Zaawansowane programowanie w języku C++

Saper

Stanislau Antanovich
Mykhailo Buzdyhan
Vladyslav Gotovchykov

Spis treści

1 Realizacja

1.1 Klasa *MyText*

```
#include <iostream>
#include <sstream>
#include <SFML/Graphics.hpp>

using namespace std;
using namespace sf;

class MyText {
private:
    string shribe;
    Font font;

public:
    sf::Text txt;

    MyText(string shribeName);

    MyText();

    void sleditForSprite(Sprite& s, float x, float y);

    void getChislo(float n);

    void setString(string name);

    void setPosition(float x, float y);

    void setFillTextColor(float R, float G, float B);

    void setCharacterSize(float a);

    void draw(RenderWindow& window);
};
```

1.1.1 Prywatne pola

- **string shribe**: Łańcuch znaków przechowujący początkowy tekts
- **Font font**: Obiekt klasy SFML *Font*, który służy do ładowania i używania określonej czcionki

1.1.2 Publiczne pola

- **sf::Text txt**: Obiekt klasy SFML *Text*, który repzenetuje tekst do wyświetlenia

1.1.3 Konstruktory

- **MyText(string shribeName)**
  - **Cel**: Inicjalizuje obiekt *MyText* z podanym łańcuchem znaków **shribeName**
  - **Parametry**:
    - **sting shribeName**; Początkowy tekst do wyświetlenia

- **Operacje**
  - Ładuje czcionkę z pliku "*fonts/bloodcrow.ttf*"
  - Ustawia czcionkę, rozmiar znaków(20) oraz kolor wypełnienia dla obiektu *text*
  - Ustawia początkowy tekst na "*shribeName*"

- **MyText()**
  - **Cel**: Konstruktor domyślny, który inicjalizuje obiekt **MyText** pustym łańcuchem znaków
  - **Operacje**:
    - Ładuje czcionkę z pliku "*fonts/bloodcrow.ttf*"
    - Ustawia czcionkę, rozmiar znaków(20) oraz kolor wypełnienia dla obiektu *text*
    - Ustawia początkowy tekst na pusty łańcuch znaków

1.1.4 Funkcje członkowskie

- **void sleditForSprite(Sprite& s, float x, float y)**
  - **Cel**: Ustawia pozycję tekstu względem pozycji podanego sprite'a *s*
  - **Parametry**:
    - **Sprite& s**: Referencja do obiektu klasy SFML *Sprite*
    - **float x**: Przesunięcie w osi X względem pozycji sprite'a
    - **float y**: Przesunięcie w osi Y względem pozycji sprite'a
  - **Operacje**: Ustawia pozycję obiektu *txt* na podstawie pozycji sprite'a oraz podanych przesunięć

- **void getChislo(float n)**
  - **Cel**: Dodaje wartość liczbową *n* do istniejącego tekstu
  - **Parametry**: **float n**: Liczba, która zostanie przekonwertowana na łańcuch znaków i dodana
  - **Operacje**:
    - Konwertuje liczbę *n* na łańcuch znaków przy użyciu *ostringstream*
    - Ustawia tekst obiektu *txt* na oryginalny łańcuch *shribe* połączony z łańcuchem reprezentującym liczbę

- **void setString(string name)**
  - **Cel**: Ustawia tekst na określony łańcuch znaków *name*
  - **Parametry**: **string name**: Nowy tekst do wyświetlenia
  - **Operacje**: Ustawia tekst obiektu *txt* na *name*

- **void setPosition(float x, float y)**
  - **Cel**: Ustawia pozycję tekstu
  - **Parametry**:
    - **float x**: Współrzędna X pozycji tekstu

        * **float y**: Współrzędna Y pozycji tekstu
    - **Operacje**: Ustawia pozycję obiektu *txt* na (x, y)

- **void setFillTextColor(float R, float G, float B)**
    - **Cel**: Ustawia kolor tekstu
    - **Parametry**:
        * **float R**: Składowa czerwona koloru (0-255)
        * **float G**: Składowa zielona koloru (0-255)
        * **float B**: Składowa niebieska koloru (0-255)
    - **Operacje**: Ustawia kolor wypełnienia obiektu *txt* przy użyciu wartości RGB

- **void setCharacterSize(float a)**
    - **Cel**: Ustawia rozmiar znaków tekstu
    - **Parametry**: **float a**: Nowy rozmiar znaków
    - **Operacje**: Ustawia rozmiar znaków obiektu *txt* na *a*

- **void draw(RenderWindow& window)**
    - **Cel**: Rysuje tekst w podanym oknie renderowania
    - **Parametry**:
        * **RenderWindow& window**: Referencja do obiektu klasy SFML **RenderWindow**
    - **Operacje**: Rysuje obiekt *txt* w podanym oknie renderowania

1.1.5 Wnioski

Klasa MyText kapsułkuje funkcjonalności związane z wyświetlaniem tekstu w bibliotece SFML, zapewniając metody do inicjalizacji, ustawiania pozycji, koloru, rozmiaru znaków oraz renderowania tekstu w graficznej aplikacji. Konstruktory ustawiają czcionkę oraz początkowy tekst, a funkcje członkowskie umożliwiają dynamiczne aktualizacje i wyświetlanie tekstu w oknie aplikacji.

1.2 Klasa *Button*

```
#include <iostream>
#include <sstream>
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
#include "MyText.h"

using namespace std;
using namespace sf;

class Button : public MyText {
private:
    float w, h;
    SoundBuffer buffer;
    Sound sound;
    bool press;
```

```
16
17 public:
18     RectangleShape button;
19
20     Button(float W, float H, string shribeName);
21
22     void sleditForSprite(Sprite& s, float x, float y);
23
24     void draw(RenderWindow& window);
25
26     bool pressed(Event& event, Vector2i mousePosition);
27
28     bool navediaMouse(Event& event, Vector2i mousePosition);
29
30     void getSound(string failAudio);
31
32     void soundPlay();
33
34     void soundSetVolume(int volume);
35
36     void setButtonSize(float W, float H);
37
38     void setOringCenter();
39
40     void setPosition(float x, float y);
41
42     void setFillRacktengelColor(float R, float G, float B);
43 };
44
45 #endif // BUTTON_H
```

1.2.1 Prywatne pola

- **float w, h**: Szerokość i wysokość przycisku
- **SoundBuffer buffer**: Bufor dźwięku używany do przechowywania danych dźwiękowych
- **Sound sound**: Obiekt dźwięku, który odtwarza dźwięk
- **bool press**: Flaga oznaczająca, czy przycisk jest wciśnięty

1.2.2 Publiczne pola

- **RectangleShape button**: Obiekt kształtu prostokąta, który reprezentuje wizualny wygląd przycisku

1.2.3 Konstruktory

- **Button(float W, float H, string shribeName)**
  - **Cel**: Inicjalizuje obiekt Button z podanymi szerokością W, wysokością H oraz tekstem shribeName
  - **Parametry**
    - float W: Szerokość przycisku
    - float H: Wysokość przycisku
    - string shribeName: Tekst wyświetlany na przycisku

- **Operacje**
  - Wywołuje konstruktor klasy bazowej MyText z shribeName
  - Ustawia szerokość w i wysokość h
  - Ustawia flagę press na false
  - Ustawia rozmiar prostokąta button
  - Ustawia pozycję tekstu txt na pozycję prostokąta button

1.2.4 Funkcje członkowskie

- **void sleditForSprite(Sprite& s, float x, float y)**
  - **Cel**: Ustawia pozycję przycisku względem pozycji podanego sprite'a s
  - **Parametry**:
    - Sprite& s: Referencja do obiektu klasy SFML Sprite
    - float x: Przesunięcie w osi X względem pozycji sprite'a
    - float y: Przesunięcie w osi Y względem pozycji sprite'a
  - **Operacje**: Ustawia pozycję prostokąta button oraz tekstu txt na podstawie pozycji sprite'a oraz podanych przesunięć
- **void draw(RenderWindow& window)**
  - **Cel**: Rysuje przycisk i tekst w podanym oknie renderowania
  - **Parametry**: RenderWindow& window: Referencja do obiektu klasy SFML RenderWindow
  - **Operacje**:
    - Ustawia pozycję tekstu txt na pozycję prostokąta button
    - Rysuje prostokąt button oraz tekst txt w podanym oknie renderowania
- **bool pressed(Event& event, Vector2i mousePosition)**
  - **Cel**: Sprawdza, czy przycisk został naciśnięty
  - **Parametry**:
    - Event& event: Referencja do obiektu klasy SFML Event
    - Vector2i mousePosition: Pozycja myszy
  - **Operacje**:
    - Sprawdza, czy przycisk został naciśnięty lewym przyciskiem myszy i aktualizuje flagę press
    - Zwraca true, jeśli przycisk został naciśnięty, w przeciwnym razie false
- **bool navediaMouse(Event& event, Vector2i mousePosition)**
  - **Cel**: Sprawdza, czy kursor myszy znajduje się nad przyciskiem
  - **Parametry**:
    - Event& event: Referencja do obiektu klasy SFML Event
    - Vector2i mousePosition: Pozycja myszy

  - **Operacje**: Zwraca true, jeśli kursor myszy znajduje się nad przyciskiem, w przeciwnym razie false

- **void getSound(string failAudio)**
  - **Cel**: Ładuje dźwięk z pliku
  - **Parametry**: string failAudio: Ścieżka do pliku dźwiękowego
  - **Operacje**:
    * Ładuje dźwięk z pliku failAudio do bufora buffer
    * Ustawia bufor buffer dla obiektu sound
- **void soundPlay()**
  - **Cel**: Odtwarza dźwięk
  - **Operacje**: Odtwarza dźwięk za pomocą obiektu sound
- **void soundSetVolume(int volume)**
  - **Cel**: Ustawia głośność dźwięku
  - **Parametry**: int volume: Poziom głośności
  - **Operacje**: Ustawia głośność dźwięku sound na wartość volume
- **void setButtonSize(float W, float H)**
  - **Cel**: Ustawia rozmiar przycisku
  - **Parametry**:
    * float W: Nowa szerokość przycisku
    * float H: Nowa wysokość przycisku
  - **Operacje**
    * Ustawia szerokość w i wysokość h
    * Ustawia rozmiar prostokąta button
- **void setOringCenter()**
  - **Cel**: Ustawia środek prostokąta button jako jego punkt oryginalny
  - **Operacje**: Ustawia punkt oryginalny prostokąta button na (w / 2, h / 2)
- **void setPosition(float x, float y)**
  - **Cel**: Ustawia pozycję przycisku
  - **Parametry**:
    * float x: Współrzędna X pozycji przycisku
    * float y: Współrzędna Y pozycji przycisku
  - **Operacje**: Ustawia pozycję prostokąta button na (x, y)
- **void setFillRacktengelColor(float R, float G, float B)**
  - **Cel**: Ustawia kolor wypełnienia prostokąta button
  - **Parametry**:
    * float R: Składowa czerwona koloru (0-255)
    * float G: Składowa zielona koloru (0-255)
    * float B: Składowa niebieska koloru (0-255)
  - **Operacje**: Ustawia kolor wypełnienia prostokąta button przy użyciu wartości RGB.

1.2.5 Wnioski

Klasa Button rozszerza funkcjonalność klasy MyText o obsługę graficznych przycisków. Umożliwia tworzenie przycisków o określonych rozmiarach, pozycjach i kolorach, które mogą reagować na kliknięcia myszy. Dodatkowo, klasa ta obsługuje odtwarzanie dźwięków, co pozwala na dodanie efektów dźwiękowych do interakcji z przyciskiem.

1.3 Plik *Main*

```
#include "MyText.h"
#include "Button.h"
using namespace sf;

const int MAXN = 17;

void clearAreaAround(int x, int y);
void openCell(int x, int y);
void dealWithEvent(RenderWindow& app, Event& e, int x, int y, Vector2i
    mousePosition);
void showDisplay(RenderWindow& app, Sprite& s, int w, int x, int y);
int numberOfFlagsAround(int x, int y);
void calculateNumberOfBombsAroundTheField();
void initializeGameField();
void win();
void initializeButtons();
void calculateBombProbabilities();
int calculateNumberAroundCell(int arr[MAXN][MAXN], int num, int i, int j);
void showProbabilities(RenderWindow& app);


int grid[MAXN][MAXN]{};       //real things in field - is mine or no in cell
int sgrid[MAXN][MAXN]{};      //the field which player see while playing
int used[MAXN][MAXN]{};     //
int probabilities[MAXN][MAXN];
int width{ 10 }, height{ 10 };
int numberOfFlags{ 0 };
bool dead{ false };
bool won{ false };
int showProbs{ 0 };
bool isFirstMoveMade{ false };
float gameTime{ 0.0 };
int numberOfEmptyCells{};
int flag{ 0 };
int bombChance{ 10 };

Button replayButton(105, 40, "Replay");
Button aiProcentButton(50, 40, " AI");

int main()
{
    srand(time(0));

    RenderWindow app(VideoMode(400, 600), "Minesweeper!");
    int w = 32; // Width of each tile in the grid


    Texture t;
    t.loadFromFile("images/tiles.jpg");

    Texture backgroundTexture;
```

```
51      backgroundTexture.loadFromFile("images/background.jpg");
52      sf::Sprite background;
53      background.setTexture(backgroundTexture);
54
55      Sprite s(t);
56      initializeButtons();
57      initializeGameField();
58      calculateNumberOfBombsAroundTheField();
59      Clock clock;
60
61
62
63      while (app.isOpen())
64      {
65          Vector2i mousePosition = Mouse::getPosition(app);
66          int x = std::max(mousePosition.x / w, 0);
67          int y = std::max((mousePosition.y - 50) / w, 0);
68
69
70
71          if (!isFirstMoveMade)
72          {
73              gameTime = 0;
74          }
75          else
76          {
77              if (flag == 0)
78              {
79                  clock.restart();
80                  flag = 1;
81              }
82              if (won == true || dead == true)gameTime = gameTime;
83              else if (flag == 1)
84                  gameTime = clock.getElapsedTime().asSeconds();
85
86          }
87          Event e;
88          while (app.pollEvent(e))
89          {
90              dealWithEvent(app, e, x, y, mousePosition);
91          }
92
93          app.clear(Color::White);
94          app.draw(background);
95          showDisplay(app, s, w, x, y);
96
97
98      }
99
100     return 0;
101 }
102
103
104 void clearAreaAround(int x, int y)
105 {
106     openCell(x, y + 1);
107     openCell(x + 1, y + 1);
108     openCell(x + 1, y);
109     openCell(x + 1, y - 1);
110     openCell(x, y - 1);
111     openCell(x - 1, y - 1);
112     openCell(x - 1, y);
```

```
113     openCell(x - 1, y + 1);
114 }
115
116 void openCell(int x, int y)
117 {
118     if (x<1 || x > width || y < 1 || y > height || used[x][y] != 0 || sgrid[x][y]
            == 11) return;
119     numberOfEmptyCells--;
120     sgrid[x][y] = grid[x][y];
121     used[x][y] = 1;
122     if (grid[x][y] == 9)dead = true;
123     if (grid[x][y] == 0)clearAreaAround(x, y);
124 }
125
126 void showDisplay(RenderWindow& app, Sprite& s, int w, int x, int y)
127 {
128     MyText NumberOfBombsText(""); //NumberOfBombsText
129     NumberOfBombsText.setPosition(45, 10);
130     NumberOfBombsText.setFillTextColor(120, 6, 19);
131     NumberOfBombsText.setCharacterSize(45);
132
133     MyText TimeText(""); //time timer
134     TimeText.setPosition(300, 10);
135     TimeText.setFillTextColor(120, 6, 19);
136     TimeText.setCharacterSize(45);
137
138     MyText WinText("You won!!!"); //time timer
139     WinText.setPosition(10, 450);
140     WinText.setFillTextColor(120, 6, 19);
141     WinText.setCharacterSize(80);
142
143     MyText LoseText("You lose!!!"); //time timer
144     LoseText.setPosition(10, 450);
145     LoseText.setFillTextColor(120, 6, 19);
146     LoseText.setCharacterSize(80);
147
148
149
150     for (int i = 1; i <= 10; i++)
151         for (int j = 1; j <= 10; j++)
152         {
153             if (sgrid[x][y] == 9 || dead == true)
154             {
155                 // std::cout << "BOOM!! " << x << ' ' << y << '\n';
156                 sgrid[i][j] = grid[i][j];
157             }
158             s.setTextureRect(IntRect(sgrid[i][j] * w, 0, w, w));
159             s.setPosition(i * w, 50 + j * w);
160             app.draw(s);
161         }
162
163     NumberOfBombsText.getChislo(numberOfFlags);
164     NumberOfBombsText.draw(app);
165
166     TimeText.getChislo(int(gameTime));
167     TimeText.draw(app);
168     if (won)
169     {
170         WinText.draw(app);
171     }
172     if (dead)
173     {
```

```
174         LoseText.draw(app);
175     }
176     if (showProbs)
177     {
178         //calculateBombProbabilities();
179         showProbabilities(app);
180     }
181 
182     replayButton.draw(app);
183     aiProcentButton.draw(app);
184     app.display();
185 }
186 
187 void dealWithEvent(RenderWindow& app, Event& e, int x, int y, Vector2i
        mousePosition)
188 {
189     //std::cout << endl << x << ' ' << y << endl;
190     if (e.type == Event::Closed)
191         app.close();
192     if (won == false && dead == false) {
193         if (e.type == Event::MouseButtonPressed) {
194             calculateBombProbabilities();
195             if (e.key.code == Mouse::Left)
196             {
197                 std::cout << sgrid[x][y] << ' ' << grid[x][y] << " left\n";
198                 if (sgrid[x][y] >= 1 && sgrid[x][y] <= 8 && numberOfFlagsAround(x,
                        y) == sgrid[x][y])
199                 {
200                     clearAreaAround(x, y);
201                 }
202                 if (!isFirstMoveMade && x > 0 && x <= width && y > 0 && y <=
                        height)
203                 {
204                     std::cout << "FMNM\n";
205                     if (grid[x][y] == 9)
206                     {
207                         grid[x][y] = 0;
208                         numberOfFlags--;
209                         numberOfEmptyCells++;
210                         calculateNumberOfBombsAroundTheField();
211                     }
212 
213                 }
214                 isFirstMoveMade = true;
215 
216                 openCell(x, y);
217                 std::cout << numberOfEmptyCells << "\n";
218                 if (numberOfEmptyCells == 0) win();
219             }
220             else if (e.key.code == Mouse::Right)
221             {
222                 std::cout << "right\n";
223                 if (sgrid[x][y] == 10)
224                 {
225                     sgrid[x][y] = 11;
226                     numberOfFlags--;
227                 }
228                 else if (sgrid[x][y] == 11)
229                 {
230                     sgrid[x][y] = 10;
231                     numberOfFlags++;
232                 }
```

```
233                 // std::cout << numberOfFlags << '\n';
234             }
235             calculateBombProbabilities();
236         }
237     }
238 
239     if (replayButton.navediaMouse(e, mousePosition))
240     {
241         replayButton.setFillRacktengelColor(255, 0, 0);
242     }
243     else
244     {
245         replayButton.setFillRacktengelColor(140, 140, 137);
246     }
247     if (replayButton.pressed(e, mousePosition))
248     {
249         cout << "PRESSED" << endl;
250         std::cout << mousePosition.x << ' ' << mousePosition.y << '/n';
251         std::cout << won << ' ' << dead << '/n';
252         initializeGameField();
253         calculateNumberOfBombsAroundTheField();
254         won = false;
255         dead = false;
256         isFirstMoveMade = false;
257         flag = 0;
258         gameTime = 0.0;
259         calculateBombProbabilities();
260     }
261 
262     if (aiProcentButton.navediaMouse(e, mousePosition))
263     {
264         aiProcentButton.setFillRacktengelColor(255, 0, 0);
265     }
266     else
267     {   if(!showProbs)
268             aiProcentButton.setFillRacktengelColor(105, 34, 29);
269         else
270             aiProcentButton.setFillRacktengelColor(16, 143, 20);
271     }
272     if (aiProcentButton.pressed(e, mousePosition))
273     {
274         cout << "PRESSED" << endl;
275         calculateBombProbabilities();
276         showProbs = 1 - showProbs;
277         for (int r = 1; r <= width; r++) {
278             for (int c = 1; c <= height; c++) {
279                 cout << probabilities[r][c] << " ";
280             }
281             cout << endl;
282         }
283 
284     }
285 
286 
287 }
288 
289 int numberOfFlagsAround(int x, int y)
290 {
291     int k = 11;
292     return sgrid[x][y + 1] / 11 +
293         sgrid[x + 1][y + 1] / 11 +
294         sgrid[x + 1][y] / 11 +
```

```
295         sgrid[x + 1][y - 1] / 11 +
296         sgrid[x][y - 1] / 11 +
297         sgrid[x - 1][y - 1] / 11 +
298         sgrid[x - 1][y] / 11 +
299         sgrid[x - 1][y + 1] / 11;
300 }
301
302
303
304
305 void initializeGameField()
306 {
307     numberOfFlags = 0;
308     for (int i = 1; i <= width; i++)
309         for (int j = 1; j <= height; j++)
310         {
311             used[i][j] = 0;
312             sgrid[i][j] = 10;
313             if (rand() % bombChance == 0) {
314                 grid[i][j] = 9;
315                 numberOfFlags++;
316             }
317
318             else grid[i][j] = 0;
319         }
320     numberOfEmptyCells = width * height - numberOfFlags;
321 }
322 void calculateNumberOfBombsAroundTheField()
323 {
324     for (int i = 1; i <= width; i++)
325         for (int j = 1; j <= height; j++)
326         {
327             int n = 0;
328             if (grid[i][j] == 9) continue;
329             if (grid[i + 1][j] == 9) n++;
330             if (grid[i][j + 1] == 9) n++;
331             if (grid[i - 1][j] == 9) n++;
332             if (grid[i][j - 1] == 9) n++;
333             if (grid[i + 1][j + 1] == 9) n++;
334             if (grid[i - 1][j - 1] == 9) n++;
335             if (grid[i - 1][j + 1] == 9) n++;
336             if (grid[i + 1][j - 1] == 9) n++;
337             grid[i][j] = n;
338         }
339 }
340
341 void win()
342 {
343     won = true;
344     for (int i = 1; i <= width; i++)
345         for (int j = 1; j <= height; j++)
346         {
347             if (sgrid[i][j] == 10)sgrid[i][j] = 11;
348
349         }
350     numberOfFlags = 0;
351
352 }
353
354 void initializeButtons()
355 {
356     replayButton.setFillRacktengelColor(140, 140, 137);
```

```
357     replayButton.setFillTextColor(0, 255, 90);
358     replayButton.setPosition(140, 15);
359     replayButton.setFillTextColor(255, 205, 5);
360     replayButton.setCharacterSize(30);
361 
362     aiProcentButton.setFillRacktengelColor(105, 34, 29);
363     aiProcentButton.setFillTextColor(0, 255, 90);
364     aiProcentButton.setPosition(40, 420);
365     aiProcentButton.setFillTextColor(255, 205, 5);
366     aiProcentButton.setCharacterSize(30);
367 
368 }
369 
370 
371 void calculateBombProbabilities() {
372     for (int r = 1; r <= width; r++) {
373         for (int c = 1; c <= height; c++) {
374             probabilities[r][c] = 0.0;
375         }
376     }
377     int numOfEmptyCells{};
378     int numOfFlags{};
379     int chance{};
380     for (int i = 1; i <= width; i++)
381         for (int j = 1; j <= height; j++)
382         {
383             if (sgrid[i][j] >= 1 && sgrid[i][j] <= 8)
384             {
385                 //// numOfEmptyCells = calculateNumberAroundCell(sgrid, 10, i, j);
386                 // numOfFlags = calculateNumberAroundCell(sgrid, 11, i, j);
387                 numOfEmptyCells = 0;
388                 numOfFlags = 0;
389 
390 
391                 if (sgrid[i + 1][j] == 10) numOfEmptyCells++;
392                 if (sgrid[i][j + 1] == 10) numOfEmptyCells++;
393                 if (sgrid[i - 1][j] == 10) numOfEmptyCells++;
394                 if (sgrid[i][j - 1] == 10) numOfEmptyCells++;
395                 if (sgrid[i + 1][j + 1] == 10) numOfEmptyCells++;
396                 if (sgrid[i - 1][j - 1] == 10) numOfEmptyCells++;
397                 if (sgrid[i - 1][j + 1] == 10) numOfEmptyCells++;
398                 if (sgrid[i + 1][j - 1] == 10) numOfEmptyCells++;
399 
400                 numOfFlags = numberOfFlagsAround(i, j);
401 
402                 if (numOfEmptyCells > 0)
403                 {
404                     chance = ((sgrid[i][j] - numOfFlags) * 100) / numOfEmptyCells;
405                     if (chance == 0 && numOfEmptyCells > 0)chance = 111;
406                     //std::cout << chance << ' ' << numOfEmptyCells << ' ' <<
                            numOfFlags << ' ' << sgrid[i][j] << ' ' << i << 'a  ' << j
                             << '\n';
407                     if (sgrid[i + 1][j] == 10) probabilities[i + 1][j] = max(
                            probabilities[i + 1][j], chance);
408                     if (sgrid[i][j + 1] == 10) probabilities[i][j + 1] = max(
                            probabilities[i][j + 1], chance);
409                     if (sgrid[i - 1][j] == 10) probabilities[i - 1][j] = max(
                            probabilities[i - 1][j], chance);
410                     if (sgrid[i][j - 1] == 10) probabilities[i][j - 1] = max(
                            probabilities[i][j - 1], chance);
411                     if (sgrid[i + 1][j + 1] == 10) probabilities[i + 1][j + 1] =
                            max(probabilities[i + 1][j + 1], chance);
```

```
412                     if (sgrid[i - 1][j - 1] == 10) probabilities[i - 1][j - 1] =
                            max(probabilities[i - 1][j - 1], chance);
413                     if (sgrid[i - 1][j + 1] == 10) probabilities[i - 1][j + 1] =
                            max(probabilities[i - 1][j + 1], chance);
414                     if (sgrid[i + 1][j - 1] == 10) probabilities[i + 1][j - 1] =
                            max(probabilities[i + 1][j - 1], chance);
415                 }
416             }
417
418         }
419
420 }
421
422 int calculateNumberAroundCell(int arr[MAXN][MAXN], int num, int i, int j)
423 {
424     int n{ 0 };
425     if (arr[i + 1][j] == num) n++;
426     if (arr[i][j + 1] == num) n++;
427     if (arr[i - 1][j] == num) n++;
428     if (arr[i][j - 1] == num) n++;
429     if (arr[i + 1][j + 1] == num) n++;
430     if (arr[i - 1][j - 1] == num) n++;
431     if (arr[i - 1][j + 1] == num) n++;
432     if (arr[i + 1][j - 1] == num) n++;
433     return n;
434 }
435
436 void showProbabilities(RenderWindow& app)
437 {
438     int w = 32;
439     MyText probText[MAXN][MAXN]; //time timer
440     for (int i = 1; i <= width; i++)
441         for (int j = 1; j <= height; j++)
442         {
443             if (sgrid[i][j] == 10 && probabilities[i][j] != 0)
444             {
445                 probText[i][j].setPosition(5 + i * w, 50 + j * w);
446                 probText[i][j].setString(to_string(probabilities[i][j]));
447                 probText[i][j].setFillTextColor(209, 113, 40);
448                 probText[i][j].setCharacterSize(20);
449                 if (probabilities[i][j] == 100)
450                 {
451                     probText[i][j].setCharacterSize(14);
452                     probText[i][j].setPosition(5 + i * w, 52 + j * w);
453                 }
454                 if (probabilities[i][j] == 111)
455                 {
456                     probText[i][j].setString("0");
457                     probText[i][j].setCharacterSize(20);
458                     probText[i][j].setPosition(10 + i * w, 50 + j * w);
459                 }
460                 probText[i][j].draw(app);
461             }
462         }
463 }
```

1.3.1 Główne zmienne globalne

- const int MAXN = 17: Stała określająca maksymalny rozmiar siatki gry.
- int grid[MAXN][MAXN]: Siatka przechowująca rzeczywisty stan gry (czy w komórce jest

mina).

- int sgrid[MAXN][MAXN]: Siatka widziana przez gracza (to, co gracz widzi na planszy).
- int used[MAXN][MAXN]: Siatka przechowująca informacje o tym, które komórki zostały już odkryte.
- int probabilities[MAXN][MAXN]: Siatka przechowująca obliczone prawdopodobieństwa wystąpienia min.
- int width, height: Szerokość i wysokość planszy (10x10).
- int numberOfFlags: Liczba flag umieszczonych przez gracza.
- bool dead: Flaga oznaczająca, czy gracz zginął.
- bool won: Flaga oznaczająca, czy gracz wygrał.
- int showProbs: Flaga określająca, czy mają być wyświetlane prawdopodobieństwa min.
- bool isFirstMoveMade: Flaga oznaczająca, czy został wykonany pierwszy ruch.
- float gameTime: Czas gry.
- int numberOfEmptyCells: Liczba pustych komórek na planszy.
- int flag: Flaga używana do sterowania zegarem.
- int bombChance: Prawdopodobieństwo wystąpienia miny.

1.3.2 Obiekty przycisków

- **Button replayButton(105, 40, "Replay")**: Przycisk do rozpoczęcia nowej gry
- **Button aiProcentButton(50, 40, "AI")**: Przycisk do włączenia/wyłączenia wyświetlania prawdopodobieństw min.

1.3.3 Funkcje

- clearAreaAround: Odkrywa komórki wokół danej komórki.
- openCell: Odkrywa daną komórkę i, jeśli jest pusta, odkrywa również komórki wokół niej.
- showDisplay: Wyświetla aktualny stan planszy i inne elementy interfejsu (np. licznik bomb, czas gry, komunikaty o wygranej/przegranej).
- dealWithEvent: Obsługuje zdarzenia (kliknięcia myszką, zamknięcie okna itp.).
- numberOfFlagsAround: Liczy liczbę flag wokół danej komórki.
- initializeGameField: Inicjalizuje planszę gry, losując położenie min.
- calculateNumberOfBombsAroundTheField: Oblicza liczbę min wokół każdej komórki na planszy.
- win: Ustawia stan gry na wygrany i oznacza wszystkie pozostałe komórki jako oznaczone flagą.

- initializeButtons: Inicjalizuje przyciski gry (ustawia ich wygląd i pozycję).
- calculateBombProbabilities: Oblicza prawdopodobieństwo wystąpienia min w nieodkrytych komórkach.
- calculateNumberAroundCell: Liczy liczbę komórek o danej wartości wokół określonej komórki.
- showProbabilities: Wyświetla prawdopodobieństwa wystąpienia min na planszy.

1.3.4 Główna funkcja main

1. Inicjalizuje okno gry.
2. Wczytuje tekstury dla kafelków i tła.
3. Inicjalizuje przyciski i planszę gry.
4. Obsługuje główną pętlę gry:
    - Pobiera aktualną pozycję myszy.
    - Jeśli pierwszy ruch nie został wykonany, resetuje licznik czasu.
    - Jeśli gra została zakończona (wygrana lub przegrana), zatrzymuje licznik czasu.
    - Obsługuje zdarzenia (np. kliknięcia myszką).
    - Wyświetla aktualny stan gry

1.3.5 Wnioski

Ten kod stanowi kompletną implementację klasycznej gry w "Saper"z dodatkową funkcjonalnością wyświetlania prawdopodobieństw wystąpienia min oraz przyciskami do restartu gry i włączenia/wyłączenia wyświetlania prawdopodobieństw.