```

Funkcja przyslaniajaca Obserever::tickUpdate().

Implementuje Observer.

Dokumentacja dla tej klasy została wygenerowana z plików:

- Symulator miasta/Symulator miasta/Game.h
- Symulator miasta/Symulator miasta/Game.cpp

4.10 Dokumentacja klasy Grid

Klasa reprezentujaca dzialanie siatki. Wykonuje wszystkie operacje zwiazane z zmianami na siatce Zapobiega wyjsciu poza dozwolony obszar.

```
#include <Grid.h>
```

Metody publiczne

- Grid (int width, int height, int vWidth, int vHeight)

  Konstruktor.
- ∼**Grid** ()

  Destruktor. Niszczy wszystkie obiekty na siatce.
- void getCurrentRange (std::pair< int, int > &horizontal, std::pair< int, int > &vertical)

  Przekazuje przez referencjezakres siatki odpowiadajacy aktualnie wyswietlanemu obszarowi.
- Tile ∗ getTile (std::pair< int, int > cords)

  Zwraca wskaznik do pola pod pozycja.
- void moveOnMap (std::pair< int, int > movement)

  Zmienia centralna komorke w zaleznosci od ruchu.
- void build (std::pair< int, int > cords, Tile ∗newTile)

  Funkcja decuduje gdzie postawic nowy obiekt. Dziala wspolnie z funkcja replace, wywoluje ja po dokonaniu obliczen.
- void updateRange (std::pair< int, int > cords, std::pair< int, int > endcords)

  Wersja funkcji build dla budowania obszarowego. Nieuzywana.
- Tile ∗ getVisibleTile (std::pair< int, int > cords)

  Zwraca wskaznik do pola pod pozycja.

4.10.1 Opis szczegółowy

Klasa reprezentujaca dzialanie siatki. Wykonuje wszystkie operacje zwiazane z zmianami na siatce Zapobiega wyjsciu poza dozwolony obszar.

4.10.2 Dokumentacja konstruktora i destruktora

4.10.2.1 Grid()

```
Grid::Grid (
            int width,
            int height,
            int vWidth,
            int vHeight )
```

Konstruktor.

Parametry

width	szerokosc siatki.
height	wysokosc siatki.
vWidth	wysokosc widzianego obszaru.
vHeight	wysokosc widzianego obszaru.

4.10.3 Dokumentacja funkcji składowych

4.10.3.1 build()

```
void Grid::build (
            std::pair< int, int > cords,
            Tile * newTile )
```

Funkcja decuduje gdzie postawic nowy obiekt. Dziala wspolnie z funkcja replace, wywoluje ja po dokonaniu obliczen.

Parametry

cords	pozycja na widocznej czesci siatki.
newTile	obiekt do polozenia.

4.10.3.2 getCurrentRange()

```
void Grid::getCurrentRange (
            std::pair< int, int > & horizontal,
            std::pair< int, int > & vertical )
```

Przekazuje przez referencjezakres siatki odpowiadajacy aktualnie wyswietlanemu obszarowi.

Parametry

horizontal	pierwsza i ostatnia kolumna.
vertical	pierwszy i ostatni wiersz.

4.10.3.3 getTile()

```
Tile * Grid::getTile (
            std::pair< int, int > cords )
```

Zwraca wskaznik do pola pod pozycja.

Parametry

cords	pozycja na siatce.

Zwraca

Tile∗

4.10.3.4 getVisibleTile()

```
Tile * Grid::getVisibleTile (
            std::pair< int, int > cords )
```

Zwraca wskaznik do pola pod pozycja.

Parametry

cords	pozycja na widocznej siatce .

Zwraca

Tile∗

4.10.3.5 moveOnMap()

```
void Grid::moveOnMap (
            std::pair< int, int > movement )
```

Zmienia centralna komorke w zaleznosci od ruchu.

Parametry

movement	zmienna przedstawiajaca kierunek ruchu.

4.10.3.6 updateRange()

```
void Grid::updateRange (
            std::pair< int, int > cords,
            std::pair< int, int > endcords )
```

Wersja funkcji build dla budowania obszarowego. Nieuzywana.

Parametry

cords	pozycja poczatkowa.
endcords	pozycja koncowa.
newTile	obiekt do polozenia.

Dokumentacja dla tej klasy została wygenerowana z plików:

- Symulator miasta/Symulator miasta/Grid.h
- Symulator miasta/Symulator miasta/Grid.cpp

4.11 Dokumentacja klasy HDense

Klasa bloku mieszkalnego. Uzywana jako przykladowy teren mieszkalny.

`#include <HDense.h>`

Diagram dziedziczenia dla HDense

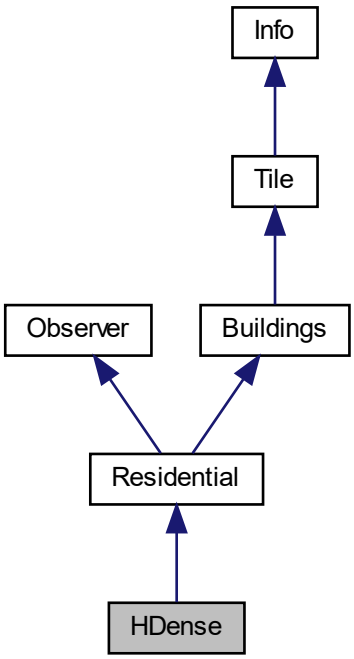
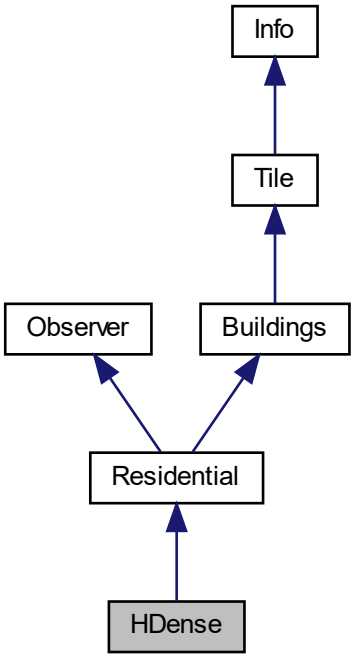


Diagram współpracy dla HDense:



Metody publiczne

- **HDense** (std::map< int, Tile * > adjacent)

Dodatkowe Dziedziczone Składowe

4.11.1 Opis szczegółowy

Klasa bloku mieszkalnego. Uzywana jako przykladowy teren mieszkalny.

Dokumentacja dla tej klasy została wygenerowana z plików:

- Symulator miasta/Symulator miasta/HDense.h
- Symulator miasta/Symulator miasta/HDense.cpp

4.12 Dokumentacja klasy Industrial

Klasa terenow przemyslowych. uzywana jako przykladowe miejce pracy.

`#include <Industrial.h>`

Diagram dziedziczenia dla Industrial

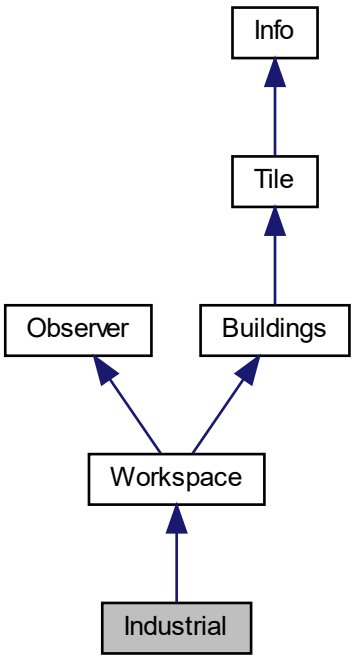
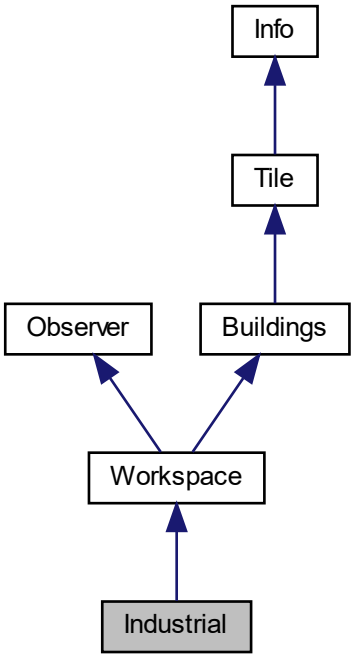


Diagram współpracy dla Industrial:



Metody publiczne

- **Industrial** (std::map< int, Tile * > adjacent)

Dodatkowe Dziedziczone Składowe

4.12.1 Opis szczegółowy

Klasa terenow przemyslowych. uzywana jako przykladowe miejce pracy.

Dokumentacja dla tej klasy została wygenerowana z plików:

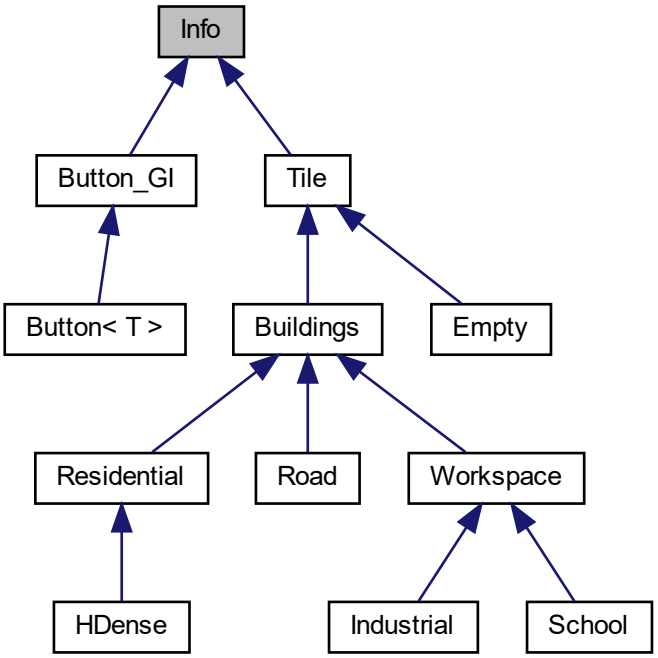
- Symulator miasta/Symulator miasta/Industrial.h
- Symulator miasta/Symulator miasta/Industrial.cpp

4.13 Dokumentacja klasy Info

Klasa Opakowujaca wykorzystywana do zbierania informacji o obiektach.

`#include <Info.h>`

Diagram dziedziczenia dla Info



Metody publiczne

- ~**Info** ()
  Destruktor.
- std::map< std::string, std::string > & getInfo ()
  Accesor mapy informacji.

Atrybuty chronione

- std::map< std::string, std::string > **info**
  Mapa zawierajaca informacje.

4.13.1 Opis szczegółowy

Klasa Opakowujaca wykorzystywana do zbierania informacji o obiektach.

4.13.2 Dokumentacja funkcji składowych

4.13.2.1 getInfo()

```
std::map< std::string, std::string > & Info::getInfo ( )
```

Accesor mapy informacji.

Zwraca

std::map<std::string, std::string>&

Dokumentacja dla tej klasy została wygenerowana z plików:

- Symulator miasta/Symulator miasta/Info.h
- Symulator miasta/Symulator miasta/Info.cpp

4.14 Dokumentacja klasy Informer

Odpowiada za wczytanie informacji z odpowiedniego elementu.

```
#include <Informer.h>
```

Metody publiczne

- Informer (Grid *grid, UI *ui)
  Konstruktor.
- void showInfo (TexManager *texManager)
  wyswietla informacje.

4.14.1 Opis szczegółowy

Odpowiada za wczytanie informacji z odpowiedniego elementu.

4.14.2 Dokumentacja konstruktora i destruktora

4.14.2.1 Informer()

```
Informer::Informer (
            Grid * grid,
            UI * ui )
```

Konstruktor.

Parametry

grid	Wskaznik klasy siatki
ui	Wskaznik klasy interfejsu uzytkownika

4.14.3 Dokumentacja funkcji składowych

4.14.3.1 showInfo()

```
void Informer::showInfo (
            TexManager * texManager )
```

wyswietla informacje.

Parametry

texManager	Wskaznik klasy managera tekstur

Dokumentacja dla tej klasy została wygenerowana z plików:

- Symulator miasta/Symulator miasta/Informer.h
- Symulator miasta/Symulator miasta/Informer.cpp

4.15 Dokumentacja klasy Log

Obiekt Singleton notujacy wydarzenia. zanotowane wydarzenia wpisujace do pliku log.txt.

`#include <Log.h>`

Metody publiczne

- **Log** (Log &other)=delete
- void **operator=** (const Log &)=delete
- void note (std::string message)

  Funkcja notujaca.

Statyczne metody publiczne

- static Log & GetInstance ()

  Statyczna funkcja zwracajaca instancje logu.

4.15.1 Opis szczegółowy

Obiekt Singleton notujacy wydarzenia. zanotowane wydarzenia wpisujace do pliku log.txt.

4.15.2 Dokumentacja funkcji składowych

4.15.2.1 GetInstance()

```
Log & Log::GetInstance ( )  [static]
```

Statyczna funkcja zwracajaca instancje logu.

Zwraca

Clock&

4.15.2.2 note()

```
void Log::note (
            std::string message )
```

Funkcja notujaca.

Parametry

message	wiadomosc do zanotowania

Dokumentacja dla tej klasy została wygenerowana z plików:

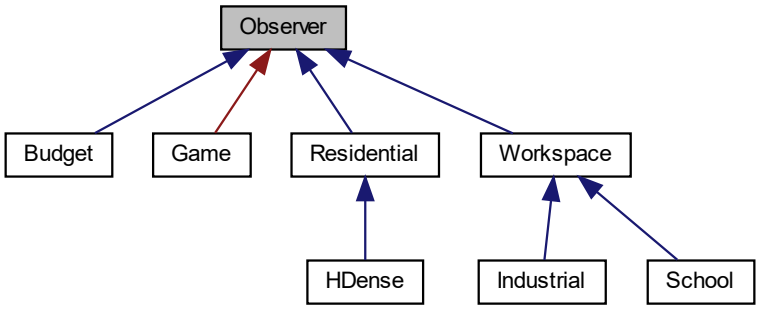
- Symulator miasta/Symulator miasta/Log.h
- Symulator miasta/Symulator miasta/Log.cpp

4.16 Dokumentacja klasy Observer

Interfejs obserwatora zegara.

```
#include <Observer.h>
```

Diagram dziedziczenia dla Observer



Metody publiczne

- virtual void tickUpdate ()=0

  Czysto wirtualna funkcja wywolywana co takt zegara.
- virtual void mTickUpdate ()=0

  Czysto wirtualna funkcja wywolywana co znaczacy takt zegara.

4.16.1 Opis szczegółowy

Interfejs obserwatora zegara.

4.16.2 Dokumentacja funkcji składowych

4.16.2.1 mTickUpdate()

```
virtual void Observer::mTickUpdate ( )  [pure virtual]
```

Czysto wirtualna funkcja wywolywana co znaczacy takt zegara.

Implementowany w Budget, Game, Residential, School i Workspace.

4.16.2.2 tickUpdate()

```
virtual void Observer::tickUpdate ( )  [pure virtual]
```

Czysto wirtualna funkcja wywolywana co takt zegara.

Implementowany w Budget, Game, Residential i Workspace.

Dokumentacja dla tej klasy została wygenerowana z plików:

- Symulator miasta/Symulator miasta/Observer.h
- Symulator miasta/Symulator miasta/Observer.cpp

4.17 Dokumentacja klasy Player

Klasa zajmujaca sie interpretacja dzialan gracza.

```
#include <Player.h>
```

Metody publiczne

- **Player** (Grid *grid, UI *ui)
- void mouse (SDL_Event event)

  Wywolywana gdy gracz wykona dzialanie myszka.
- void key (SDL_Event event)

  Wywolywana gdy gracz nacisnie klawisz.

4.17.1 Opis szczegółowy

Klasa zajmujaca sie interpretacja dzialan gracza.

4.17.2 Dokumentacja funkcji składowych

4.17.2.1 key()

```
void Player::key (
            SDL_Event event )
```

Wywolywana gdy gracz nacisnie klawisz.

Parametry

event	obiekt wydarzenia

4.17.2.2 mouse()

```
void Player::mouse (
            SDL_Event event )
```

Wywolywana gdy gracz wykona dzialanie myszka.

Parametry

event	obiekt wydarzenia

Dokumentacja dla tej klasy została wygenerowana z plików:

- Symulator miasta/Symulator miasta/Player.h
- Symulator miasta/Symulator miasta/Player.cpp

4.18 Dokumentacja klasy Residential

Klasa posrednia reprezentujaca obszar mieszkalny.

`#include <Residential.h>`

Diagram dziedziczenia dla Residential

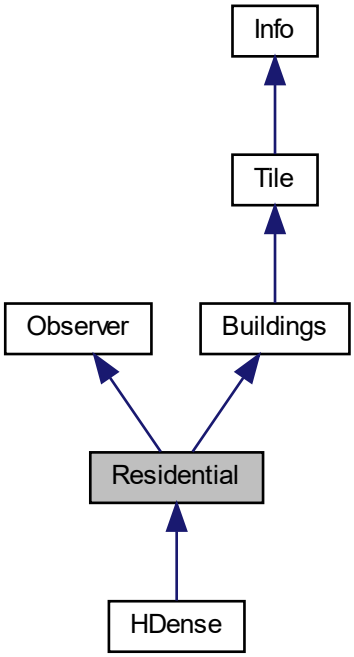
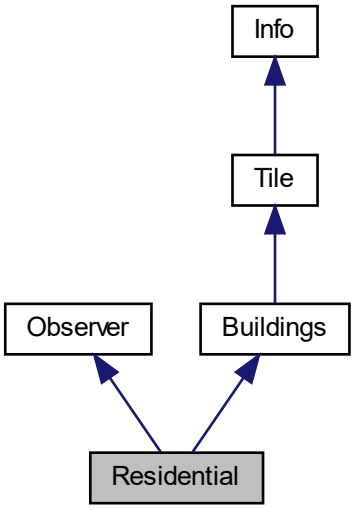


Diagram współpracy dla Residential:



Metody publiczne

- **Residential** (unsigned short int popcap, unsigned short int baseValue, unsigned short int maintenance, unsigned short int cost, int texture, std::map< int, Tile * > adjacent)
- virtual void mTickUpdate ()

  Czysto wirtualna funkcja wywolywana co znaczacy takt zegara.
- virtual void tickUpdate ()

  Czysto wirtualna funkcja wywolywana co takt zegara.
- int getPop ()

  Zwraca aktualna ilosc mieszkancow.
- void propagate (int side, int range, std::set< Tile * > &q, std::list< Tile * > &s)

  funkcja propagacji, uzywana przez obiekty komunalne do spisania obserwatorow

Dodatkowe Dziedziczone Składowe

4.18.1 Opis szczegółowy

Klasa posrednia reprezentujaca obszar mieszkalny.

4.18.2 Dokumentacja funkcji składowych

4.18.2.1 getPop()

```
int Residential::getPop ( )
```

Zwraca aktualna ilosc mieszkancow.

Zwraca

int

4.18.2.2 mTickUpdate()

```
void Residential::mTickUpdate ( )  [virtual]
```

Czysto wirtualna funkcja wywolywana co znaczacy takt zegara.

Implementuje Observer.

4.18.2.3 propagate()

```
void Residential::propagate (
            int side,
            int range,
            std::set< Tile * > & q,
            std::list< Tile * > & s )  [virtual]
```

funkcja propagacji, uzywana przez obiekty komunalne do spisania obserwatorow

Parametry

side	strona
range	pozostaly zasieg
q	zbior obiektow odwiedzonych
s	lista obiektow zainteresowania

Reimplementowana z Tile.

4.18.2.4 tickUpdate()

```
void Residential::tickUpdate ( )  [virtual]
```

Czysto wirtualna funkcja wywolywana co takt zegara.

Implementuje Observer.

Dokumentacja dla tej klasy została wygenerowana z plików:

- Symulator miasta/Symulator miasta/Residential.h
- Symulator miasta/Symulator miasta/Residential.cpp

4.19 Dokumentacja klasy Road

Klasa drogi. zmienia swoja teksture w zaleznosci od sasiednich drog wszystkie methody sa przeslaniajace.

`#include <Road.h>`

Diagram dziedziczenia dla Road

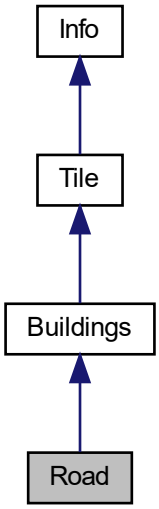
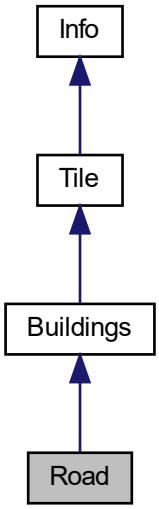


Diagram współpracy dla Road:



Metody publiczne

- **Road** (std::map< int, Tile * > adjacent)
- void changeAdjecent (Tile *newAdj, int side)

  Dodaje sasiada.
- void removeAdjecent (int side)

  usuwa sasiada po konkretnej stronie.
- void update ()

  funkcja aktualizacji. dzialanie znaczaco rozni sie w zaleznosci od klasy potomnej
- void propagate (int side, int range, std::set< Tile * > &q, std::list< Tile * > &s)

  funkcja propagacji, uzywana przez obiekty komunalne do spisania obserwatorow

Dodatkowe Dziedziczone Składowe

4.19.1 Opis szczegółowy

Klasa drogi. zmienia swoja teksture w zaleznosci od sasiednich drog wszystkie methody sa przeslaniajace.

4.19.2 Dokumentacja funkcji składowych

4.19.2.1 changeAdjecent()

```
void Road::changeAdjecent (
            Tile * newAdj,
            int side )  [virtual]
```

Dodaje sasiada.

Parametry

newAdj	wskaznik sasiad
side	strona

Reimplementowana z Tile.

4.19.2.2 propagate()

```
void Road::propagate (
            int side,
            int range,
            std::set< Tile * > & q,
            std::list< Tile * > & s )  [virtual]
```

funkcja propagacji, uzywana przez obiekty komunalne do spisania obserwatorow

Parametry

side	strona
range	pozostaly zasieg
q	zbior obiektow odwiedzonych
s	lista obiektow zainteresowania

Reimplementowana z Tile.

4.19.2.3 removeAdjecent()

```
void Road::removeAdjecent (
            int side )  [virtual]
```

usuwa sasiada po konkretnej stronie.

Parametry

side	strona

Reimplementowana z Tile.

4.19.2.4 update()

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

funkcja aktualizacji. dzialanie znaczaco rozni sie w zaleznosci od klasy potomnej

Reimplementowana z Tile.

Dokumentacja dla tej klasy została wygenerowana z plików:

- Symulator miasta/Symulator miasta/Road.h
- Symulator miasta/Symulator miasta/Road.cpp

4.20 Dokumentacja klasy School

Klasa szkoly. uzywana jako przykladowe budynek komunalny.

`#include <School.h>`

Diagram dziedziczenia dla School

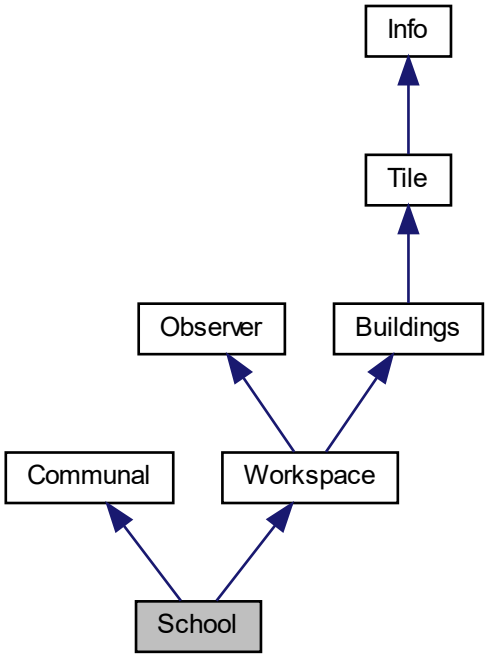
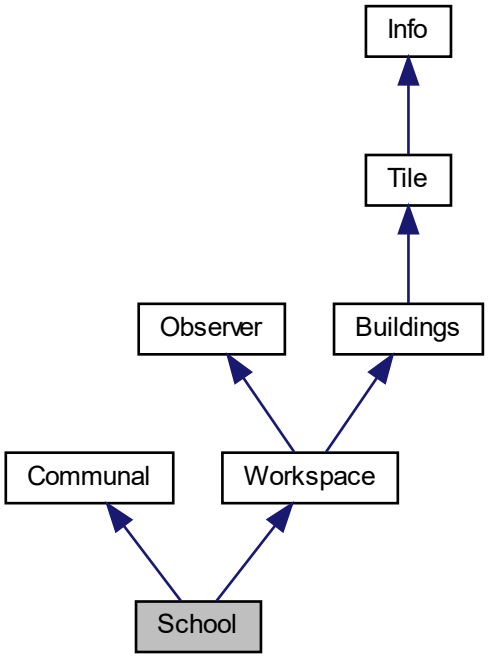


Diagram współpracy dla School:



Metody publiczne

- **School** (std::map< int, Tile * > adjacent)
- std::list< Tile * > **relist** ()

  Funkcja zwraca liste obiektow zainteresowanych obserwowaniem tego budynku uruchamia metode propagate w sasiednich polach return std::list<Tile>*
- void mTickUpdate ()

  Czysto wirtualna funkcja wywolywana co znaczacy takt zegara.

Dodatkowe Dziedziczone Składowe

4.20.1 Opis szczegółowy

Klasa szkoly. uzywana jako przykladowe budynek komunalny.

4.20.2 Dokumentacja funkcji składowych

4.20.2.1 mTickUpdate()

```
void School::mTickUpdate ( )  [virtual]
```

Czysto wirtualna funkcja wywolywana co znaczacy takt zegara.

Reimplementowana z Workspace.

Dokumentacja dla tej klasy została wygenerowana z plików:

- Symulator miasta/Symulator miasta/School.h
- Symulator miasta/Symulator miasta/School.cpp

4.21 Dokumentacja klasy TexManager

Klasa managera tekstur. Odpowiedzialna za przetwarzanie oraz wyswietlanie textur i napisow.

```
#include <TexManager.h>
```

Metody publiczne

- **TexManager** (SDL_Renderer *renderer, int tileSize, std::vector< Button_GI * > buttons)
- void renderTile (int textureInt, int x, int y)

  Wyswietla teksture kratki.
- void renderImage (int textureInt, int x, int y, int w, int h)

  Wyswietla niestandardowa teksture.
- void renderText (int x, int y, int w, int h, const char *text, SDL_Color color)

  Wyswietla tekst.
- void **renderUI** ()

  Wyswietla interfejs uzytkownika.
- void renderInfo (std::map< std::string, std::string > &info)

  Wyswietla tresc informatora.

4.21.1 Opis szczegółowy

Klasa managera tekstur. Odpowiedzialna za przetwarzanie oraz wyswietlanie textur i napisow.

4.21.2 Dokumentacja funkcji składowych

4.21.2.1 renderImage()

```
void TexManager::renderImage (
            int textureInt,
            int x,
            int y,
            int w,
            int h )
```

Wyswietla niestandardowa teksture.

Parametry

textureInt	identyfikator tekstury
x	pozycja (szerokosc)
y	pozycja (wysokosc)
w	szerokosc
h	wysokosc

4.21.2.2 renderInfo()

```
void TexManager::renderInfo (
            std::map< std::string, std::string > & info )
```

Wyswietla tresc informatora.

Parametry

info	mapa informacji

4.21.2.3 renderText()

```
void TexManager::renderText (
            int x,
            int y,
            int w,
            int h,
            const char * text,
            SDL_Color color )
```

Wyswietla tekst.

Parametry

textureInt	identyfikator tekstury
x	pozycja (szerokosc)
y	pozycja (wysokosc)
w	szerokosc
h	wysokosc
text	tresc
color	kolor

4.21.2.4 renderTile()

```
void TexManager::renderTile (
            int textureInt,
            int x,
            int y )
```

Wyswietla teksture kratki.

Parametry

textureInt	identyfikator tekstury
x	pozycja na widocznej siatce (szerokosc)
y	pozycja na widocznej siatce (wysokosc)

Dokumentacja dla tej klasy została wygenerowana z plików:

- Symulator miasta/Symulator miasta/TexManager.h
- Symulator miasta/Symulator miasta/TexManager.cpp

4.22 Dokumentacja klasy Tile

Wirtualna Klasa reprezentujaca kratke na siatce.

`#include <Tile.h>`

Diagram dziedziczenia dla Tile

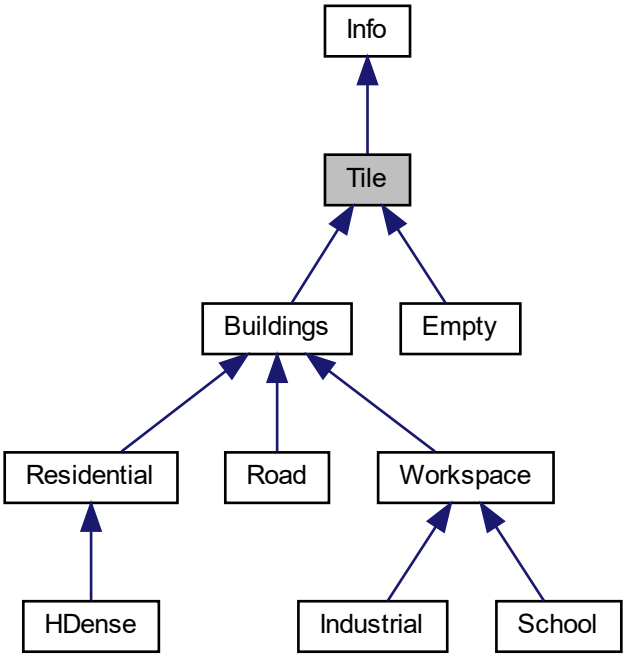


Diagram współpracy dla Tile:

Metody publiczne

- Tile (int texture, std::map< int, Tile * > adjacent)
  Konstruktor.
- virtual void changeTex (int tex)
  setter dla texture.
- virtual int getTex ()
  getter dla texture.
- virtual void changeAdjecent (Tile *newAdj, int side)
  Dodaje sasiada.
- virtual void removeAdjecent (int side)
  usuwa sasiada po konkretnej stronie.
- virtual void update ()
  funkcja aktualizacji. dzialanie znaczaco rozni sie w zaleznosci od klasy potomnej
- virtual void propagate (int side, int range, std::set< Tile * > &q, std::list< Tile * > &s)
  funkcja propagacji, uzywana przez obiekty komunalne do spisania obserwatorow

Atrybuty chronione

- unsigned short int texture
- std::map< int, Tile * > adjacent

4.22.1 Opis szczegółowy

Wirtualna Klasa reprezentujaca kratke na siatce.

4.22.2 Dokumentacja konstruktora i destruktora

4.22.2.1 Tile()

```
Tile::Tile (
            int texture,
            std::map< int, Tile * > adjacent )
```

Konstruktor.

Parametry

texture	identyfikator tekstury
adjacent	mapa sasiednich kratek

4.22.3 Dokumentacja funkcji składowych

4.22.3.1 changeAdjecent()

```
void Tile::changeAdjecent (
            Tile * newAdj,
            int side )  [virtual]
```

Dodaje sasiada.

Parametry

newAdj	wskaznik sasiad
side	strona

Reimplementowana w Empty i Road.

4.22.3.2 changeTex()

```
void Tile::changeTex (
            int tex )  [virtual]
```

setter dla texture.

Parametry

tex	identyfikator nowej tekstury

Reimplementowana w Empty.

4.22.3.3 getTex()

```
int Tile::getTex ( )  [virtual]
```

getter dla texture.

Zwraca

int

Reimplementowana w Empty.

4.22.3.4 propagate()

```
void Tile::propagate (
            int side,
            int range,
            std::set< Tile * > & q,
            std::list< Tile * > & s )  [virtual]
```

funkcja propagacji, uzywana przez obiekty komunalne do spisania obserwatorow

Parametry

side	strona
range	pozostaly zasieg
q	zbior obiektow odwiedzonych
s	lista obiektow zainteresowania

Reimplementowana w Residential i Road.

4.22.3.5 removeAdjecent()

```
void Tile::removeAdjecent (
            int side )  [virtual]
```

usuwa sasiada po konkretnej stronie.

Parametry

side	strona

Reimplementowana w Empty i Road.

4.22.3.6 update()

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

funkcja aktualizacji. dzialanie znaczaco rozni sie w zaleznosci od klasy potomnej

Reimplementowana w Empty i Road.

4.22.4 Dokumentacja atrybutów składowych

4.22.4.1 adjacent

```
std::map<int,Tile*> Tile::adjacent  [protected]
```

mapa sasiednich kratek

4.22.4.2 texture

```
unsigned short int Tile::texture  [protected]
```

identyfikator tekstury

Dokumentacja dla tej klasy została wygenerowana z plików:

- Symulator miasta/Symulator miasta/Tile.h
- Symulator miasta/Symulator miasta/Tile.cpp

4.23 Dokumentacja klasy UI

Klasa zajmujaca sie dzialaniem interfejsu uzytkownika.

```
#include <UI.h>
```

Metody publiczne

- std::vector< Button_GI * > getButtons ()
  Getter dla buttons.
- Button_GI * point (std::pair< int, int > mpos)
  Sprawdza czy mysz wskazuje na przycisk, jesli tak zwraca go.
- void click (std::pair< int, int > mpos, Grid *grid)
  Sprawdza czy nacisnieto przycisk, jesli tak aktywuje go.
- void onBoard (SDL_Event event, std::pair< int, int > mpos)
  Przesyla pozycje myszy do odpowiedniej metody przycisku.

4.23.1 Opis szczegółowy

Klasa zajmujaca sie dzialaniem interfejsu uzytkownika.

4.23.2 Dokumentacja funkcji składowych

4.23.2.1 click()

```
void UI::click (
            std::pair< int, int > mpos,
            Grid * grid )
```

Sprawdza czy nacisnieto przycisk, jesli tak aktywuje go.

Parametry

mpos	pozycja myszki
grid	wskaznik siatki

4.23.2.2 getButtons()

```
std::vector< Button_GI * > UI::getButtons ( )
```

Getter dla buttons.

Zwraca

std::vector<Button_GI*>

4.23.2.3 onBoard()

```
void UI::onBoard (
            SDL_Event event,
            std::pair< int, int > mpos )
```

Przesyla pozycje myszy do odpowiedniej metody przycisku.

Parametry

event	wydarzenie
mpos	pozycja myszki

4.23.2.4 point()

```
Button_GI * UI::point (
            std::pair< int, int > mpos )
```

Sprawdza czy mysz wskazuje na przycisk, jesli tak zwraca go.

Parametry

mpos	pozycja myszki

Zwraca

Button_GI*

Dokumentacja dla tej klasy została wygenerowana z plików:

- Symulator miasta/Symulator miasta/UI.h
- Symulator miasta/Symulator miasta/UI.cpp

4.24 Dokumentacja klasy Workspace

Klasa posrednia reprezentujaca miejsce pracy.

`#include <Workspace.h>`

Diagram dziedziczenia dla Workspace

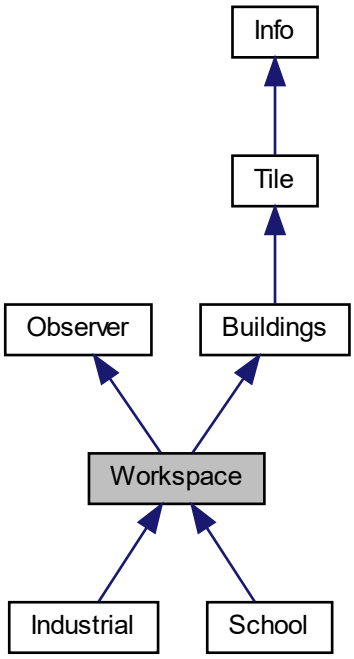
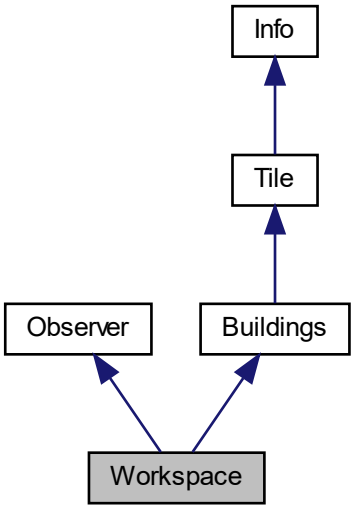


Diagram współpracy dla Workspace:



Metody publiczne

- **Workspace** (unsigned short int jobs, unsigned short int baseValue, unsigned short int maintenance, unsigned short int cost, int tex, std::map< int, Tile * > adjacent)
- virtual void mTickUpdate ()

  Czysto wirtualna funkcja wywolywana co znaczacy takt zegara.
- virtual void tickUpdate ()

  Czysto wirtualna funkcja wywolywana co takt zegara.
- virtual void setWorkforce (int &pop)

  wylicza ilosc osob pracujacych i odejmuje ja przez referencje od bezrobotnych

Dodatkowe Dziedziczone Składowe

4.24.1 Opis szczegółowy

Klasa posrednia reprezentujaca miejsce pracy.

4.24.2 Dokumentacja funkcji składowych

4.24.2.1 mTickUpdate()

```
void Workspace::mTickUpdate ( )  [virtual]
```

Czysto wirtualna funkcja wywolywana co znaczacy takt zegara.

Implementuje Observer.

Reimplementowana w School.

4.24.2.2 setWorkforce()

```
void Workspace::setWorkforce (
            int & pop )  [virtual]
```

wylicza ilosc osob pracujacych i odejmuje ja przez referencje od bezrobotnych

Parametry

pop	liczba osob bezrobotnych

4.24.2.3 tickUpdate()

```
void Workspace::tickUpdate ( )  [virtual]
```

Czysto wirtualna funkcja wywolywana co takt zegara.

Implementuje Observer.

Dokumentacja dla tej klasy została wygenerowana z plików:

- Symulator miasta/Symulator miasta/Workspace.h
- Symulator miasta/Symulator miasta/Workspace.cpp

Rozdział 5

Dokumentacja plików

5.1 Budget.h

```
1 #pragma once
2 #include "Clock.h"
3 #include "Buildings.h"
4 #include "Residential.h"
5 #include "Workspace.h"
6 #include <unordered_map>
7
11 class Budget : public Observer
12 {
13 private:
17     long long treasury;
22     int balance, pop, free, bid, rid, wid;
26     std::unordered_map<int, Buildings*> buildings;
30     std::unordered_map<int, Residential*> res;
34     std::unordered_map<int, Workspace*> workspace;
35
36     Budget();
40     void recount();
44     void recountPop();
45
46 public:
47
48     Budget(Budget& other) = delete;
49
50     void operator=(const Budget&) = delete;
55     static Budget& GetInstance();
59     void tickUpdate();
64     void mTickUpdate();
69     int addBuilding(Buildings* b);
74     void removeBuilding(int id);
79     int addPop(Residential* b);
84     void removePop(int id);
89     int addWork(Workspace* b);
94     void removeWork(int id);
99     int getTreasury();
104     int getBalance();
109     int getPop();
110
111 };
```

5.2 Builder.h

```
1 #pragma once
2 #include "Grid.h"
3 #include "Log.h"
4 #include "Buildings.h"
5 #include "Road.h"
6 #include "HDense.h"
7 #include "Industrial.h"
8 #include "Empty.h"
9 #include "School.h"
16 template <class T>
17 class Builder
18 {
```

```
22      Grid* grid;
26      int mode;
30      bool chk;
34      std::pair<int, int> iniCords;
35  public:
36
42      Builder(Grid* grid, int mode) :grid(grid), mode(mode)
43      {
44          chk = 0;
45          Log::GetInstance().note("Builder<T> invoked");
46          if (mode == 0)
47              Log::GetInstance().note("singular mode");
48          else
49              Log::GetInstance().note("area mode");
50
51      }
52
53
54      ~Builder()
55      {
56          Log::GetInstance().note("Builder revoked");
57      }
58
65      bool mouseDown(std::pair<int, int> cords)
66      {
67          cords = { cords.first / 50, cords.second / 50 };
68          Log::GetInstance().note("mouse down " + std::to_string(cords.first) + "x" +
        std::to_string(cords.second));
69          if (cords.first > 31 || cords.second > 13)
70          {
71              Log::GetInstance().note("Builder failed, out of bounds");
72              return 0;
73          }
74
75          if (mode == 0)
76          {
77              /*switch (type)
78              {
79              default:
80                  break;
81              }*/
82              return 1;
83          }
84          else
85          {
86              chk = 1;
87              iniCords = cords;
88              return 0;
89          }
90      }
91
99      bool mouseUp(std::pair<int, int> cords)
100     {
101         if (chk == 0)
102             return 0;
103         cords = { cords.first / 50, cords.second / 50 };
104         Log::GetInstance().note("mouse up " + std::to_string(cords.first) + "x" +
        std::to_string(cords.second));
105         if (cords.first > 31 || cords.second > 13)
106         {
107             Log::GetInstance().note("Builder failed, out of bounds");
108             return 0;
109         }/*
110         switch (type)
111         {
112         case 10:
113             buildRoad(cords);
114             break;
115         case 20:
116             buildHD(cords);
117             break;
118         case 30:
119             buildInd(cords);
120             break;
121         default:
122             break;
123         }*/
124         build(cords);
125         return 1;
126
127     }
128
135     std::map<int, Tile*> Adjecent(int i, int j)
136     {
137         std::map<int, Tile*> adjRoads;
138         adjRoads[0] = grid->getVisibleTile({ i - 1,j });
139         adjRoads[1] = grid->getVisibleTile({ i ,j + 1 });
```

```
140         adjRoads[2] = grid->getVisibleTile({ i + 1,j });
141         adjRoads[3] = grid->getVisibleTile({ i,j - 1 });
142         return adjRoads;
143     }
144
149     void build(std::pair<int, int> cords)
150     {
151         if (iniCords.first > cords.first)
152         {
153             std::swap(iniCords.first, cords.first);
154         }
155         if (iniCords.second > cords.second)
156         {
157             std::swap(iniCords.second, cords.second);
158         }
159         for (int i = iniCords.first;i <= cords.first;++i)
160         {
161             for (int j = iniCords.second;j <= cords.second;++j)
162             {
163                 if (grid->getVisibleTile({i,j})->getTex() != 0)
164                     grid->build({ i,j }, new T(Adjecent(i, j)));
165             }
166         }
167
168     }
169
170 };
171
```

5.3 Buildings.h

```
1 #pragma once
2 #include "Clock.h"
3 #include "Tile.h"
4 #include "Communal.h"
8 class Buildings : public Tile
9 {
13     unsigned short int cost;
17     unsigned short int id;
18 protected:
22     std::set<Communal*> observing;
26     std::set<Buildings*> observers;
30     int maintenance;
34     int revenue;
35 public:
36     Buildings(int maintenance, int cost, int tex, std::map<int, Tile*> adjacent);
37     ~Buildings();
42     int getBalance();
47     int getCost();
52     void addObserving(Communal* b);
53     //void addObserver(Buildings* c);
58     void removeObserving(Communal* b);
59     //void removeObserver(Buildings* c);
60
61 };
```

5.4 Button.h

```
1 #pragma once
2 #include <utility>
3 #include "Builder.h"
4 #include "Button_GI.h"
10 template <class T>
11 class Button: public Button_GI
12 {
16     Builder<T>* builder;
17 public:
18
19     Button(int texId, std::pair<int, int> position) : Button_GI(texId, position)
20     {
21         active = 0;
22         info["Object"] = "Button";
23
24     }
25
26
27     void activate(Grid* grid)
28     {
29         if (active == 1)
```

```
30          {
31              delete builder;
32          }
33          active = 1;
34          Log::GetInstance().note("przycisk");
35          builder = new Builder<T>(grid, 1);
36      }
37
38      void deactivate()
39      {
40          delete builder;
41          active = 0;
42      }
43
44      bool mouseDown(std::pair<int, int> cords)
45      {
46          return builder->mouseDown(cords);
47      }
48
49      bool mouseUp(std::pair<int, int> cords)
50      {
51          return builder->mouseUp(cords);
52      }
53 };
```

5.5 Button_GI.h

```
1 #pragma once
2 #include "Info.h"
3 #include "Grid.h"
4 #include <utility>
8 class Button_GI : public Info
9 {
13      int texId;
14 protected:
18      bool active;
22      std::pair<int, int> position;
23 public:
24      Button_GI(int texId, std::pair<int, int> position);
29      std::pair<int, int> getPosition();
34      int getTex();
39      virtual void activate(Grid* grid)=0;
43      virtual void deactivate()=0;
49      virtual bool mouseDown(std::pair<int, int> cords)=0;
55      virtual bool mouseUp(std::pair<int, int> cords)=0;
56 };
```

5.6 Clock.h

```
1 #pragma once
2 #include <set>
3 #include <chrono>
4 #include <thread>
5 #include "Observer.h"
15 class Clock
16 {
20      std::set<Observer*> observers, majorObservers;
24      bool running = 1;
28      Clock();
29
30 public:
34      Clock(Clock& other) = delete;
35
36      void operator=(const Clock&) = delete;
37      ~Clock();
42      static Clock& GetInstance();
46      void tick();
51      void newOb(Observer* ob);
56      void newMajorOb(Observer* ob);
61      void deleteOb(Observer* ob);
66      void deleteMajorOb(Observer* ob);
71      void tickInform();
75      void majorTickInform();
79      void start();
83      void stop();
84
85 private:
86
87 };
```

5.7 Communal.h

```
1 #pragma once
2 #include "Tile.h"
6 class Communal
7 {
11     float influance;
15     unsigned short int range;
16
17 public:
18     Communal(float influance, unsigned short int range);
19     ~Communal();
20
25     float getInfluance();
30     unsigned short int  getRange();
31 };
```

5.8 Empty.h

```
1 #pragma once
2 #include "Tile.h"
8 class Empty : public Tile
9 {
10 protected:
11 public:
12     Empty(int texture, std::map<int, Tile*> adjacent);
13     Empty(std::map<int, Tile*> adjacent);
14     ~Empty();
15     void changeTex(int tex);
16     int getTex();
17     void changeAdjecent(Tile* newAdj, int side);
18     void removeAdjecent(int side);
19     void update();
20 };
```

5.9 Game.h

```
1 #pragma once
2 #include "SDL.h"
3 #include "SDL_image.h"
4 #include "Grid.h"
5 #include "TexManager.h"
6 #include "Clock.h"
7 #include "Player.h"
8 #include "Budget.h"
9 #include "UI.h"
10 #include "Informer.h"
11 #include <iostream>
12 #include <fstream>
13 #include <vector>
14 #include <string>
15
19 class Game : private Observer
20 {
21 private:
25     bool isActive;
29     SDL_Window* mainWin;
33     SDL_Renderer* renderer;
37     Grid* grid;
41     TexManager* texManager;
45     Player* player;
49     Informer* informer;
53     std::vector<int> importSettings();
54
55 public:
56
60     Game();
64     ~Game();
68     void events();
72     void render();
76     void clean();
80     SDL_Renderer* getRenderer();
85     void renderVisibleArea();
89     void mTickUpdate();
93     void tickUpdate();
98     bool active();
99
100
101
102 };
```

5.10 Grid.h

```
1 #pragma once
2 #include <string>
3 #include "Empty.h"
9 class Grid
10 {
14     Tile*** grid;
18     int height, width, vHeight, vWidth;
22     std::pair<int, int> centered;
29     void replace(std::pair<int, int> cords, Tile* newTile);
30 public:
38     Grid(int width, int height, int vWidth, int vHeight );
43     ~Grid();
44
50     void getCurrentRange(std::pair<int, int>& horizontal, std::pair<int, int>& vertical);
56     Tile* getTile(std::pair<int, int> cords);
61     void moveOnMap(std::pair<int, int> movement);
68     void build(std::pair<int, int> cords, Tile* newTile);
76     void updateRange(std::pair<int, int> cords, std::pair<int, int> endcords);
82     Tile* getVisibleTile(std::pair<int, int> cords);
83 };
84
```

5.11 HDense.h

```
1 #pragma once
2 #include "Clock.h"
3 #include "Tile.h"
4 #include "Residential.h"
9 class HDense: public Residential
10 {
11 public:
12     HDense(std::map<int, Tile*> adjacent);
13     ~HDense();
14 };
```

5.12 Industrial.h

```
1 #pragma once
2 #include "Clock.h"
3 #include "Tile.h"
4 #include "Workspace.h"
9 class Industrial : public Workspace
10 {
11 public:
12     Industrial(std::map<int, Tile*> adjacent);
13     ~Industrial();
14 };
```

5.13 Info.h

```
1 #pragma once
2 #include <string>
3 #include <map>
7 class Info
8 {
9 protected:
13     std::map<std::string, std::string> info;
14 public:
18      ~Info();
23     std::map<std::string, std::string>& getInfo();
24 };
```

5.14 Informer.h

```
1 #pragma once
2 #include "Info.h"
3 #include "UI.h"
4 #include "Grid.h"
```

```
5 #include "TexManager.h"
9 class Informer
10 {
14     Grid* grid;
18     UI* ui;
22     std::pair<int, int> pos;
23 public:
29     Informer(Grid* grid, UI* ui);
34     void showInfo(TexManager* texManager);
35 };
```

5.15 Log.h

```
1 #pragma once
2
3 #include <fstream>
4 #include <string>
5 #include <iostream>
10 class Log
11 {
12 private:
13     std::ofstream logFile;
14     Log();
15
16 public:
17
18     Log(Log& other) = delete;
19
20     void operator=(const Log&) = delete;
25     static Log& GetInstance();
30     void note(std::string message);
31
32 };
```

5.16 Observer.h

```
1 #pragma once
5 class Observer
6 {
7 public:
8
9     Observer();
10     ~Observer();
14     virtual void tickUpdate() = 0;
18     virtual void mTickUpdate() = 0;
19
20 private:
21
22 };
```

5.17 Player.h

```
1 #pragma once
2 #include "Clock.h"
3 #include "SDL.h"
4 #include "Grid.h"
5 #include "UI.h"
9 class Player
10 {
14     Grid* grid;
18     UI* ui;
19
20
21     public:
22     Player(Grid* grid, UI* ui);
23     ~Player();
28     void mouse(SDL_Event event);
33     void key(SDL_Event event);
34
35 };
```

5.18 Residential.h

```
1 #pragma once
2 #include "Clock.h"
3 #include "Buildings.h"
7 class Residential : public Observer, public Buildings
8 {
12     unsigned short int pop, popcap;
16     double baseValue;
20     float qol;
24     unsigned short int popid;
25 public:
26     Residential(unsigned short int popcap, unsigned short int baseValue, unsigned short int maintenance,
       unsigned short int cost, int texture, std::map<int, Tile*> adjacent);
27     ~Residential();
28
29     virtual void mTickUpdate();
30     virtual void tickUpdate();
35     int getPop();
36     void propagate(int side, int range, std::set<Tile*>& q, std::list<Tile*>& s);
37 };
```

5.19 Road.h

```
1 #pragma once
2 #include "Tile.h"
3 #include "Buildings.h"
9 class Road : public Buildings
10 {
11 public:
12     Road(std::map<int, Tile*> adjacent);
13     ~Road();
14
15     void changeAdjecent(Tile* newAdj, int side);
16     void removeAdjecent(int side);
17     void update();
18     void propagate(int side, int range, std::set<Tile*>& q, std::list<Tile*>& s);
19 };
```

5.20 School.h

```
1 #pragma once
2 #include "Clock.h"
3 #include "Workspace.h"
4 #include "Communal.h"
9 class School : public Communal, public Workspace
10 {
11 public:
12     School(std::map<int, Tile*> adjacent);
13     ~School();
19     std::list<Tile*> relist();
20     void mTickUpdate();
21 };
```

5.21 TexManager.h

```
1 #pragma once
2 #include "SDL.h"
3 #include "SDL_image.h"
4 #include "SDL_ttf.h"
5 #include "Button.h"
6 #include <vector>
11 class TexManager
12 {
16     SDL_Texture* textures[100];
20     SDL_Texture* UI[40];
24     SDL_Renderer* renderer;
28     std::vector<Button_GI*> buttons;
32     int tileSize;
36     SDL_Color fColor, bColor;
40     TTF_Font* font;
41 public:
42     TexManager(SDL_Renderer* renderer, int tileSize, std::vector<Button_GI*> buttons);
43     ~TexManager();
50     void renderTile(int textureInt, int x, int y);
```

```
59      void renderImage(int textureInt, int x, int y, int w, int h);
70      void renderText(int x, int y, int w, int h, const char* text, SDL_Color color);
74      void renderUI();
79      void renderInfo(std::map<std::string, std::string>& info);
80
81
82  private:
89      SDL_Texture* assignTexture(const char* path, SDL_Renderer* renderer);
94      void importTextures(SDL_Renderer* renderer);
99      void importUI(SDL_Renderer* renderer);
100
101 };
```

5.22 Tile.h

```
1  #pragma once
2  #include <map>
3  #include <list>
4  #include <set>
5  #include "Info.h"
9  class Tile: public Info
10 {
11 protected:
15      unsigned short int texture;
19      std::map<int,Tile*> adjacent;
20 public:
26      Tile(int texture, std::map<int, Tile*> adjacent);
27      virtual ~Tile();
32      virtual void changeTex(int tex);
37      virtual int getTex();
43      virtual void changeAdjecent(Tile* newAdj, int side);
48      virtual void removeAdjecent(int side);
53      virtual void update();
62      virtual void propagate(int side, int range, std::set<Tile*>& q, std::list<Tile*>& s);
63 };
```

5.23 UI.h

```
1  #pragma once
2  #include "Log.h"
3  #include "Button_GI.h"
4  #include <vector>
5  #include "SDL.h"
9  class UI
10 {
14      std::vector<Button_GI*> buttons;
18      bool mode;
22      Button_GI* used;
23
24 public:
25      UI();
30      std::vector<Button_GI*> getButtons();
36      Button_GI* point(std::pair<int, int> mpos);
42      void click(std::pair<int, int> mpos, Grid* grid);
48      void onBoard(SDL_Event event, std::pair<int, int> mpos);
49
50 };
```

5.24 Workspace.h

```
1  #pragma once
2  #include "Clock.h"
3  #include "Buildings.h"
7  class Workspace : public Observer, public Buildings
8  {
12      int jobs, taken;
16      double value, baseValue;
20      unsigned short int jobid;
21 public:
22      Workspace(unsigned short int jobs, unsigned short int baseValue, unsigned short int maintenance,
        unsigned short int cost, int tex, std::map<int, Tile*> adjacent);
23      ~Workspace();
24
25      virtual void mTickUpdate();
26      virtual void tickUpdate();
31      virtual void setWorkforce(int& pop);
32 };
```

Indeks