Liszaj

Wygenerowano przez Doxygen 1.8.14

Spis treści

Rozdział 1

Indeks przestrzeni nazw

1.1 Lista przestrzeni nazw

Tutaj znajdują się wszystkie przestrzenie nazw wraz z ich krótkimi opisami:

Rozdział 2

Indeks hierarchiczny

2.1 Hierarchia klas

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

Rozdział 3

Indeks klas

3.1 Lista klas

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

Rozdział 4

Indeks plików

4.1 Lista plików

Tutaj znajduje się lista wszystkich plików z ich krótkimi opisami:

Rozdział 5

Dokumentacja przestrzeni nazw

5.1 Dokumentacja przestrzeni nazw ImGui

Funkcje

- bool Combo (const char *label, int *currIndex, std::vector< std::string > &values)

Zmienne

- static auto vector_getter

5.1.1 Opis szczegółowy

Przestrzeń nazw biblioteki ImGui, do której zostały dodane nowe funkcje.

5.1.2 Dokumentacja funkcji

5.1.2.1 Combo()

```
bool ImGui::Combo (
            const char * label,
            int * currIndex,
            std::vector< std::string > & values )
```

Własna funkcja Combo do tworzenia combo boxa.

Parametry

label	Wyświetlana nazwa elementu
currIndex	wskaźnik do aktualnie wybranego elementu.
values	wektor stringów przechowujący dzielniki liczb

Wewnątrz funkcji badane jest czy wektor jest pusty i zwarcane jest false kiedy tak jest. W przeciwnym wypadku wywoływana jest funkcja o prototypie: IMGUI_API bool Combo(const char∗ label, int∗ current_item, bool(∗ items↩ _getter)(void∗ data, int idx, const char∗∗ out_text), void∗ data, int items_count, int popup_max_height_in_items = -1);

```
40     {
41         if (values.empty()) { return false; }
42         return Combo(label, currIndex, vector_getter,
43             static_cast<void*>(&values), values.size());
44     }
```

5.1.3 Dokumentacja zmiennych

5.1.3.1 vector_getter

```
auto ImGui::vector_getter  [static]
```

Wartość początkowa:

```
= [](void* vec, int idx, const char** out_text)
    {
        auto& vector = *static_cast<std::vector<std::string>*>(vec);
        if (idx < 0 || idx >= static_cast<int>(vector.size())) { return false; }
        *out_text = vector.at(idx).c_str();
        return true;
    }
```

Funkcja anonimowa utworzona na wzór funkcji anonimowej z prototypu funkcji. Prototyp tej funkcji: IMGUI_API bool Combo(const char∗ label, int∗ current_item, bool(∗ items_getter)(void∗ data, int idx, const char∗∗ out_text), void∗ data, int items_count, int popup_max_height_in_items = -1);

Parametry

vec	Przekazuje wskaźnik (służy jako wskaźnik do dowolnego typu).
idx	Indeks elementu w wektorze.
out_text	Wskaźnik przez który przekazwyany jest wyjściowy tekst.

Do zmiennej vector przypisywany jest wskaźnik rzutowany do wektora stringów Jeżeli index jest ujemny lub większy niż rozmiar wektora to zwracane jest false. W przeciwnym razie wartość ta przypisywana out_text i zwracane jest true.

5.2 Dokumentacja przestrzeni nazw thor

Komponenty

- class ColorAnimationModified

5.2.1 Opis szczegółowy

Przestrzeń nazw thor. Jest to przestrzeń nazw wykorzystywana przez bibliotekę thor.

Rozdział 6

Dokumentacja klas

6.1 Dokumentacja klasy thor::ColorAnimationModified

`#include <ColorAnimationModified.h>`

Metody publiczne

- ColorAnimationModified (ColorGradient ∗gradient)
- template<class Animated >
  void operator() (Animated &animated, float progress) const

Atrybuty prywatne

- thor::ColorGradient ∗ m_gradient

6.1.1 Opis szczegółowy

Zmodyfikowana klasa ColorAnimation z biblioteki thor.

6.1.2 Dokumentacja konstruktora i destruktora

6.1.2.1 ColorAnimationModified()

```
thor::ColorAnimationModified::ColorAnimationModified (
            ColorGradient * gradient )  [explicit]
```

Konstruktor przyjmujący jako parametr wskaźnik to ColorGradient. Przy oznaczeniu konstruktora jako explicit, nie będzie przeprowadzana automatyczna konwersja przy pomocy tego konstruktora.

```
6                                                                   :
      m_gradient(gradient)
7 {
8
9 }
```

6.1.3 Dokumentacja funkcji składowych

6.1.3.1 operator()()

```
template<class Animated >
void thor::ColorAnimationModified::operator() (
             Animated & animated,
             float progress ) const
```

Szablon z parametrem klasy

Przeciążenei operatora (). Służy zmianie koloru danemu elementowi.

Parametry

animated	Parametr przekazujący adres do elementu, w którym ma być zmieniony kolor.
progress	Parametr który przyjmuje postęp w zakresie 0f do 100f.

```
49     {
50         thor::setColor(animated, m_gradient->sampleColor(progress));
51     }
```

6.1.4 Dokumentacja atrybutów składowych

6.1.4.1 m_gradient

```
thor::ColorGradient* thor::ColorAnimationModified::m_gradient  [private]
```

Zmienna prywatna, wskaźnik do ColorGradient.

Dokumentacja dla tej klasy została wygenerowana z plików:

- ColorAnimationModified.h
- ColorAnimationModified.cpp

6.2 Dokumentacja klasy ControlPanel

```
#include <ControlPanel.h>
```

Metody publiczne

- ControlPanel ()
- ~ControlPanel ()
- void draw ()
- void setTile (Tile *tile)

Atrybuty publiczne

- sf::Vector2f windowPosition
- sf::Vector2f windowSize

Atrybuty prywatne

- Options m_options
- Statistics m_statistics
- Tile * m_tile

6.2.1 Opis szczegółowy

Klasa odpowiadająca za rysowania głównego okna Panelu Sterowania.

6.2.2 Dokumentacja konstruktora i destruktora

6.2.2.1 ControlPanel()

```
ControlPanel::ControlPanel ( )
```

Domyślny konstruktor.

```
6 {
7     windowPosition = sf::Vector2f(0, 0);
8     windowSize = sf::Vector2f(0, 0);
9     m_options.setTile(m_tile);
10 }
```

6.2.2.2 ~ControlPanel()

```
ControlPanel::~ControlPanel ( )
```

Destruktor

```
14 {
15 }
```

6.2.3 Dokumentacja funkcji składowych

6.2.3.1 draw()

```
void ControlPanel::draw ( )
```

Metoda odpowiadająca za rysowanie (wyświetlanie) okna Panelu Sterowania.

```
18 {
19     ImGui::Begin("Panel Sterowania");
20 
21     if(!ImGui::CollapsingHeader("Ustawienia"))
22     {
23         m_options.draw();
24     }
25     if(ImGui::CollapsingHeader("Statystyki"))
26     {
27         m_statistics.draw();
28     }
29 
30 
31     windowPosition = sf::Vector2f(ImGui::GetWindowPos().x, ImGui::GetWindowPos().y);
32     windowSize = sf::Vector2f(ImGui::GetWindowSize().x, ImGui::GetWindowSize().y);
33 
34     ImGui::End();
35 
36 }
```

6.2.3.2 setTile()

```
void ControlPanel::setTile (
            Tile * tile )
```

Metoda odpowiadająca za ustawienie zmiennej prywatnej m_tile.

Parametry

tile	Przekazuje wskaźnik do siatki (Tile), do której będą odnosiły się opcje w panelu.

```
39 {
40     m_tile = tile;
41 }
```

6.2.4 Dokumentacja atrybutów składowych

6.2.4.1 m_options

```
Options ControlPanel::m_options  [private]
```

Zmienna, która odpowiada za zakładkę Opcje w Panelu Sterowania i zawiera wszystkie kontroli wchodzące w skład tej sekcji.

6.2.4.2 m_statistics

```
Statistics ControlPanel::m_statistics  [private]
```

Zmienna, która odpowiada za zakładkę Statystyki w Panelu sterowania i zawiera wszystkie kontroli wchodzące w skład tej sekcji.

6.2.4.3 m_tile

```
Tile* ControlPanel::m_tile  [private]
```

Zmienna do której przypisywany jest wskaźnik przez metodę: void ControlPanel::setTile(Tile *tile)

6.2.4.4 windowPosition

```
sf::Vector2f ControlPanel::windowPosition
```

Zmienna przechowująca pozycję okna Panelu Sterowania.

6.2.4.5 windowSize

```
sf::Vector2f ControlPanel::windowSize
```

Zmienna przechowująca rozmiar okna Panelu Sterowania.

Dokumentacja dla tej klasy została wygenerowana z plików:

- ControlPanel.h
- ControlPanel.cpp

6.3 Dokumentacja klasy Options

```
#include <Options.h>
```

Metody publiczne

- Options ()
- ~Options ()
- void draw ()
- void setTile (Tile *tile)

Atrybuty publiczne

- float infectionTime
- float resistanceTime
- float infectionGapTime
- std::vector< std::string > factors
- int selected
- ImVec4 infectedColor
- ImVec4 resistanceColor
- ImVec4 healthyColor

Atrybuty prywatne

- Tile * m_tile

6.3.1 Opis szczegółowy

Klasa służąca do obsługi i wyświetlania Opcji w Panelu Sterowania.

6.3.2 Dokumentacja konstruktora i destruktora

6.3.2.1 Options()

```
Options::Options ( )
```

Domyślny konstruktor.

```
50 {
51
52     infectionGapTime = SimulationSettings::getInstance()->
       lengthOfInfectionGap.asSeconds();
53     infectionTime = SimulationSettings::getInstance()->
       lengthOfInfection.asSeconds();
54     resistanceTime = SimulationSettings::getInstance()->
       lengthOfResistance.asSeconds();
55     //Przesłanie przez referencje
56     Utility().allDivisors(&factors);
57     infectedColor = ImColor(QuadSettings::getInstance()->
       infectedColor.r, QuadSettings::getInstance()->
       infectedColor.g, QuadSettings::getInstance()->
       infectedColor.b, QuadSettings::getInstance()->
       infectedColor.a);
58     resistanceColor = ImColor(QuadSettings::getInstance()->
       resistanceColor.r, QuadSettings::getInstance()->
       resistanceColor.g, QuadSettings::getInstance()->
       resistanceColor.b, QuadSettings::getInstance()->
       resistanceColor.a);
59     healthyColor = ImColor(QuadSettings::getInstance()->
       healthyColor.r, QuadSettings::getInstance()->
       healthyColor.g, QuadSettings::getInstance()->
       healthyColor.b, QuadSettings::getInstance()->
       healthyColor.a);
60
61
62
63 }
```

6.3.2.2 ~Options()

```
Options::~Options ( )
```

Destruktor.

```
67 {
68 }
```

6.3.3 Dokumentacja funkcji składowych

6.3.3.1 draw()

```
void Options::draw ( )
```

Metoda obsługująca rysowanie wszystkich elementów.

```
71 {
72
73     // Pausing logic of program
74     if (ImGui::Button("Play/Pause")) SimulationSettings::getInstance()->
       stopLogic = !SimulationSettings::getInstance()->
       stopLogic;
75     if (SimulationSettings::getInstance()->stopLogic &&
       SimulationSettings::getInstance()->globalClock.isRunning())
       SimulationSettings::getInstance()->globalClock.stop();
76     if (!SimulationSettings::getInstance()->stopLogic && !
       SimulationSettings::getInstance()->globalClock.isRunning())
       SimulationSettings::getInstance()->globalClock.start();
77
78     //Restarting simulation
79     if (ImGui::Button("Restart"))
80     {
81         if (isinf(QuadSettings::getInstance()->quadDimensions.x) || isinf(
       QuadSettings::getInstance()->quadDimensions.y))
       QuadSettings::getInstance()->quadDimensions = sf::Vector2f(10,10);
82         m_tile->init();
83     }
84
85     ImGui::Separator();
86
87     if (ImGui::Combo("Rozmiary komorek", &selected, factors))
88     {
89         QuadSettings::getInstance()->quadDimensions = sf::Vector2f(
       std::stoi(factors[selected]), std::stoi(factors[selected]));
90         m_tile->init();
91     }
92
93     if (ImGui::Button("+")) m_tile->add(); ImGui::SameLine(); if (ImGui::Button("-"))
       m_tile->remove(); ImGui::SameLine(); ImGui::Text("Dodawanie i usuwanie komorek");
94
95     ImGui::Separator();
96
97     ImGui::Checkbox("Punkt zarazenia", &SimulationSettings::getInstance()->
       pointOfInfection);
98
99     ImGui::Separator();
100
101     ImGui::Checkbox("Gradacja kolorow", &SimulationSettings::getInstance()->
       colorGradation);
102
103     ImGui::Separator();
104
105     ImGui::Text("Kolory komorek");
106
107
108
109     if (ImGui::ColorButton("Kolor zainfekowanej komorki", infectedColor)) ImGui::OpenPopup("
       InfectedColorPicker");
```

```
110     if(ImGui::BeginPopup("InfectedColorPicker"))
111     {
112         if (ImGui::ColorPicker3("Zarazone", (float*)&infectedColor))
113         {
114             QuadSettings::getInstance()->
    updateInfectedTextureColor(infectedColor);
115             QuadSettings::getInstance()->
    updateResistanceTextureColor(resistanceColor);
116         }
117
118
119         ImGui::EndPopup();
120     }
121     ImGui::SameLine();
122     ImGui::Text("Zarazona");
123     ImGui::SameLine();
124     if (ImGui::ColorButton("Kolor odpornej komorki", resistanceColor)) ImGui::OpenPopup("
    ResistedColorPicker");
125     if (ImGui::BeginPopup("ResistedColorPicker"))
126     {
127         if (ImGui::ColorPicker3("Odporna", (float*)&resistanceColor))
128         {
129             QuadSettings::getInstance()->
    updateResistanceTextureColor(resistanceColor);
130             QuadSettings::getInstance()->
    updateInfectedTextureColor(infectedColor);
131         }
132
133
134         ImGui::EndPopup();
135     }
136     ImGui::SameLine();
137     ImGui::Text("Odporna");
138     ImGui::SameLine();
139     if (ImGui::ColorButton("Kolor zdrowej komorki", healthyColor)) ImGui::OpenPopup("
    HealthyColorPicker");
140     if (ImGui::BeginPopup("HealthyColorPicker"))
141     {
142         if (ImGui::ColorPicker3("Zdrowa", (float*)&healthyColor))
143         {
144             QuadSettings::getInstance()->
    updateHealthyTextureColor(healthyColor);
145         }
146
147
148         ImGui::EndPopup();
149     }
150     ImGui::SameLine();
151     ImGui::Text("Zdrowa");
152
153     ImGui::Separator();
154
155     ImGui::Checkbox("Pojedyncze zarazenie", &SimulationSettings::getInstance
    ()->multiInfectionsOfQuad);
156     ImGui::SliderFloat("Prawdopodobienstwo", &SimulationSettings::getInstance
    ()->probabilityOfInfection, 0, 1, "%.1f");
157
158     ImGui::Separator();
159
160     ImGui::DragFloat("Czas infekcji", &infectionTime, 0.1, 0, 60);
161     ImGui::DragFloat("Czas odpornosci", &resistanceTime, 0.1, 0, 60);
162     ImGui::DragFloat("Czas pomiedzy infekcjami", &infectionGapTime, 0.1, 0, 60);
163
164     if(ImGui::Button("Zastosuj czasy"))
165     {
166         SimulationSettings::getInstance()->
    lengthOfResistance = sf::seconds(resistanceTime);
167         SimulationSettings::getInstance()->
    lengthOfInfection = sf::seconds(infectionTime);
168         SimulationSettings::getInstance()->
    lengthOfInfectionGap = sf::seconds(infectionGapTime);
169     }
170 }
```

6.3.3.2 setTile()

```
void Options::setTile (
            Tile * tile )
```

Metoda służąca do ustawienia wskaźnika zmiennej m_tile.

```
173 {
174     m_tile = tile;
175 }
```

6.3.4 Dokumentacja atrybutów składowych

6.3.4.1 factors

```
std::vector<std::string> Options::factors
```

Wektor przechowujący wszystkie całkowite dzielniki szerokośći i wysokości ekranu.

6.3.4.2 healthyColor

```
ImVec4 Options::healthyColor
```

Zmienna pomocnicza przechowująca kolor zdrowej komórki.

6.3.4.3 infectedColor

```
ImVec4 Options::infectedColor
```

Zmienna pomocnicza przechowująca kolor zarażonej komórki.

6.3.4.4 infectionGapTime

```
float Options::infectionGapTime
```

Zmienna pomocnicza przechowująza czas pomiędzy infekcjami.

6.3.4.5 infectionTime

```
float Options::infectionTime
```

Zmienna pomocnicza przechowująca czas infekcji.

6.3.4.6 m_tile

```
Tile* Options::m_tile  [private]
```

Zmienna przechowująca wskaźnik do siatki w której mają być wdrażane zmiany. Zmiennej przypisywany jest wskaźnik przez metodę: void Options::setTile(Tile *tile);

6.3.4.7 resistanceColor

```
ImVec4 Options::resistanceColor
```

Zmienna pomocnicza przechowująca kolor odpornej komórki.

6.3.4.8 resistanceTime

```
float Options::resistanceTime
```

Zmienna pomocnicza przechowująca czas odporności.

6.3.4.9 selected

```
int Options::selected
```

Zmienna pomocnicza przechowująca numer wybranego elementu z listy możliwych dzielników.

Dokumentacja dla tej klasy została wygenerowana z plików:

- Options.h
- Options.cpp

6.4 Dokumentacja klasy Quad

```
#include <Quad.h>
```

Metody publiczne

- Quad ()
- ~Quad ()
- void setInfected ()
- void updateState ()
- void setPosition (int x, int y)
- void updateNearbyElements ()
- sf::Sprite * getSprite ()

Atrybuty publiczne

- bool healthy
- bool infected
- bool resistant
- bool pointOfInfection
- sf::Time hitTime
- sf::Vector2u position
- thor::ColorAnimationModified p_animation

Metody prywatne

- void setHealthy ()
- void setResistance ()

Atrybuty prywatne

- sf::Sprite sprite
- bool m_nearbyInfected
- std::vector< Quad * > m_nearbyQuads
- sf::Sprite m_sprite

6.4.1 Opis szczegółowy

Klasa która reprezentuje komórkę i zarządza jej stanami.

6.4.2 Dokumentacja konstruktora i destruktora

6.4.2.1 Quad()

```
Quad::Quad ( )
```

Domyślny konstruktor.

```
5           :p_animation(&QuadSettings::getInstance()->infectionGradient)
6 {
7     healthy = true;
8     infected = false;
9     resistant = false;
10     m_nearbyInfected = false;
11     pointOfInfection = false;
12
13     sprite.setTexture(QuadSettings::getInstance()->transparentTexture);
14 }
```

6.4.2.2 ~Quad()

```
Quad::~Quad ( )
```

Destruktor

```
18 {
19
20 }
```

6.4.3 Dokumentacja funkcji składowych

6.4.3.1 getSprite()

```
sf::Sprite * Quad::getSprite ( )
```

Metoda zwracająca wskaźnik do sprite'a komórki.

Zwraca

sf::Sprite*

```
126 {
127     return &sprite;
128 }
```

6.4.3.2 setHealthy()

```
void Quad::setHealthy ( )  [private]
```

Metoda która ustawia komórkę w zdrowy stan.

```
23 {
24     healthy = true;
25     infected = false;
26     resistant = false;
27     sprite.setTexture(QuadSettings::getInstance()->transparentTexture);
28 
29 }
```

6.4.3.3 setInfected()

```
void Quad::setInfected ( )
```

Metoda która ustawia komórkę w stan zarażenia.

```
32 {
33     healthy = false;
34     infected = true;
35     resistant = false;
36     sprite.setTexture(QuadSettings::getInstance()->solidTexture);
37 
38     p_animation = thor::ColorAnimationModified(&
       QuadSettings::getInstance()->infectionGradient);
39     p_animation(sprite, 0.0f);
40 
41     hitTime = SimulationSettings::getInstance()->
       globalClock.getElapsedTime();
42 
43 
44     m_nearbyQuads.clear();
45     m_nearbyQuads.shrink_to_fit();
46 
47     Utility().generateNearbyElements(&
       TileSettings::getInstance()->tileQuads,&m_nearbyQuads,
       TileSettings::getInstance()->tileSize,position);
48 }
```

6.4.3.4 setPosition()

```
void Quad::setPosition (
            int x,
            int y )
```

Metoda odpowiadająca za ustawienie komórki na planszy.

Parametry

x	Wartość typu int opisująca położenie na osi x.
y	Wartość typu int opisująca położenie na osi y.

```
111 {
112     position = sf::Vector2u(x, y);
113     sprite.setPosition(x * QuadSettings::getInstance()->quadDimensions.x, y*
       QuadSettings::getInstance()->quadDimensions.y);
114 }
```

6.4.3.5 setResistance()

```
void Quad::setResistance ( )  [private]
```

Metoda która ustawia komórkę w stan odporności.

```
51 {
52     healthy = false;
53     infected = false;
54     resistant = true;
55     m_nearbyInfected = false;
56
57     sprite.setTexture(QuadSettings::getInstance()->solidTexture);
58
59     p_animation = thor::ColorAnimationModified(&
      QuadSettings::getInstance()->resistanceGradient);
60     p_animation(sprite, 0.0f);
61 }
```

6.4.3.6 updateNearbyElements()

```
void Quad::updateNearbyElements ( )
```

Metoda odpowiadająca za aktualizację wektora pobliskich elementów. Wektor jest czyszczony z elementów. Następnie jest wywoływana funkcja z klasy Utility o prototypie: void Utility::generateNearbyElements(std::vector<Quad> *tileQuads, std::vector<Quad>* ∗nearbyQuads, sf::Vector2u tileSize, sf::Vector2u position);

```
117 {
118     m_nearbyQuads.clear();
119     m_nearbyQuads.shrink_to_fit();
120
121     Utility().generateNearbyElements(&
      TileSettings::getInstance()->tileQuads, &m_nearbyQuads,
      TileSettings::getInstance()->tileSize, position);
122 }
```

6.4.3.7 updateState()

```
void Quad::updateState ( )
```

Metoda aktualizująca stan komórki. Sprawdza czasy poszczególnych etapów i ustawia odpowiedni stan komórki.

```
64 {
65     if (healthy) return;
66     if(infected || pointOfInfection)
67     {
68         if(SimulationSettings::getInstance()->globalClock.getElapsedTime() -
       hitTime < SimulationSettings::getInstance()->
       lengthOfInfection || pointOfInfection)
69         {
70             if (SimulationSettings::getInstance()->
       colorGradation)
71             {
72                 if (!pointOfInfection)p_animation(
       sprite, (SimulationSettings::getInstance()->globalClock.getElapsedTime
       () - hitTime) / SimulationSettings::getInstance()->
       lengthOfInfection);
73
74             }
75             else p_animation(sprite, 0.0f);
76             if(!m_nearbyInfected && !(
       SimulationSettings::getInstance()->globalClock.getElapsedTime() -
       hitTime < SimulationSettings::getInstance()->
       lengthOfInfectionGap))
77             {
78
79                 if (SimulationSettings::getInstance()->
       multiInfectionsOfQuad) m_nearbyInfected = true;
80                 for(auto &ne : m_nearbyQuads)
81                 {
82                     if(Utility().yesOrNo(
       SimulationSettings::getInstance()->probabilityOfInfection) && ne->healthy)
83                     {
84                         ne->setInfected();
85                         ne->hitTime = SimulationSettings::getInstance()->
       globalClock.getElapsedTime();
86                     }
87                 }
88             }
89         }else
90         {
91             setResistance();
92             hitTime = SimulationSettings::getInstance()->
       globalClock.getElapsedTime();
93         }
94     }else if(resistant)
95     {
96         if (SimulationSettings::getInstance()->globalClock.getElapsedTime()
       - hitTime < SimulationSettings::getInstance()->
       lengthOfResistance)
97         {
98             if (SimulationSettings::getInstance()->
       colorGradation)
99             {
100                 if (!pointOfInfection)p_animation(
       sprite, (SimulationSettings::getInstance()->globalClock.getElapsedTime
       () - hitTime) / SimulationSettings::getInstance()->
       lengthOfResistance);
101
102             }
103             else p_animation(sprite, 0.0f);
104             return;
105         }
106         else setHealthy();
107     }
108 }
```

6.4.4 Dokumentacja atrybutów składowych

6.4.4.1 healthy

```
bool Quad::healthy
```

Zmienna typu bool odpowiadająca za poinformowanie czy komórka jest zdrowa.

6.4.4.2 hitTime

```
sf::Time Quad::hitTime
```

Zmienna przechowująca czas w którym została zainfekowana.

6.4.4.3 infected

```
bool Quad::infected
```

Zmienna typu bool odpowiadająca za poinformowanie czy komórka jest zainfekowana.

6.4.4.4 m_nearbyInfected

```
bool Quad::m_nearbyInfected  [private]
```

Zmienna typu bool mająca na celu informowania czy sąsiednie komórki zostały zarażone.

6.4.4.5 m_nearbyQuads

```
std::vector<Quad*> Quad::m_nearbyQuads  [private]
```

Wektor psoiadający wskaźniki do sąsiednich elementów.

6.4.4.6 m_sprite

```
sf::Sprite Quad::m_sprite  [private]
```

Zmienna typu sf::Sprite posiadająca Sprite komórki.

6.4.4.7 p_animation

```
thor::ColorAnimationModified Quad::p_animation
```

Zmienna służąca animacji koloru komórki.

6.4.4.8 pointOfInfection

```
bool Quad::pointOfInfection
```

Zmienna typu bool odpowiadająca za poinformowanie czy komórka jest stałym punktem infekcji.

6.4.4.9 position

```
sf::Vector2u Quad::position
```

Zmienna opisująca położenie komórki na planszy.

6.4.4.10 resistant

```
bool Quad::resistant
```

Zmienna typu bool odpowiadająca za poinformowanie czy komórka jest odporna.

6.4.4.11 sprite

```
sf::Sprite Quad::sprite  [private]
```

Zmienna przechwoująca sprite komórki.

Dokumentacja dla tej klasy została wygenerowana z plików:

- Quad.h
- Quad.cpp

6.5 Dokumentacja klasy QuadSettings

```
#include <QuadSettings.h>
```

Metody publiczne

- void updateResistanceTextureColor (ImVec4 resisted)
- void updateInfectedTextureColor (ImVec4 infected)
- void updateHealthyTextureColor (ImVec4 healthy)
- void updateTextures ()

Statyczne metody publiczne

- static QuadSettings * getInstance ()

Atrybuty publiczne

- sf::Vector2f quadDimensions
- sf::Image transparentImage
- sf::Texture transparentTexture
- sf::Image solidImage
- sf::Texture solidTexture
- sf::Color infectedColor
- sf::Color resistanceColor
- sf::Color healthyColor
- thor::ColorGradient infectionGradient
- thor::ColorGradient resistanceGradient

Metody prywatne

- QuadSettings ()
- ~QuadSettings ()

Statyczne atrybuty prywatne

- static QuadSettings * s_quad_settings = nullptr

6.5.1 Opis szczegółowy

Singleton który przechowuje najważniejsze informacje dla komórki, które są współdzielone przez nie wszystkie.

6.5.2 Dokumentacja konstruktora i destruktora

6.5.2.1 QuadSettings()

```
QuadSettings::QuadSettings ( )  [private]
```

Domyślny konstruktor.

```
47 {
48     infectedColor = sf::Color::Red;
49     resistanceColor = sf::Color::Yellow;
50     healthyColor = sf::Color(255, 170, 128, 255);
51 
52     quadDimensions = sf::Vector2f(10, 10);
53 
54     solidImage.create(quadDimensions.x, quadDimensions.y,
     sf::Color::White);
55     solidTexture.loadFromImage(solidImage);
56 
57     transparentImage.create(quadDimensions.x,
     quadDimensions.y, sf::Color(0, 0, 0, 0));
58     transparentTexture.loadFromImage(transparentImage);
59 
60     infectionGradient[0.0f] = infectedColor;
61     infectionGradient[1.0f] = resistanceColor;
62 
63     resistanceGradient[0.0f] = resistanceColor;
64     resistanceGradient[1.0f] = healthyColor;
65 }
```

6.5.2.2 ~QuadSettings()

```
QuadSettings::~QuadSettings ( )  [private]
```

Destruktor

```
69 {
70     delete s_quad_settings;
71 }
```

6.5.3 Dokumentacja funkcji składowych

6.5.3.1 getInstance()

```
QuadSettings * QuadSettings::getInstance ( )  [static]
```

Metoda mająca na celu uzyskanie instancji singletonu.

Zwraca

QuadSettings*

```
6 {
7     if (!s_quad_settings)
8         s_quad_settings = new QuadSettings();
9     return s_quad_settings;
10
11 }
```

6.5.3.2 updateHealthyTextureColor()

```
void QuadSettings::updateHealthyTextureColor (
            ImVec4 healthy )
```

Metoda mająca na celu aktualizacje koloru zdrowej komórki oraz gradientów.

```
30 {
31     healthyColor = sf::Color(255 * healthy.x, 255 * healthy.y, 255 * healthy.z, 255 * healthy.w
       );
32     resistanceGradient[0.0f] = resistanceColor;
33     resistanceGradient[1.0f] = healthyColor;
34 }
```

6.5.3.3 updateInfectedTextureColor()

```
void QuadSettings::updateInfectedTextureColor (
            ImVec4 infected )
```

Metoda mająca na celu aktualizacje koloru zarażenia oraz gradientów.

```
22 {
23     infectedColor = sf::Color(255*infected.x, 255 * infected.y, 255 * infected.z, 255 *
       infected.w);
24     infectionGradient[0.0f] = infectedColor;
25     infectionGradient[1.0f] = resistanceColor;
26
27 }
```

6.5.3.4 updateResistanceTextureColor()

```
void QuadSettings::updateResistanceTextureColor (
            ImVec4 resisted )
```

Metoda mająca na celu aktualizacje koloru odporności oraz gradientów.

```
14 {
15     resistanceColor = sf::Color(255 * resisted.x, 255 * resisted.y, 255 * resisted.z, 255 *
      resisted.w);
16     resistanceGradient[0.0f] = resistanceColor;
17     resistanceGradient[1.0f] = healthyColor;
18 }
```

6.5.3.5 updateTextures()

```
void QuadSettings::updateTextures ( )
```

Metoda mająca na celu aktualizacje wymiarów tekstur i obrazków, czyli aktualizacji wielkosci komórki na planszy.

```
37 {
38     transparentImage.create(quadDimensions.x,
      quadDimensions.y, sf::Color(0, 0, 0, 0));
39     transparentTexture.loadFromImage(transparentImage);
40
41     solidImage.create(quadDimensions.x, quadDimensions.y,
      sf::Color::White);
42     solidTexture.loadFromImage(solidImage);
43
44 }
```

6.5.4 Dokumentacja atrybutów składowych

6.5.4.1 healthyColor

```
sf::Color QuadSettings::healthyColor
```

Zmienna przechowująca kolor dla zdrowej komórki.

6.5.4.2 infectedColor

```
sf::Color QuadSettings::infectedColor
```

Zmienna przechowująca kolor dla zarażonej komórki.

6.5.4.3 infectionGradient

```
thor::ColorGradient QuadSettings::infectionGradient
```

Zmienna przechowująca gradient dla zarażonej komórki.

6.5.4.4 quadDimensions

```
sf::Vector2f QuadSettings::quadDimensions
```

Zmienna przechowująca wymiary komórki.

6.5.4.5 resistanceColor

```
sf::Color QuadSettings::resistanceColor
```

Zmienna przechowująca kolor dla odpornej komórki.

6.5.4.6 resistanceGradient

```
thor::ColorGradient QuadSettings::resistanceGradient
```

Zmienna przechwoująca gradient dla odpornej komórki.

6.5.4.7 s_quad_settings

```
QuadSettings * QuadSettings::s_quad_settings = nullptr  [static], [private]
```

Zmienna statyczna która posiada jedyną instancje klasy QuadSettings. Pozyskiwana jest przez funkcję: static QuadSettings* QuadSettings::getInstance();

6.5.4.8 solidImage

```
sf::Image QuadSettings::solidImage
```

Zmienna przechowująca biały obrazek.

6.5.4.9 solidTexture

```
sf::Texture QuadSettings::solidTexture
```

Zmienna przechowująca białą teksturę.

6.5.4.10 transparentImage

```
sf::Image QuadSettings::transparentImage
```

Zmienna przechowująca przeźroczysty obraz.

6.5.4.11 transparentTexture

```
sf::Texture QuadSettings::transparentTexture
```

Zmienna przechowująca przeżroczystą teksturę.

Dokumentacja dla tej klasy została wygenerowana z plików:

- QuadSettings.h
- QuadSettings.cpp

6.6 Dokumentacja klasy SimulationSettings

```
#include <SimulationSettings.h>
```

Statyczne metody publiczne

- static SimulationSettings * getInstance ()

Atrybuty publiczne

- thor::StopWatch globalClock
- sf::Time lengthOfInfection
- sf::Time lengthOfResistance
- sf::Time lengthOfInfectionGap
- float probabilityOfInfection
- bool multiInfectionsOfQuad
- bool stopLogic
- bool pointOfInfection
- bool colorGradation
- float currentFPS

Metody prywatne

- SimulationSettings ()
- ~SimulationSettings ()

Statyczne atrybuty prywatne

- static SimulationSettings * s_simulation_settings = nullptr

6.6.1 Opis szczegółowy

Singleton, który przechwuje najważniejsze informacje o symulacji. Informacje wykorzystywane są przez wszystkie komórki oraz zakładkę Statystyki w Panelu Sterowania.

6.6.2 Dokumentacja konstruktora i destruktora

6.6.2.1 SimulationSettings()

```
SimulationSettings::SimulationSettings ( )  [private]
```

Domyślny konstruktor.

```
13 {
14     lengthOfInfection = sf::seconds(6);
15     lengthOfResistance = sf::seconds(3);
16     lengthOfInfectionGap = sf::seconds(0.1f);
17     globalClock.restart();
18 
19     probabilityOfInfection = 0.1f;
20 
21     multiInfectionsOfQuad = false;
22 
23     pointOfInfection = false;
24 
25     stopLogic = false;
26 
27     colorGradation = true;
28 
29     currentFPS = 0.f;
30 }
```

6.6.2.2 ~SimulationSettings()

```
SimulationSettings::~SimulationSettings ( )  [private]
```

Destruktor.

```
34 {
35     delete s_simulation_settings;
36 }
```

6.6.3 Dokumentacja funkcji składowych

6.6.3.1 getInstance()

```
SimulationSettings * SimulationSettings::getInstance ( )  [static]
```

Metoda zwracająca instancje klasy.

Zwraca

SimulationSettings*

```
6 {
7     if (!s_simulation_settings)
8         s_simulation_settings = new SimulationSettings();
9     return s_simulation_settings;
10 }
```

6.6.4 Dokumentacja atrybutów składowych

6.6.4.1 colorGradation

```
bool SimulationSettings::colorGradation
```

Zmienna, która włącza bądź wyłącza gradacje kolorów.

6.6.4.2 currentFPS

```
float SimulationSettings::currentFPS
```

Zmienna przechowująca aktualną liczbę klatek na sekundę.

6.6.4.3 globalClock

```
thor::StopWatch SimulationSettings::globalClock
```

Zegar globalny dla symulacji.

6.6.4.4 lengthOfInfection

```
sf::Time SimulationSettings::lengthOfInfection
```

Czas w sekundach, jak długo ma trwać choroba.

6.6.4.5 lengthOfInfectionGap

```
sf::Time SimulationSettings::lengthOfInfectionGap
```

Czas w sekundach pomiędzy kolejnymi infekcjami.

6.6.4.6 lengthOfResistance

```
sf::Time SimulationSettings::lengthOfResistance
```

Czas w sekundach, jak długo ma trwać odporność.

6.6.4.7 multiInfectionsOfQuad

```
bool SimulationSettings::multiInfectionsOfQuad
```

Zmienna, która mówi o tym czy jest dozwolone wielokrotne zarażanie przez komórkę.

6.6.4.8 pointOfInfection

```
bool SimulationSettings::pointOfInfection
```

Zmienna mówiąca o tym czy aktualnie zarażane komórki są stałymi punktami zarażenia.

6.6.4.9 probabilityOfInfection

```
float SimulationSettings::probabilityOfInfection
```

Zmienna przechowująca prawdopodobieństwo zarażenia.

6.6.4.10 s_simulation_settings

```
SimulationSettings * SimulationSettings::s_simulation_settings = nullptr  [static], [private]
```

Jedyna instancja klasy SimulationSettings, która jest udostępniana przez static SimulationSettings* SimulationSettings::getInstance()

6.6.4.11 stopLogic

```
bool SimulationSettings::stopLogic
```

Zmienna, która służy do zatrzymywania symulacji. Tylko i wyłącznie logika symulacji jest zatrzymywana.

Dokumentacja dla tej klasy została wygenerowana z plików:

- SimulationSettings.h
- SimulationSettings.cpp

6.7 Dokumentacja klasy Statistics

```
#include <Statistics.h>
```

Metody publiczne

- Statistics ()
- ~Statistics ()
- void draw ()

6.7.1 Opis szczegółowy

Klasa, która odpowiada za kartę Statystyki w Panelu Sterowania.

6.7.2 Dokumentacja konstruktora i destruktora

6.7.2.1 Statistics()

```
Statistics::Statistics ( )
```

Domyślny konstruktor.

```
6 {
7 }
```

6.7.2.2 ∼Statistics()

```
Statistics::~Statistics ( )
```

Destruktor.

```
11 {
12 }
```

6.7.3 Dokumentacja funkcji składowych

6.7.3.1 draw()

```
void Statistics::draw ( )
```

Metoda która rysuje zakładkę Statystyki.

```
15 {
16
17     ImGui::Text("Rozmiar okna symulacji:");
18     ImGui::SameLine();
19     ImGui::Text(std::to_string(TileSettings::getInstance()->mainWindowSize.x).
       c_str());
20     ImGui::SameLine();
21     ImGui::Text("px");
22     ImGui::SameLine();
23     ImGui::Text("x");
24     ImGui::SameLine();
25     ImGui::Text(std::to_string(TileSettings::getInstance()->mainWindowSize.y).
       c_str());
26     ImGui::SameLine();
27     ImGui::Text("px");
28
29     ImGui::Text("Ilosc kwadratow: ");
30     ImGui::SameLine();
31     ImGui::Text(std::to_string(TileSettings::getInstance()->tileSize.x *
       TileSettings::getInstance()->tileSize.y).c_str());
32
33     ImGui::Text("Rozmiar kwadratu: ");
34     ImGui::SameLine();
```

```
35      ImGui::Text(std::to_string(QuadSettings::getInstance()->quadDimensions.x).
        c_str());
36      ImGui::SameLine();
37      ImGui::Text("px");
38
39
40      ImGui::Text("Rozmiar siatki:");
41      ImGui::SameLine();
42      ImGui::Text(std::to_string(TileSettings::getInstance()->tileSize.x).c_str());
43      ImGui::SameLine();
44      ImGui::Text("elementow");
45      ImGui::SameLine();
46      ImGui::Text("na");
47      ImGui::SameLine();
48      ImGui::Text(std::to_string(TileSettings::getInstance()->tileSize.y).c_str());
49      ImGui::SameLine();
50      ImGui::Text("elementow");
51
52      ImGui::Text("FPS:");
53      ImGui::SameLine();
54      ImGui::Text(std::to_string(SimulationSettings::getInstance()->currentFPS
        ).c_str());
55
56
57  }
```

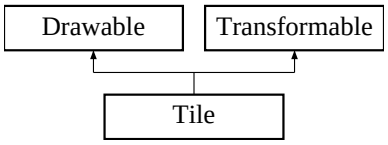
Dokumentacja dla tej klasy została wygenerowana z plików:

- Statistics.h
- Statistics.cpp

6.8 Dokumentacja klasy Tile

`#include <Tile.h>`

Diagram dziedziczenia dla Tile



Metody publiczne

- Tile ()
- void init ()
- void infect (sf::Vector2f mousePosition)
- void remove ()
- void add ()
- ~Tile ()

Metody prywatne

- void updatePositions ()
- virtual void draw (sf::RenderTarget &target, sf::RenderStates states) const

6.8.1 Opis szczegółowy

Klasa służąca zarządzaniu całą siatką kwadratów.

6.8.2 Dokumentacja konstruktora i destruktora

6.8.2.1 Tile()

```
Tile::Tile ( )
```

Domyślny konstruktor

```
6 {
7 }
```

6.8.2.2 ∼Tile()

```
Tile::∼Tile ( )
```

Destruktor

```
120 {
121
122 }
```

6.8.3 Dokumentacja funkcji składowych

6.8.3.1 add()

```
void Tile::add ( )
```

Metoda służąca dodawaniu komórek do siatki. Dodawana jest kolumna i wiersz na samych końcach.

```
67 {
68
69     if (TileSettings::getInstance()->tileSize.x ==
      TileSettings::getInstance()->mainWindowSize.x ||
      TileSettings::getInstance()->tileSize.y ==
      TileSettings::getInstance()->mainWindowSize.y) return;
70
71     if(TileSettings::getInstance()->tileQuads.empty())
72     {
73         TileSettings::getInstance()->tileSize = sf::Vector2u(1,1);
74         QuadSettings::getInstance()->quadDimensions = sf::Vector2f(
      TileSettings::getInstance()->mainWindowSize.x,
      TileSettings::getInstance()->mainWindowSize.y);
75         QuadSettings::getInstance()->updateTextures();
76         Quad quad;
77         quad.setPosition(0, 0);
78         quad.getSprite()->setTextureRect(sf::IntRect(0, 0,
      QuadSettings::getInstance()->quadDimensions.x,
      QuadSettings::getInstance()->quadDimensions.y));
79         TileSettings::getInstance()->tileQuads.push_back(quad);
80
81         return;
82     }
83
84     TileSettings::getInstance()->tileSize = sf::Vector2u(
      TileSettings::getInstance()->tileSize.x,
      TileSettings::getInstance()->tileSize.y + 1);
85
86     for (unsigned int i = 0; i < TileSettings::getInstance()->
      tileSize.x; i++)
87     {
88         Quad quad;
89         quad.setPosition(i, TileSettings::getInstance()->tileSize.y - 1
      );
90         unsigned int index = (TileSettings::getInstance()->
      tileSize.y - 1) * (i + 1) + i;
91         if(index == TileSettings::getInstance()->
      tileQuads.size())
92         {
93             TileSettings::getInstance()->tileQuads.push_back(quad);
94         }else
95         {
96             TileSettings::getInstance()->tileQuads.insert(
      TileSettings::getInstance()->tileQuads.begin() + index, quad);
97         }
98     }
99
100     TileSettings::getInstance()->tileSize = sf::Vector2u(
      TileSettings::getInstance()->tileSize.x + 1,
      TileSettings::getInstance()->tileSize.y);
101
102     for (unsigned int j = 0; j < TileSettings::getInstance()->
      tileSize.y; j++)
103     {
104         Quad quad;
105         quad.setPosition(TileSettings::getInstance()->tileSize.x - 1, j
      );
106         TileSettings::getInstance()->tileQuads.push_back(quad);
107     }
108
109     // Dodanie tego do funkcji updatePosition();
110     QuadSettings::getInstance()->quadDimensions = sf::Vector2f(float
      (TileSettings::getInstance()->mainWindowSize.x) /
      TileSettings::getInstance()->tileSize.x, float(
      TileSettings::getInstance()->mainWindowSize.y) /
      TileSettings::getInstance()->tileSize.y);
111     QuadSettings::getInstance()->updateTextures();
112
113     updatePositions();
114
115
116 }
```

6.8.3.2 draw()

```
virtual void Tile::draw (
            sf::RenderTarget & target,
            sf::RenderStates states ) const  [inline], [private], [virtual]
```

Metoda, która została nadpisana z sf::Drawable

Parametry

target	RenderTarget do którego będzie rysowane
states	Aktualny stan rysowania

```
54     {
55         states.transform *= getTransform();
56 
57         for (auto &element : TileSettings::getInstance()->tileQuads) {
58             if(!SimulationSettings::getInstance()->stopLogic && !element.
     healthy)
59             {
60                 element.updateState();
61             }
62             target.draw(*element.getSprite(), states);
63         }
64 
65     }
```

6.8.3.3 infect()

```
void Tile::infect (
            sf::Vector2f mousePosition )
```

Metoda służąca infekowaniu komórki

Parametry

mousePosition	Aktualna pozycja kursora na planszy.

```
34 {
35     for (auto &element : TileSettings::getInstance()->tileQuads) {
36         if (element.getSprite()->getGlobalBounds().contains(mousePosition) /*&& element.healthy */) {
37             if(SimulationSettings::getInstance()->
     pointOfInfection)element.pointOfInfection = true;
38             element.setInfected();
39         }
40     }
41 }
```

6.8.3.4 init()

```
void Tile::init ( )
```

Metoda służąca inicjacji zmiennych oraz domyślnej siatki.

```
10 {
11
12     TileSettings::getInstance()->tileSize = sf::Vector2u(
       TileSettings::getInstance()->mainWindowSize.x/
       QuadSettings::getInstance()->quadDimensions.x,
13         TileSettings::getInstance()->mainWindowSize.y /
       QuadSettings::getInstance()->quadDimensions.y);
14     TileSettings::getInstance()->tileQuads.clear();
15     TileSettings::getInstance()->tileQuads.shrink_to_fit();
16     TileSettings::getInstance()->tileQuads.reserve(
       TileSettings::getInstance()->tileSize.x *
       TileSettings::getInstance()->tileSize.y);
17
18     QuadSettings::getInstance()->updateTextures();
19
20         for(unsigned int i = 0; i < TileSettings::getInstance()->
       tileSize.x; i++)
21         {
22             for(unsigned int j = 0; j < TileSettings::getInstance()->
       tileSize.y; j++)
23             {
24                 Quad quad;
25                 quad.setPosition(i, j);
26                 TileSettings::getInstance()->tileQuads.push_back(quad);
27             }
28         }
29
30
31 }
```

6.8.3.5 remove()

```
void Tile::remove ( )
```

Metoda służąca usuwaniu komórek z siatki. Usuwana jest ostatnia kolumna i wiersz.

```
44 {
45
46     if (TileSettings::getInstance()->tileSize.x == 0 ||
       TileSettings::getInstance()->tileSize.y == 0) return;
47
48
49     TileSettings::getInstance()->tileQuads.erase(
       TileSettings::getInstance()->tileQuads.end() -
       TileSettings::getInstance()->tileSize.y,
       TileSettings::getInstance()->tileQuads.end());
50     TileSettings::getInstance()->tileQuads.shrink_to_fit();
51     for (unsigned int i = 0; i < TileSettings::getInstance()->
       tileSize.y - 1; i++)
52     {
53         int index = TileSettings::getInstance()->
       tileSize.y + i * TileSettings::getInstance()->
       tileSize.y - (i + 1);
54         TileSettings::getInstance()->tileQuads.erase(
       TileSettings::getInstance()->tileQuads.begin() + index);
55     }
56     TileSettings::getInstance()->tileQuads.shrink_to_fit();
57     TileSettings::getInstance()->tileSize = sf::Vector2u(
       TileSettings::getInstance()->tileSize.x -1,
       TileSettings::getInstance()->tileSize.y -1);
58
59     QuadSettings::getInstance()->quadDimensions = sf::Vector2f(float
       (TileSettings::getInstance()->mainWindowSize.x) /
       TileSettings::getInstance()->tileSize.x, float(
       TileSettings::getInstance()->mainWindowSize.y) /
       TileSettings::getInstance()->tileSize.y);
60     QuadSettings::getInstance()->updateTextures();
61
62     updatePositions();
63
64 }
```

6.8.3.6 updatePositions()

```
void Tile::updatePositions ( )  [private]
```

Metoda służąca aktualizacji pozycji komórek na planszy oraz ich rozmiaru.

```
125 {
126     for (unsigned int i = 0; i < TileSettings::getInstance()->
    tileSize.x; i++)
127     {
128         for (unsigned int j = 0; j < TileSettings::getInstance()->
    tileSize.y; j++)
129         {
130             Quad *quad = &TileSettings::getInstance()->
    tileQuads.at(j + i * TileSettings::getInstance()->tileSize.y);
131             quad->setPosition(i, j);
132             quad->getSprite()->setTextureRect(sf::IntRect(0, 0,
    QuadSettings::getInstance()->quadDimensions.x,
    QuadSettings::getInstance()->quadDimensions.y));
133             if (quad->infected) quad->updateNearbyElements();
134         }
135     }
136 }
```

Dokumentacja dla tej klasy została wygenerowana z plików:

- Tile.h
- Tile.cpp

6.9 Dokumentacja klasy TileSettings

```
#include <TileSettings.h>
```

Statyczne metody publiczne

- static TileSettings * getInstance ()

Atrybuty publiczne

- sf::Vector2u tileSize
- std::vector< Quad > tileQuads
- sf::Vector2u mainWindowSize

Metody prywatne

- TileSettings ()
- ~TileSettings ()

Statyczne atrybuty prywatne

- static TileSettings * s_tile_settings = nullptr

6.9.1 Opis szczegółowy

Singleton odpowiadający za ustawienia siatki komórek. Przechowuje najważniejsze zmienne.

6.9.2 Dokumentacja konstruktora i destruktora

6.9.2.1 TileSettings()

```
TileSettings::TileSettings ( )  [private]
```

Domyślny konstruktor.

```
13 {
14     tileQuads.clear();
15     tileQuads.shrink_to_fit();
16 
17     tileSize = sf::Vector2u(0, 0);
18 
19     mainWindowSize = sf::Vector2u(0, 0);
20 }
```

6.9.2.2 ~TileSettings()

```
TileSettings::~TileSettings ( )  [private]
```

Destruktor.

```
24 {
25     delete s_tile_settings;
26 }
```

6.9.3 Dokumentacja funkcji składowych

6.9.3.1 getInstance()

```
TileSettings * TileSettings::getInstance ( )  [static]
```

Metoda zwracająca instancje klasy.

Zwraca

TileSettings*

```
6 {
7     if (!s_tile_settings)
8         s_tile_settings = new TileSettings();
9     return s_tile_settings;
10 }
```

6.9.4 Dokumentacja atrybutów składowych

6.9.4.1 mainWindowSize

```
sf::Vector2u TileSettings::mainWindowSize
```

Zmienna przechowująca rozmiar głównego okna.

6.9.4.2 s_tile_settings

```
TileSettings * TileSettings::s_tile_settings = nullptr  [static], [private]
```

Jedyna instancja klasy TileSettings, która jest udostępniana przez static TileSettings* TileSettings::getInstance()

6.9.4.3 tileQuads

```
std::vector<Quad> TileSettings::tileQuads
```

Wektor przechowujący wszystkie komórki.

6.9.4.4 tileSize

```
sf::Vector2u TileSettings::tileSize
```

Zmienna opisująca szerokość i długość siatki.

Dokumentacja dla tej klasy została wygenerowana z plików:

- TileSettings.h
- TileSettings.cpp

6.10 Dokumentacja klasy Utility

```
#include <Utility.h>
```

Metody publiczne

- Utility ()
- ~Utility ()
- bool yesOrNo (float probability)
- void generateNearbyElements (std::vector< Quad > *tileQuads, std::vector< Quad *> *nearbyQuads, sf::Vector2u tileSize, sf::Vector2u position)
- bool clickedInWindow (sf::Vector2f windowSize, sf::Vector2f windowPosition, sf::Vector2u mousePosition)
- void allDivisors (std::vector< std::string > *v_factors)

Metody prywatne

- bool existQuad (sf::Vector2u point, sf::Vector2u tileSize)

6.10.1 Opis szczegółowy

Klasa posiadająca podręczne metody dla symulacji.

6.10.2 Dokumentacja konstruktora i destruktora

6.10.2.1 Utility()

```
Utility::Utility ( )
```

Domyślny konstruktor.

```
7 {
8 }
```

6.10.2.2 ∼Utility()

```
Utility::~Utility ( )
```

Destruktor.

```
12 {
13 }
```

6.10.3 Dokumentacja funkcji składowych

6.10.3.1 allDivisors()

```
void Utility::allDivisors (
            std::vector< std::string > * v_factors )
```

Metoda oblicza wszystkie wspólne dzielniki szerokości i długośći głównego okna.

Parametry

∗v_factors	wskaźnik do wektora stringów

```
59 {
60     int n, m;
61 
62 
63     v_factors->clear();
64     v_factors->shrink_to_fit();
65 
66 
67     if(TileSettings::getInstance()->mainWindowSize.x >
      TileSettings::getInstance()->mainWindowSize.y )
68     {
69         m = TileSettings::getInstance()->mainWindowSize.x;
70         n = TileSettings::getInstance()->mainWindowSize.y;
71     }else
72     {
73         n = TileSettings::getInstance()->mainWindowSize.x;
74         m = TileSettings::getInstance()->mainWindowSize.y;
75     }
76 
77 
78     for (int i = 1; i <= n; i++)
79     {
80         if (n%i == 0 && m%i == 0)
81         {
82             v_factors->push_back(std::to_string(n/i));
83         }
84     }
85 }
```

6.10.3.2 clickedInWindow()

```
bool Utility::clickedInWindow (
             sf::Vector2f windowSize,
             sf::Vector2f windowPosition,
             sf::Vector2u mousePosition )
```

Metoda zwracająca bool. Sprawdza czy kliknięcie zostało zarejestrowane np. w oknie Panelu Sterowania.

Parametry

windowSize	Rozmiar okna
windowPosition	Pozycja okna na ekranie.
mousePosition	Pozycja kursora na ekranie.

Zwraca

bool

```
52 {
53     if (mousePosition.x >= windowPosition.x && mousePosition.x <= windowPosition.x + windowSize.x &&
      mousePosition.y >= windowPosition.y && mousePosition.y <= windowPosition.y + windowSize.y) return true;
54     return false;
55 }
```

6.10.3.3 existQuad()

```
bool Utility::existQuad (
            sf::Vector2u point,
            sf::Vector2u tileSize )  [private]
```

Metoda zwracająca bool, sprawdza czy dana komórka na danej pozycji istnieje.

Parametry

point	Miejsce, w którym poszukiwana jest komórka.
tileSize	Rozmiar siatki komórek.

```
88 {
89
90     if (point.x >= 0 && point.x < tileSize.x && point.y >= 0 && point.y < tileSize.y && (point.y + point.x*
       tileSize.x) < tileSize.x*tileSize.y) return true;
91     return false;
92 }
```

6.10.3.4 generateNearbyElements()

```
void Utility::generateNearbyElements (
            std::vector< Quad > * tileQuads,
            std::vector< Quad *> * nearbyQuads,
            sf::Vector2u tileSize,
            sf::Vector2u position )
```

Funkcja służąca generowaniu sąsiednich elementów komórki.

Parametry

∗*tileQuads*	wskaźnik do wektora komórek.
∗*nearbyQuads*	wskaźnik do wektora wskaźników sąsiednich komórek.
tileSize	Rozmiar siatki komórek.
position	Pozycja komórki na siatce.

```
22 {
23
24     //Left top
25     if (existQuad(sf::Vector2u(position.x - 1, position.y - 1), tileSize))
26         nearbyQuads->push_back(&tileQuads->at((position.y - 1) + ((position.x - 1)*tileSize.y)));
27     //Left
28     if (existQuad(sf::Vector2u(position.x - 1, position.y), tileSize))
29         nearbyQuads->push_back(&tileQuads->at((position.y) + ((position.x - 1)*tileSize.y)));
30     //Left bottom
31     if (existQuad(sf::Vector2u(position.x - 1, position.y + 1), tileSize))
32         nearbyQuads->push_back(&tileQuads->at((position.y + 1) + ((position.x - 1)*tileSize.y)));
33     //Top
34     if (existQuad(sf::Vector2u(position.x, position.y - 1), tileSize))
35         nearbyQuads->push_back(&tileQuads->at((position.y - 1) + ((position.x)*tileSize.y)));
36     //Bottom
37     if (existQuad(sf::Vector2u(position.x, position.y + 1), tileSize))
38         nearbyQuads->push_back(&tileQuads->at((position.y + 1) + ((position.x)*tileSize.y)));
39     //Right top
40     if (existQuad(sf::Vector2u(position.x + 1, position.y - 1), tileSize))
```

```
41          nearbyQuads->push_back(&tileQuads->at((position.y - 1) + ((position.x + 1)*tileSize.y)));
42      //Right
43      if (existQuad(sf::Vector2u(position.x + 1, position.y), tileSize))
44          nearbyQuads->push_back(&tileQuads->at((position.y) + ((position.x + 1)*tileSize.y)));
45      //Right bottom
46      if (existQuad(sf::Vector2u(position.x + 1, position.y + 1), tileSize))
47          nearbyQuads->push_back(&tileQuads->at((position.y + 1) + ((position.x + 1)*tileSize.y)));
48
49  }
```

6.10.3.5 yesOrNo()

```
bool Utility::yesOrNo (
            float probability )
```

Funkcja zwracająca bool. Jako parametr jest podawane prawdopodobieństwo wystąpienia prawdy.

Parametry

probability	Prawdopodobieństwo wystąpienia prawdy.

Zwraca

bool

```
16  {
17      return rand() % 100 < (probability * 100);
18  }
```

Dokumentacja dla tej klasy została wygenerowana z plików:

- Utility.h
- Utility.cpp

Rozdział 7

Dokumentacja plików

7.1 Dokumentacja pliku ColorAnimationModified.cpp

```
#include "ColorAnimationModified.h"
```

7.2 Dokumentacja pliku ColorAnimationModified.h

```
#include <Thor/Config.hpp>
#include <Thor/Graphics/ColorGradient.hpp>
#include <Thor/Graphics/UniformAccess.hpp>
```

Komponenty

- class thor::ColorAnimationModified

Przestrzenie nazw

- thor

7.3 Dokumentacja pliku ControlPanel.cpp

```
#include "ControlPanel.h"
```

7.4 Dokumentacja pliku ControlPanel.h

```
#include <imgui.h>
#include <SFML/System/Vector2.hpp>
#include "Tile.h"
#include "Options.h"
#include "Statistics.h"
```

Komponenty

- class ControlPanel

7.5 Dokumentacja pliku Main.cpp

```
#include <SFML/Graphics.hpp>
#include <imgui.h>
#include <sfml-imgui/sfml-imgui/imgui-SFML.hpp>
#include <sfml-imgui/sfml-imgui/imconfig-SFML.hpp>
#include "TileSettings.h"
#include "SimulationSettings.h"
#include "QuadSettings.h"
#include "Tile.h"
#include "Options.h"
#include "ControlPanel.h"
```

Funkcje

- int main ()

7.5.1 Dokumentacja funkcji

7.5.1.1 main()

```
int main ( )
```

Funkcj główna main. Zawarte jest w niej główna pętla aplikacji.

Na początku tworzone jest okno główne. Ilość klatek blokowana jest na odświeżanie ekranu.

Następnie deklarowane są najważniejsze zmienne. Następnie inicjowane jest okno Panelu Sterowania. Następnie inicjowane jest siatka wraz z Panelem Sterowania.

Na samym końcu znajduje się główna pętla w której jest pętla odpowiadająca za przechwytywanie zdarzeń (np. kliknięcie myszką). Pod koniec pętli rysowane są wszystkie elementy.

```
25 {
26     sf::RenderWindow mainWindow(sf::VideoMode(800,800), "Liszaj", sf::Style::Close);
27     mainWindow.setVerticalSyncEnabled(true);
28     sf::Event mainWindowEvent;
29     sf::Clock deltaClockImGui;
30     sf::Clock deltaClockFPS;
31
32
33
34     TileSettings::getInstance()->mainWindowSize = mainWindow.getSize
       ();
35
36     ImGui::SFML::Init(mainWindow);
37
38     Tile tile;
```

```
39      tile.init();
40
41      ControlPanel controlPanel;
42      controlPanel.setTile(&tile);
43
44
45
46      while(mainWindow.isOpen())
47      {
48
49          while(mainWindow.pollEvent(mainWindowEvent))
50          {
51              ImGui::SFML::ProcessEvent(mainWindowEvent);
52              if (mainWindowEvent.type == sf::Event::Closed) mainWindow.close();
53              if(mainWindowEvent.type == sf::Event::MouseButtonPressed && mainWindowEvent.mouseButton.button
    == sf::Mouse::Button::Left &&
54                  !Utility().clickedInWindow(controlPanel.windowSize, controlPanel.
    windowPosition, sf::Vector2u(mainWindowEvent.mouseButton.x, mainWindowEvent.mouseButton.y)) )
55                  tile.infect(sf::Vector2f(mainWindowEvent.mouseButton.x, mainWindowEvent.mouseButton.y
    ));
56
57          }
58          ImGui::SFML::Update(mainWindow, deltaClockImGui.restart());
59
60          mainWindow.clear(QuadSettings::getInstance()->healthyColor);
61          mainWindow.draw(tile);
62          controlPanel.draw();
63
64          ImGui::SFML::Render(mainWindow);
65
66          SimulationSettings::getInstance()->
    currentFPS = 1 / deltaClockFPS.restart().asSeconds();
67
68          mainWindow.display();
69      }
70      ImGui::SFML::Shutdown();
71      return 0;
72  }
```

7.6 Dokumentacja pliku Options.cpp

```
#include "Options.h"
```

Przestrzenie nazw

- ImGui

Funkcje

- bool ImGui::Combo (const char *label, int *currIndex, std::vector< std::string > &values)

Zmienne

- static auto ImGui::vector_getter

7.7 Dokumentacja pliku Options.h

```
#include <SFML/System/Vector2.hpp>
#include <imgui.h>
#include "SimulationSettings.h"
#include "Tile.h"
```

Komponenty

- class Options

7.8 Dokumentacja pliku Quad.cpp

```
#include "Quad.h"
```

7.9 Dokumentacja pliku Quad.h

```
#include <SFML/Graphics/Sprite.hpp>
#include <SFML/System/Time.hpp>
#include <Thor/Time.hpp>
#include <Thor/Animations/ColorAnimation.hpp>
#include <vector>
#include "Utility.h"
#include "SimulationSettings.h"
#include "TileSettings.h"
#include "QuadSettings.h"
#include "ColorAnimationModified.h"
```

Komponenty

- class Quad

7.10 Dokumentacja pliku QuadSettings.cpp

```
#include "QuadSettings.h"
```

7.11 Dokumentacja pliku QuadSettings.h

```
#include <SFML/System/Vector2.hpp>
#include <SFML/Graphics/Image.hpp>
#include <SFML/Graphics/Texture.hpp>
#include <Thor/Animations/ColorAnimation.hpp>
#include <imgui.h>
```

Komponenty

- class QuadSettings

7.12 Dokumentacja pliku SimulationSettings.cpp

```
#include "SimulationSettings.h"
```

7.13 Dokumentacja pliku SimulationSettings.h

```
#include <SFML/System/Time.hpp>
#include <Thor/Time/StopWatch.hpp>
```

Komponenty

- class SimulationSettings

7.14 Dokumentacja pliku Statistics.cpp

```
#include "Statistics.h"
#include "TileSettings.h"
```

7.15 Dokumentacja pliku Statistics.h

```
#include <sfml-imgui/sfml-imgui/imgui-SFML.hpp>
#include <imgui.h>
```

Komponenty

- class Statistics

7.16 Dokumentacja pliku Tile.cpp

```
#include "Tile.h"
```

7.17 Dokumentacja pliku Tile.h

```
#include <SFML/Graphics/Drawable.hpp>
#include <SFML/Graphics/Transformable.hpp>
#include "TileSettings.h"
#include "QuadSettings.h"
#include <SFML/Graphics/RenderTarget.hpp>
```

Komponenty

- class Tile

7.18 Dokumentacja pliku TileSettings.cpp

```
#include "TileSettings.h"
```

7.19 Dokumentacja pliku TileSettings.h

```
#include <SFML/System/Vector2.hpp>
#include <iostream>
#include "Quad.h"
```

Komponenty

- class TileSettings

7.20 Dokumentacja pliku Utility.cpp

```
#include "Utility.h"
```

7.21 Dokumentacja pliku Utility.h

```
#include <iostream>
#include "Quad.h"
```

Komponenty

- class Utility

Skorowidz