Projekt z programowania

Patryk Marek i Michał Koziński

Spis treści

1 Temat projektu

Tematem projektu jest gra strategiczno-planszowa "Okręty" (ang. Battleship). W wersji oryginalnej stworzonej przez Clifforda Von Vickera na początku XX

wieku przeznaczona była dla dwóch graczy, w naszym projekcie jest to pojedynek z komputerem. Gra realizowana jest w klasycznych zasadach rozgrywki.

2 Wykorzystane technologie

2.1 Język programowania

Projekt napisany został w języku programistycznym C++ w wersji stabilnej C++17 z wykorzystaniem kompilatora GCC.

2.2 Interfejs graficzny

Interfejs graficzny stworzony został za pomocą przenośnej, wieloplatformowej biblioteki wxWidgets w wersji stabilnej 2.8.8.

2.3 Wykorzystane biblioteki

W projekcie wykorzystaliśmy następujące darmowe biblioteki:

- iostream
- map
- cstdlib
- vector

3 Instrukcja użytkowania

Kompletna instrukcja użytkowania programu:

3.1 Uruchomienie programu

Po uruchomieniu programu wyskakuje okienko powitalne (Rysunek 1).
Następnie uruchamia się właściwe okno z grą (Rysunek 2).
Znajdują się na nim:

- Plansza gracza (po lewej stronie),

- Plansza komputera (po prawej stronie),
- Statystyki,
- Pole wyboru z opcją rozpoczynającej strony,
- Przycisk z zasadami gry,
- Przycisk restartujący grę.

Po wciśnięciu przycisku z zasadami pojawi sie okno informacyjne (Rysunek 3).

3.2 Etap rozstawiania

Plansza gracza po kliknięciu na pole (Rysunek 4).

Pełne rozstawienie statków zgodnie z instrukcją (Rysunek 5).

3.3 Etap strzelania

W tym etapie możliwe są dwa statusy po kliknięciu na pole przeciwnika:

- Pudło (Rysunek 6).
- Statek trafiony (Rysunek 7).

Na tym samym etapie ruchy wykonuje także komputer, możliwe są dwa statusy pól:

- Pudło (Rysunek 8).
- Statek trafiony (Rysunek 9).

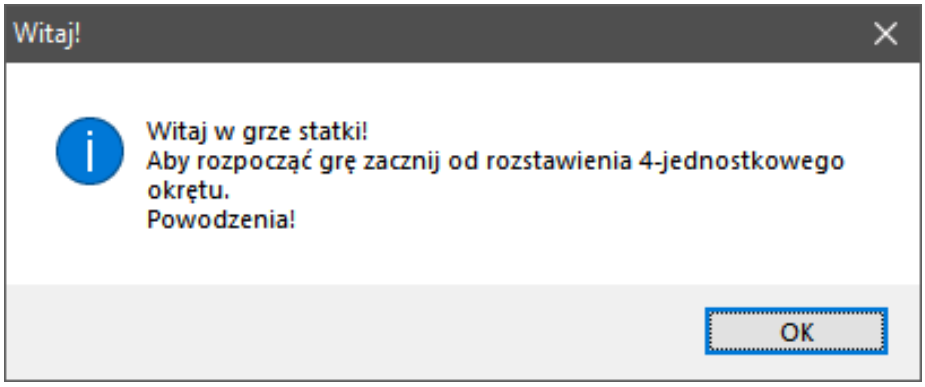
Dodatkowo trafienie statku przez komputer komunikowane jest przez okienko ostrzegawcze (Rysunek 10).

3.4 Koniec gry

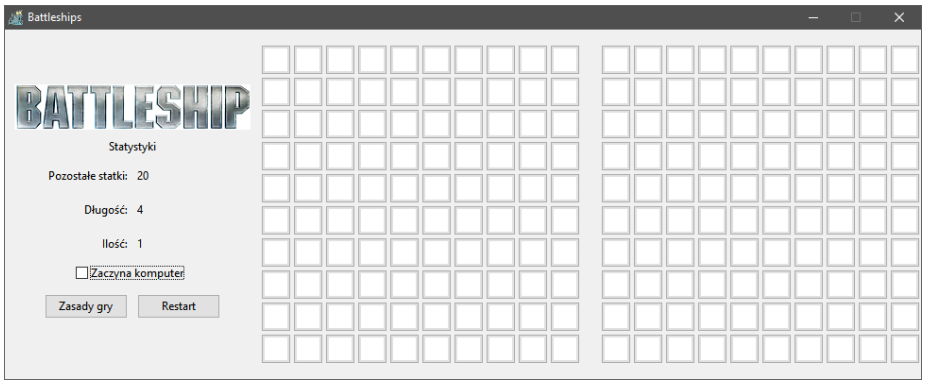
Możliwe są dwa wyniki rozgrywki komunikowane odpowiednimi okienkami:

- Wygrana (Rysunek 11)
- Przegrana (Rysunek 12)

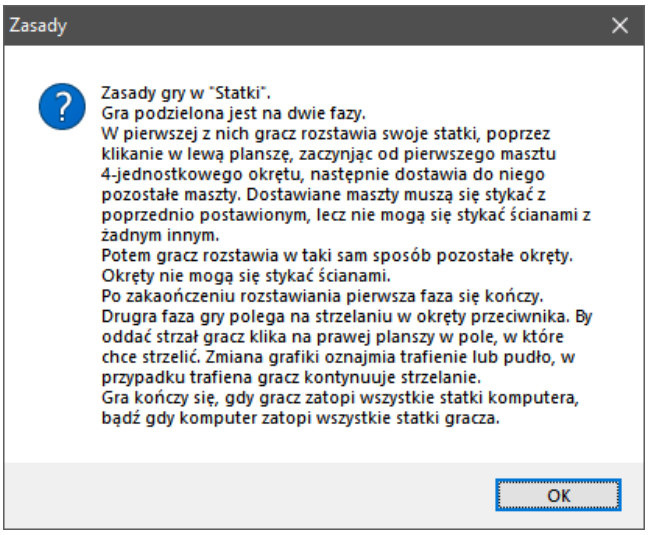
Po zakończeniu rozgrywki można użyć przycisku restart do wyzerowania statystyk i plansz.



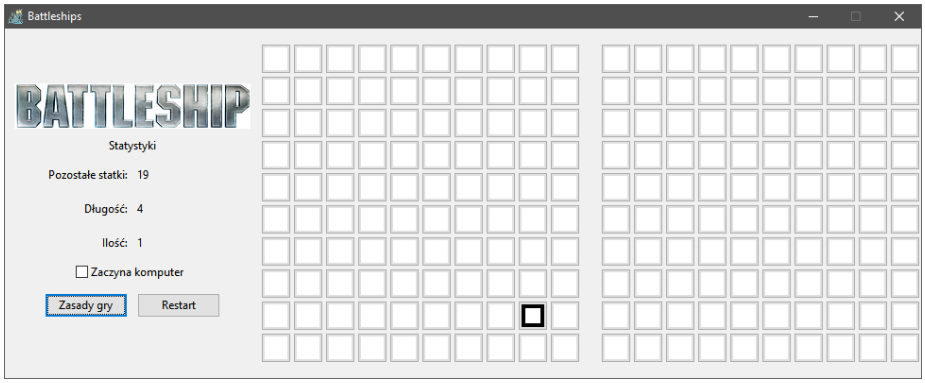
Rysunek 1:



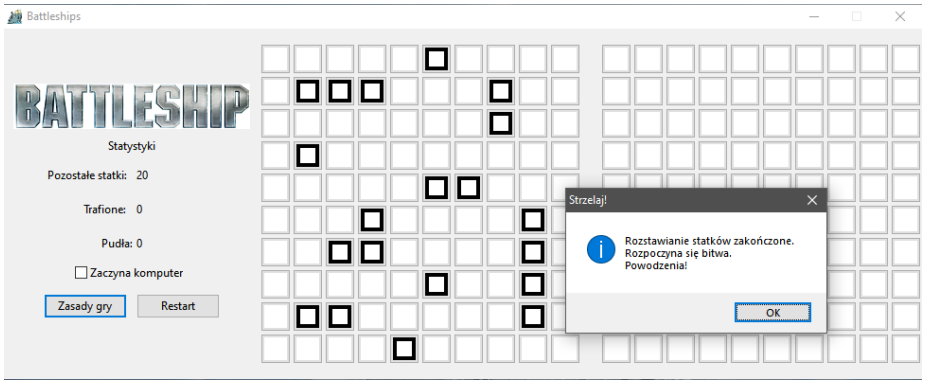
Rysunek 2:



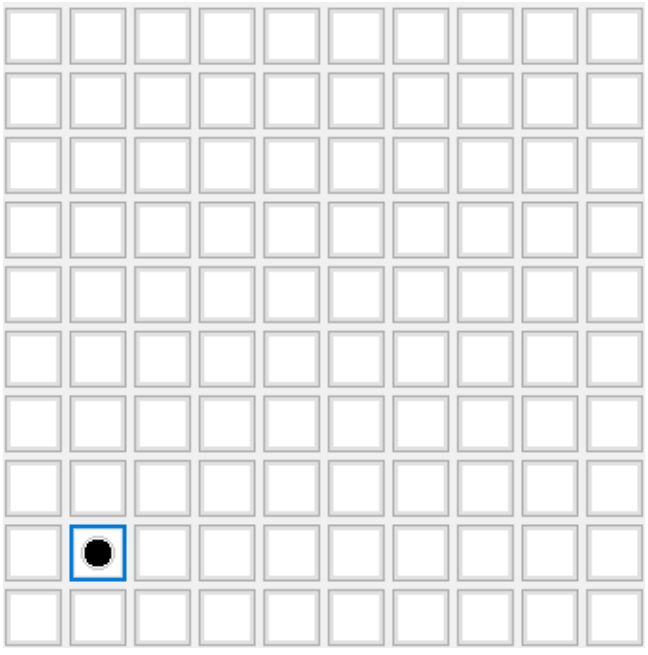
Rysunek 3:



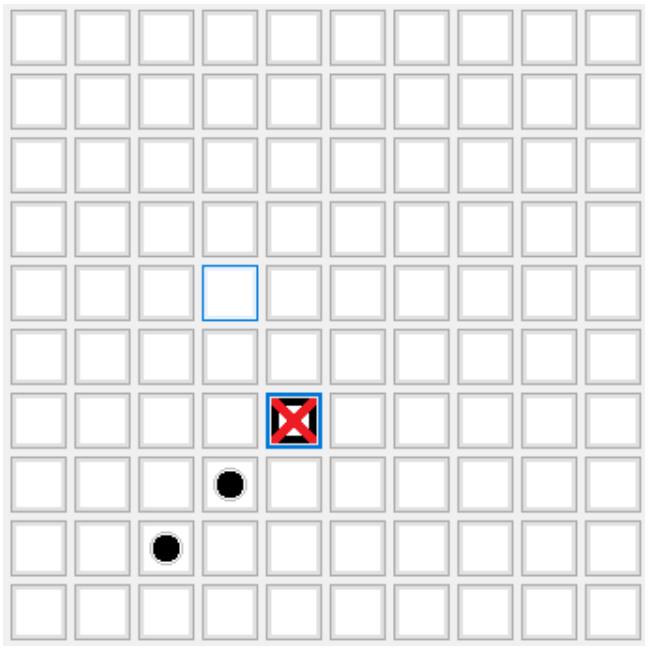
Rysunek 4:



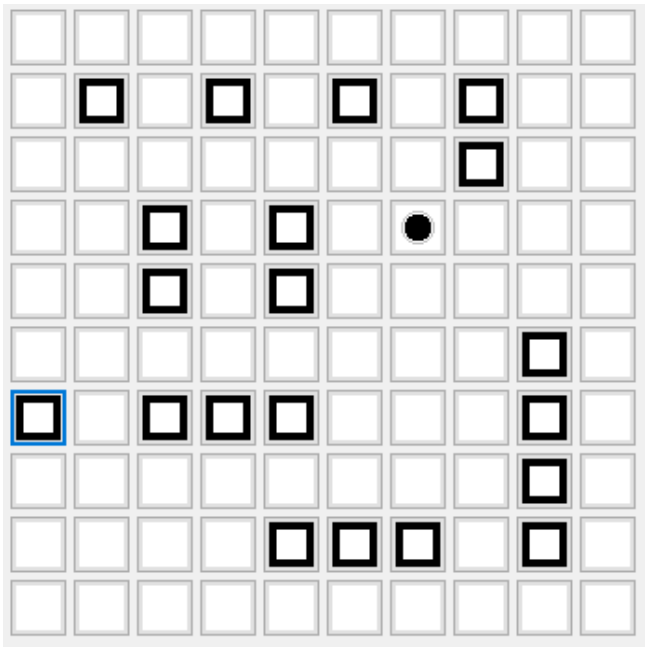
Rysunek 5:



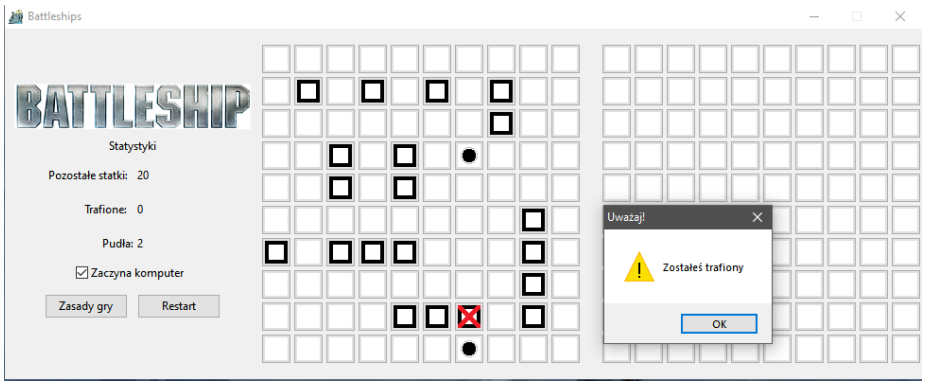
Rysunek 6:



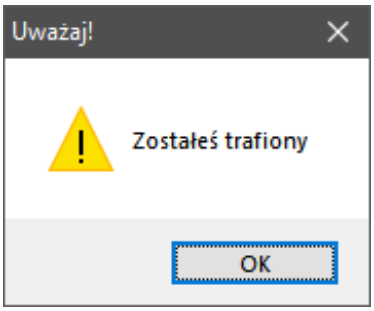
Rysunek 7:



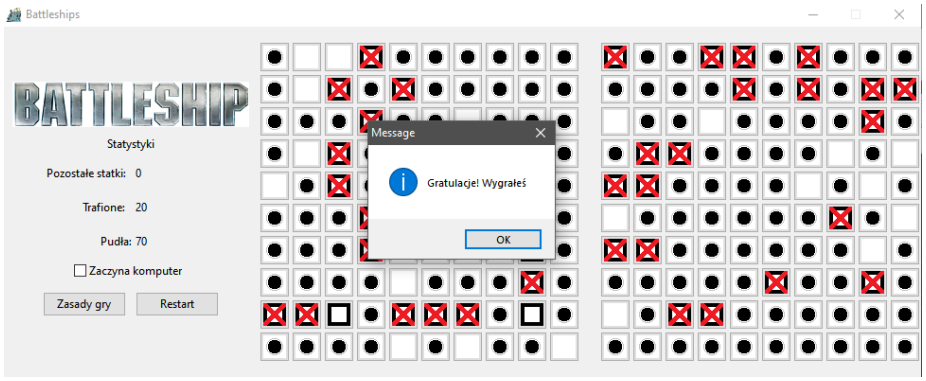
Rysunek 8:



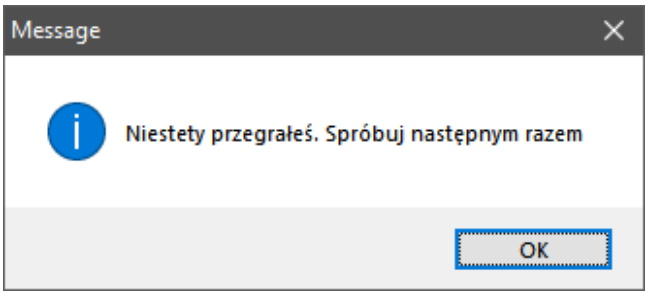
Rysunek 9:



Rysunek 10:



Rysunek 11:



Rysunek 12:

4 Opis klasy i wykorzystanych metod

4.1 Pola w klasie

```
1  private:
2          int l_tab[11][11], p_tab[11][11];
3          /*
4          0 - puste
5          1 - statek
6          2 - trafiony
7          3 - pudlo
8          */
9          string komunikat;
10         pair<int,int> last_cpu;
11         pair<int,int> before;
12         pair<int,int> last;
13         vector<pair<int,int>> statki;
14         int hited,missed,l_rand,ship_left;
15         /*
16         hited == dlugosc
17         missed == ile pozostalo danego typu
18         */
19         bool faza; // 0 - rozstawianie      1 - strzelanie
20         int length,clicks,loop; //length == ilosc statkow
21         int cpu_hited;
22         int lp,lg;
23         int lengthTab[11]={4,3,3,2,2,2,1,1,1,1,0};
```

Najważniejszymi polami w klasie są `l_tab[11][11]` oraz `p_tab[11][11]`. Są to tablice dwuwymiarowe odpowiednio przyporządkowane lewej i prawej planszy. Wartości w tablicach odpowiadają danemu statusowi pola [x][y]:

- 0 - pole puste,
- 1 - statek rozstawiony,
- 2 - statek trafiony,
- 3 - pudło.

Trzy pola: `last_cpu`, `before` i `last` przechowują poprzednie ruchy komputera oraz gracza w celu wykorzystania ich do metod zabezpieczających prawidłowość ruchu.
Pola typu całkowitego `hited`, `missed`, `ship_left` oraz `cpu_hited` służą do prowadzenia statystyk w trakcie gry.
Pole `faza` typu bool'owskiego służy do podziału gry na fazy: wartość 0 oznacza fazę rozstawiania, wartość 1 oznacza fazę strzelania.

4.2 Metody w klasie

```
1  public:
2          game();
3          virtual ~game();
4          bool rozstaw(int,int);
5          void cpu_rozstaw();
6          bool strzal(int,int);
7          pair<int,int> cpu_strzal();
8          pair<int,int> cpu_rozstaw_test();
9          pair<int,int> cpu_dobuduj();
10         pair<int,int> get_cpu_ship_pos();
11         bool rozstaw_test(int,int,bool,bool kto);
12         /*
13         1 - nowy statek, 0 - dostawianie starego
14         1 - gracz, 0 - komputer
15         */
16         bool wygrana();
17         void reset();
18         int random();
19         string get_komunikat();
20         bool get_faza();
21         void set_faza(bool wart)
22         int get_hited()
23         int get_missed()
24         int get_left()
25         int get_l_pole_status(int a, int b)
```

```
26          int get_p_pole_status(int a, int b)
```

Metody w naszej klasie składają się z bezparametrowego konstruktora i destruktora, get'erów, set'erów oraz metod które można podzielić na trzy grupy:

- Metody gracza,
- Metody komputera,
- Metody ogólne (odnoszące się do rozgrywki)

Metody te są odwzorowaniem zasad gry w statki. Dla fazy rozstawiania zastosowanie mają metody: rozstaw, cpu_rozstaw, cpu_rozstaw_test, cpu_dobuduj i rozstaw_test. W przypadku tej fazy gry liczba metod dla komputera (z prefiksem cpu) jest bardziej liczna niż dla gracze ze względu na pesudo-losowy charakter metod komputerowych. Wymagało to wprowadzenia dodatkowej metody i szerego zabezpieczeń przed błędnymi ruchami. Do obsługi drugiej fazy gry służą dwie metody: strzal i cpu_strzal. Tutaj kwestia była znacznie prostsza niż w przypadku fazy rozstawiania ze względu na brak możliwych nielegalnych ruchów. Do metod ogólnych zalicza się cała reszta, ich zadaniem jest obsługa funkcji gry takich jak sprawdzanie aktualnej fazy lub czy gra się zakończyła.

5 Poszczególne algorytmy

```
1  bool game::rozstaw_test(int a, int b, bool pierw, bool czy_gracz)
2  {
3          int i=0;
4          bool l=0; //do sprawdzania, czy jest odpowiednio, do
               ostatniego ustawionego, ustawiony
5          if(czy_gracz==1)
6          {
7                  //stykanie z ostatnim
8                  if(a-1 == last.first && b == last.second)
9                          l = 1;
10                 if(a == last.first && b+1 == last.second)
11                         l = 1;
12                 if(a+1 == last.first && b == last.second)
13                         l = 1;
14                 if(a == last.first && b-1 == last.second)
15                         l = 1;
16                 //stykanie z innym
```

```
17                  if(!(a-1<0) && ((a-1 == last.first && b == last.
                        second) || l_tab[a-1][b]==1))
18                      i++;
19                  if(!(b+1>10) && ((a == last.first && b+1 == last.
                        second) || l_tab[a][b+1]==1))
20                      i++;
21                  if(!(a+1>10) && ((a+1 == last.first && b == last.
                        second) || l_tab[a+1][b]==1))
22                      i++;
23                  if(!(b-1<0) && ((a == last.first && b-1 == last.
                        second) || l_tab[a][b-1]==1))
24                      i++;
25          }
26          else
27          {
28                  if(!(a-1<=0) && p_tab[a-1][b]==1)
29                      i++;
30                  if(!(b+1>10) && p_tab[a][b+1]==1)
31                      i++;
32                  if(!(a+1>10) && p_tab[a+1][b]==1)
33                      i++;
34                  if(!(b-1<=0) && p_tab[a][b-1]==1)
35                      i++;
36          }
37          if(!pierw) //nie jest rozstawiany pierwszy element statku
38                  if(i==1  && (!czy_gracz || l==1))
39                      return 1;
40                  else
41                      return 0;
42          else //rozstawiany jest pierwszy element statku
43                  if(i==0)
44                      return 1;
45                  else
46                      return 0;
```

Z punktu widzenia naszego projektu jest to jedna z bardziej kluczowych metod. Ma ona charakter zabezpieczający i służy ona do rozpoznawania czy elementy stawiane stykają się odpowiednio z poprzednio postawionym elementem oraz czy nie stykają się z żadnym innym statkiem. Metoda ta ma podział na testowanie ruchu gracza i ruchu komputera w zależności od przekazanego parametru typu bool. Wykorzystaliśmy tutaj pomocniczą zmienną i typu całkowitego zwiekszająca swoją wartość o jeden za każdy element statku w pobliżu pola na którym

aktualnie umieszczany jest statek. Żeby metoda zwróciła wartość TRUE pozwalającą wykonać ruch zmienna ta musi mieć wartość 1 w przypadku dostawiania elementów do już istniejącego statku oraz wartość 0 w przypadku stawiania pierwszego elementu. Dodatkowym zabezpieczniem ruchu w tej metodzie jest wykorzystanie pola last zawierającego informacje o poprzednim ruchu. Aby móc dostawić element musi on być w otoczeniu poprzednio postawionego elementu.

```
1  bool game::rozstaw(int a,int b)
2  {
3          if(l_tab[a][b]==1 || length<0)
4                  return 0;
5          if(clicks==lengthTab[loop])
6                  if(rozstaw_test(a,b,1,1))
7                  {
8                          l_tab[a][b]=1;
9                          ship_left--;
10                         last.first = a;
11                         last.second = b;
12                 }
13                 else
14                         return 0;
15         else
16                 if(rozstaw_test(a,b,0,1))
17                 {
18                         l_tab[a][b]=1;
19                         ship_left--;
20                         last.first = a;
21                         last.second = b;
22                 }
23                 else
24                         return 0;
25         clicks--;
26         if(clicks==0)
27         {
28                 length--;
29                 loop++;
30                 clicks=lengthTab[loop];
31                 hited=clicks;
32                 missed=5-clicks;
33         }
34         if(length==0 && ship_left==0)
35         {
```

```
36                  faza=1;
37                  ship_left=20;
38                  missed=0;
39                  hited=0;
40          }
41          return 1;
42  }
```

Metoda rozstawiania elementów przez gracza opiera się na sprawdzaniu i uzuepłnianiu wartości w lewej tablicy oraz na przeprowadzaniu testów z wykorzystaniem poprzednio pokazanej metody (rozstaw_test). Istotną rolę w tej metodzie odgrywa odpowiednie ustawianie parametrów (clicks,length,loop,hited,missed) pozwalające na odpowiednie sterowanie statystykami oraz ogólnym przebiegiem fazy rozstawiania (W ostatnim warunku jeżeli parametr length i ship_left osiągają wartość 0 zmieniana jest faza rozgrywki).

```
1   bool game::strzal(int a, int b)
2   {
3           if(faza)
4           {
5                   if(p_tab[a][b]==1)
6                   {
7                           p_tab[a][b]=2;
8                           hited++;
9                           ship_left--;
10                          return 1;
11                  }//trafiony
12                  else
13                  {
14                          p_tab[a][b]=3;
15                          missed++;
16                          return 0;
17                  }//nietrafiony
18          }
19          return 0;
20  }
```

```
1           pair<int,int> game::cpu_strzal()
2           {
3                   int a,b;
4                   a=random();
5                   b=random();
```

```
6                 while(l_tab[a][b]==2||l_tab[a][b]==3) // losuj a i
                      b tak dlugo az pole nie bedzie wczesniej
                      wylosowane
7                 {
8                         a=random();
9                         b=random();
10                }
11                if(l_tab[a][b]==0) //jezeli trafil w puste
12                l_tab[a][b]=3; //ustaw wartosc na pudlo
13                else if(l_tab[a][b]==1) //jezeli trafil w statek
14                {
15                        l_tab[a][b]=2; //ustaw wartosc na trafiony
16                        cpu_hited++;
17                }
18                return make_pair(a,b);
19
20        }
```

Metody obsługujące fazę strzalania są bardzo podobne dla gracza i komputera (główną różnicą jest to że komputer najpierw losuje koordynaty niestrzelonego wcześniej pola). Główną ideą tych metod jest zmiana wartości w tabelach na "pudło" lub "trafiony" oraz zmiana prametrów służacych do śledzenia statystyk.

```
1  bool game::wygrana()
2  {
3         if(hited==20)
4         {
5                 komunikat="Gratulacje! Wygrales";
6                 return 1;
7         }
8         else if(cpu_hited==20)
9         {
10                komunikat="Niestety przegrales. Sprobuj nastepnym
                      razem";
11                return 1;
12        }
13        else
14        return 0;
15 }
```

Metoda sprawdzająca czy gra dobiegła końca działa na zasadzie sprawdzania odpowiedniej statystyki i zwróceniu komunikatu. Jeżeli parametr hited (stat-

ki trafione przez gracza) równy jest 20 tzn. wszystkie elementy statków wroga zostały zestrzelony metoda zwraca wartość 1 i kończy rozgrywkę. W ten sam sposób działa dla komputera, sprawdza czy cpu_hited równe jest 20. Jeżeli żaden z tych warunków nie jest spełniony metoda zwraca 0 i nie ma żadnego efektu na rozgrywkę.

6 Wyzwania i wyniki testowania programu

Głównym wyzwaniem jakie pojawiło się w trakcie tworzenia programu było stworzenie metod pozwalających komputerowi na losowe rozmieszczanie statków zapewniające niepowtarzalność każdej rozgrywki z jednoczesnym zapewnieniem pełnej funkcjonalności aplikacji. Przez 14 wersji naszego projektu problem z tym zagadnieniem pojawiał się najczęściej w postaci np. wykraczania z ruchami poza tablicę, zaburzona kolejność rozstawiania, zawieszanie aplikacji w pętlach zabezpieczających ruch itp. Jednak największym problemem jaki pojawił się było postawienie statku przez komputer w takim miejscu że nie istniał żaden legalny ruch. Pomimo tego że sytuacja taka mogła zaistnieć niezwykle rzadko wymagało to od nas zdefiniowania części metod na nowo. W razie takiego scenariusza plansza komputera jest zerowana oraz wszystkie statki rozstawiane są od nowa.

7 Podział pracy

- Obsługa interakcji z planszą - Patryk 45%, Michał 55%
- Metody obsługujące ruchy gracza - Patryk 20%, Michał 80%
- Metody obsługujące ruchy komputera - Patryk 80%, Michał 20%
- Zabezpieczenie rozstawiania - Patryk 30%, Michał 70%
- Warstwa graficzna - Patryk 40%, Michał 60%
- Dokumentacja - Patryk 85%, Michał 15%