POLITECHNIKA LUBELSKA

WYDZIAŁ MATEMATYKI I INFORMATYKI TECHNICZNEJ

Kierunek: Inżynieria i Analiza Danych

Przedmiot:
Projekt z zakresu programowania
Dokumentacja do projektu

Autorzy:

Mateusz Rosiński, Jakub Przemski, Igor Rybiński

Praca wykonana pod kierunkiem:

dr Pawła Wlazia

Spis treści

1 Cel projektu

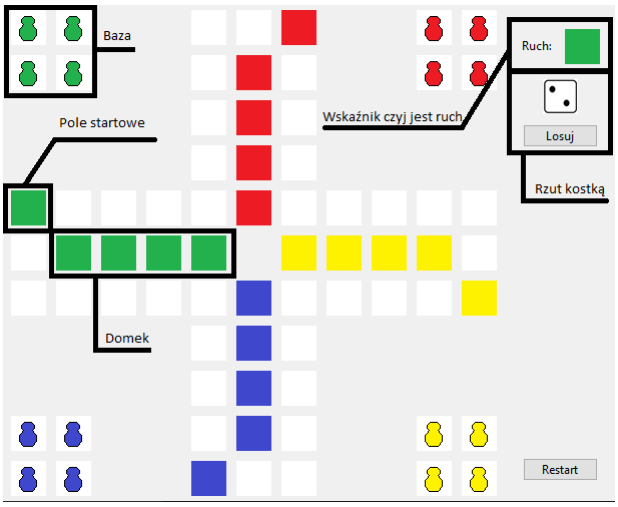
Celem projektu jest stworzenie kompleksowej implementacji popularnej gry planszowej "Chińczyk" w języku C++ z wykorzystaniem biblioteki `WxWidgets` do budowy wieloplatformowego interfejsu graficznego. Projekt ma na celu nie tylko dostarczenie funkcjonalnej gry, ale również zaprezentowanie umiejętności programowania w C++ oraz korzystania z biblioteki WxWidgets. Projekt ma stanowić przykład praktyczny, prezentujący efektywne wykorzystanie zaawansowanych struktur programistycznych w kontekście gier planszowych.

2 Opis gry

2.1 Zasady gry

- Do gry potrzebnych jest 4 graczy, którym przed rozpoczęciem rozgrywki przypisujemy jeden z czterech kolorów - zielony, czerwony, żółty, niebieski.
- Z bazy wyjąć pionka można tylko i wyłącznie po wylosowaniu w rzucie kostką liczby 6.
- Każde wylosowanie liczby 6 wiąże się również z ponownym rzutem kostką.
- Poruszać mogą się tylko pionki wystawione z bazy na pole planszy o kolorze danego gracza.
- Pionki mogą nad sobą przeskakiwać (niezależnie od koloru).
- Jeśli pionek stanie na polu zajmowanym przez pionek innego koloru, to stojący poprzednio na tym polu pionek zostaje zbity - wraca do swojej bazy.
- Gdy gracz spróbuje wykonać nielegalny ruch, to traci turę.
- Gra polega na obejściu całej planszy i wstawieniu wszystkich pionków do domku odpowiedniego koloru.
- Na jednym polu w domku może stać więcej niż jeden pionek
- Wygrywa gracz, który jako pierwszy osiągnie wyżej wymieniony cel.
- Gra toczy się do momentu zdobycia 2. i 3. miejsca przez któregoś z graczy. Po spełnieniu tego warunku gra jest zakończona.

Dokładne objaśnienie pól znajduje się na następującym rysunku.



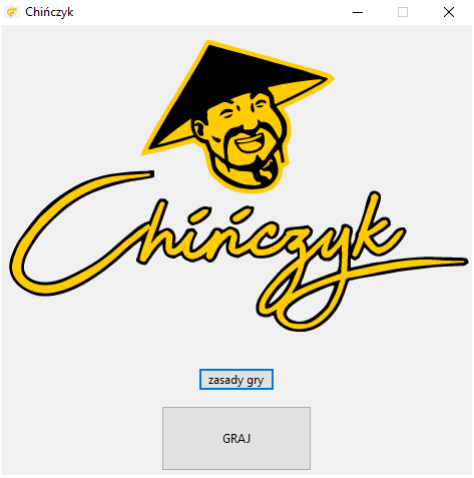
Rysunek 1: Objaśnienie głównych terminów i pól

2.2 Rozgrywka

2.2.1 Rozpoczęcie gry

Po uruchomieniu programu zostaje otworzone okno z dwoma przypiskami:

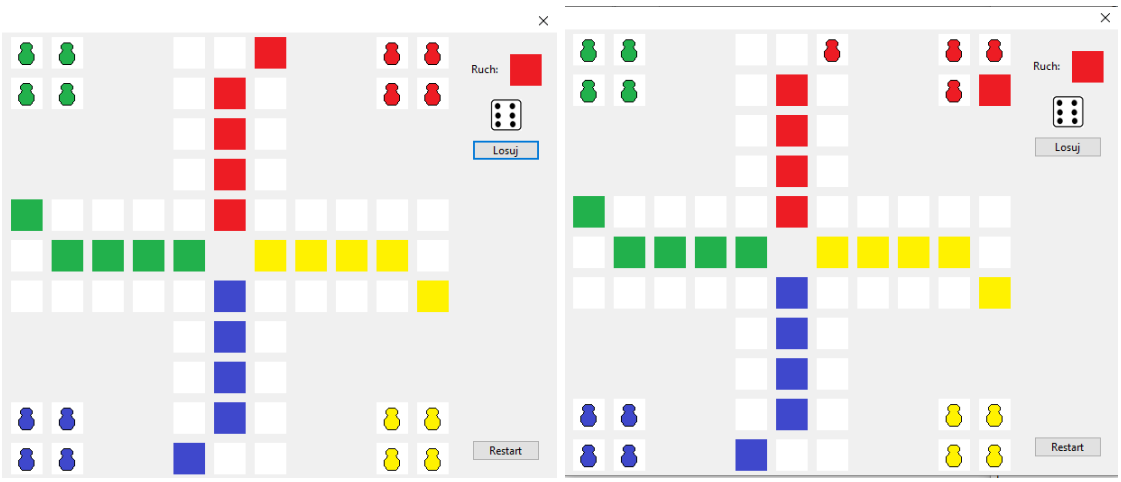
- ”Zasady gry” - jak wskazuje nazwa przycisk służący do wyświetlenia zasad gdy, opisanych w poprzednim punkcie
- ”GRAJ” - przycisk służący do otworzenia głównego okna z grą.



Rysunek 2: Okno po uruchomieniu programu

Aby wyjść pionkiem z bazy należy wylosować kostką liczbę 6. Gracze rzucają kostką do momentu spełnienia tego warunku. Gdy warunek zostanie spełniony gracz, który wylosował liczbę 6 może wystawić jednego pionka z bazy na swoje pole startowe.

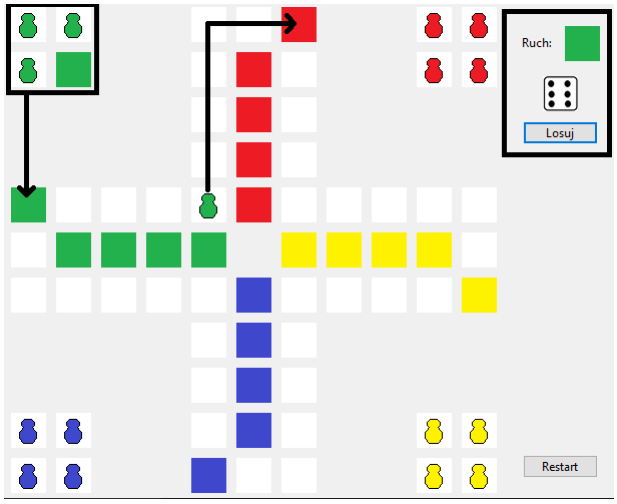
Poniżej przykład dla gracza o kolorze czerwonym.



Rysunek 3: Proces akcji wystawiania pionka na pole startowe

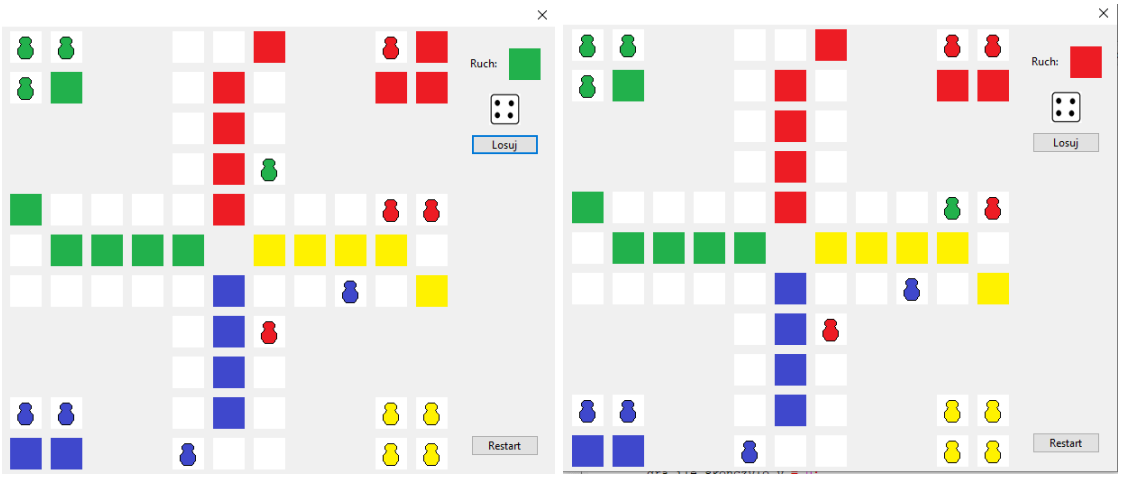
2.2.2 Głowna faza rozgrywki

Gracze przesuwają jeden z pionków na planszy o taką liczbę pól, jaką wylosują w rzucie kostką (1-6).
Jeżeli któryś z graczy wyrzuci liczbę 6, ma prawo do wyboru pomiędzy ustawieniem kolejnego ze swoich pionków na polu startowym a ruchem pionkiem znajdującym się już na planszy. Ponadto, gracz, który wylosował liczbę 6 rzuca kostką jeszcze raz (do momentu, aż wylosuje coś innego).



Rysunek 4: Możliwości akcji gracza zielonego po wylosowaniu liczby 6

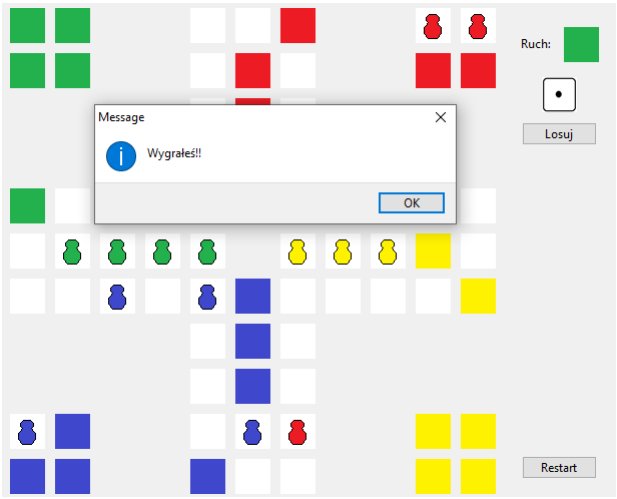
Jeśli podczas gry pionek jednego gracza stanie na polu zajmowanym przez drugiego, pionek stojący na tym polu poprzednio zostaje zbity i wraca do swojej bazy.



Rysunek 5: Proces akcji zbijania (zielony pionek zbija czerwonego)

2.2.3 Koniec gry

Po wprowadzeniu wszystkich pionków danego gracza do domku zostaje wyświetlony komunikat o zakończeniu rozgrywki przez tego gracza, wraz z zajętym miejscem (Wygrana oznacza pierwsze miejsce).



Rysunek 6: Przykład wygranej (gracza zielonego)

2.3 Deklaracje metod użytych w programie

2.3.1 Stworzenie planszy i funkcjonalnych pól

Plansza została utworzona za pomocą wxBitmap, ze względu na niezbędny indywidualizm każdego z przycisków pola do gry. Aby mieć pewność, że funkcjonalne zostaną tylko pola w kształcie krzyża oraz cztery pola na rogach (jako bazy dla pionków graczy) zostały nałożone ograniczenia zadeklarowane w następujący sposób

```
pola[0]= BitmapButton1;
pola[0]->SetBitmap(pionki[0]);
Connect(pola[0]->GetId(),wxEVT_COMMAND_BUTTON_CLICKED,
   (wxObjectEventFunction)&GameDialog::OnBitmapButton1Click);


for(int i=1; i<121;i++){

   // Przypisanie szarego tla kazdemu z pol
   pola[i] = new wxBitmapButton(this, wxNewId(), rysunki[4], wxDefaultPosition, wxDefaultSize,
       wxBU_AUTODRAW|wxBORDER_SIMPLE|wxBORDER_NONE, wxDefaultValidator, _T("ID_BITMAPBUTTON1"));
   FlexGridSizer2->Add(pola[i], 1, wxALL|wxALIGN_CENTER_HORIZONTAL|wxALIGN_CENTER_VERTICAL, 5);


   // Okreslenie funkcjonalnych pol
   if (krzyz.find(i) != krzyz.end()){
      pola[i]->SetBitmap(rysunki[5]);
      Connect(pola[i]->GetId(),wxEVT_COMMAND_BUTTON_CLICKED,
         (wxObjectEventFunction)&GameDialog::OnBitmapButton1Click);
   }
   if(base_g.find(i) != base_g.end()){
      pola[i]->SetBitmap(rysunki[0]);
   }
   if(base_r.find(i) != base_r.end()){
      pola[i]->SetBitmap(rysunki[1]);
   }
   if(base_y.find(i) != base_y.end()){
```

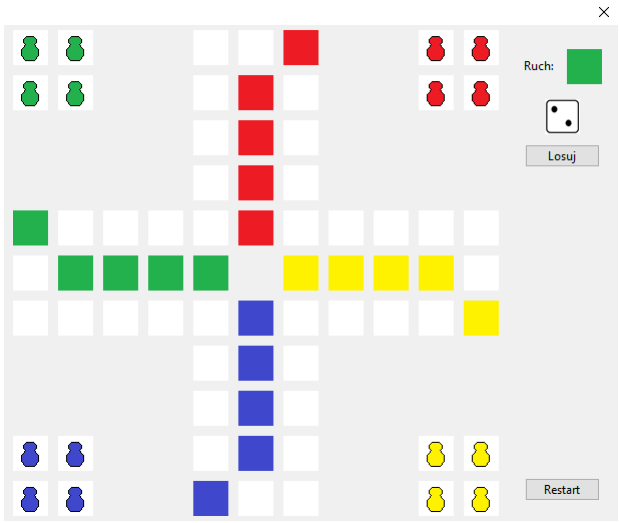
```
      pola[i]->SetBitmap(rysunki[2]);
   }
   if(base_b.find(i) != base_b.end()){
      pola[i]->SetBitmap(rysunki[3]);
   }


   // Okreslenie baz poszczegolnych graczy
   if(square_g.find(i) != square_g.end()){
      pola[i]->SetBitmap(pionki[0]);
      Connect(pola[i]->GetId(),wxEVT_COMMAND_BUTTON_CLICKED,
         (wxObjectEventFunction)&GameDialog::OnBitmapButton1Click);
   }
   if(square_r.find(i) != square_r.end()){
      pola[i]->SetBitmap(pionki[1]);
      Connect(pola[i]->GetId(),wxEVT_COMMAND_BUTTON_CLICKED,
         (wxObjectEventFunction)&GameDialog::OnBitmapButton1Click);
   }
   if(square_y.find(i) != square_y.end()){
      pola[i]->SetBitmap(pionki[2]);
      Connect(pola[i]->GetId(),wxEVT_COMMAND_BUTTON_CLICKED,
         (wxObjectEventFunction)&GameDialog::OnBitmapButton1Click);
   }
   if(square_b.find(i) != square_b.end()){
      pola[i]->SetBitmap(pionki[3]);
      Connect(pola[i]->GetId(),wxEVT_COMMAND_BUTTON_CLICKED,
         (wxObjectEventFunction)&GameDialog::OnBitmapButton1Click);
   }

}

// utworzenie numerow id dla wszystkich pol
id2nr[pola[0]->GetId()]=0;
for(int i=1; i<121;i++){
   id2nr[pola[i]->GetId()]=i;
}

}
```

Funkcjonalne pola zostały zadeklarowane w formie numerów id.

```
std::map<int,int> id2nr; // zamienia id pola na nr od 0 do 121

wxBitmapButton* pola[121];

wxBitmap rysunki[6];

wxBitmap pionki[4];
wxBitmap pionki2[4];
wxBitmap pionki3[4];
wxBitmap pionki4[4];

wxBitmap kostka[6];

int home_g[4] = {0,1,11,12};
int home_r[4] = {9,10,20,21};
int home_b[4] = {99,100,110,111};
int home_y[4] = {108,109,119,120};

// Deklaracja numerow ID pol w bazach poszczegolnych kolorow
std::set<int> square_g = {0,1,11,12};
```

```
std::set<int> square_r = {9,10,20,21};
std::set<int> square_b = {99,100,110,111};
std::set<int> square_y = {108,109,119,120};

// Deklaracja numerow ID, gdzie pionki sie aktualnie znajduja
std::multiset<int> curr_g = {0,1,11,12};
std::multiset<int> curr_r = {9,10,20,21};
std::multiset<int> curr_y = {108,109,119,120};
std::multiset<int> curr_b = {99,100,110,111};

int current_g[4] = {0,1,11,12};
int current_r[4] = {9,10,20,21};
int current_y[4] = {108,109,119,120};
int current_b[4] = {99,100,110,111};

// Deklaracja pol dostepnych tylko dla graczy danego koloru - bazy i domku
std::set<int> base_g = {0,1,11,12,44,56,57,58,59};
std::set<int> base_r = {9,10,20,21,6,16,27,38,49};
std::set<int> base_b = {99,100,110,111,114,71,82,93,104};
std::set<int> base_y = {108,109,119,120,76,61,62,63,64};

// Deklaracja glownej planszy do gry, po ktorej wszystkie pionki sie poruszaja
std::set<int> krzyz = {4,5,6,15,17,26,28,37,39,44,45,46,47,48,50,51,52,53,54,55,65,66,67,68,69,
   70,72,73,74,75,76,81,83,92,94,103,105,114,115,116};

// Deklaracje drog, po ktorych moga sie poruszac pionki danego koloru
int valid_g[45] = {44,45,46,47,48,37,26,15,4,5,6,17,28,39,50,51,52,53,54,65,76,75,
   74,73,72,83,94,105,116,115,114,103,92,81,70,69,68,67,66,55,56,57,58,59,60};
int valid_r[45] = {6,17,28,39,50,51,52,53,54,65,76,75,74,73,72,83,94,105,116,115,
   114,103,92,81,70,69,68,67,66,55,44,45,46,47,48,37,26,15,4,5,16,27,38,49,60};
int valid_y[45] = {76,75,74,73,72,83,94,105,116,115,114,103,92,81,70,69,68,67,66,
   55,44,45,46,47,48,37,26,15,4,5,6,17,28,39,50,51,52,53,54,65,64,63,62,61,60};
int valid_b[45] = {114,103,92,81,70,69,68,67,66,55,44,45,46,47,48,37,26,15,4,5,6,
   17,28,39,50,51,52,53,54,65,76,75,74,73,72,83,94,105,116,115,104,93,82,71,60};
```

Po stworzeniu plansza prezentuje się w następujący sposób.



Rysunek 7: Gotowa plansza z funkcjonalną Bitmapą

2.3.2 Funkcja losuj i funkcjonalność przycisku "losuj"

Rzut kostką odbywa się poprzez funkcję losuj, która przypisuje do zmiennej los liczbę 1, 2, 3, 4, 5 lub 6. Jej doklaracja wygląda następująco

```
void Gra::losuj(){
   srand(time(0));
   los = rand() % 6 + 1;
}
```

Po wciśnięciu przycisku "Losuj" zostanie wywołana metoda, która automatycznie sprawdzi, czy jakiś pionek danego koloru jest już na planszy oraz czy została wylosowana liczba 6. Jeśli oba te warunki nie zostaną spełnione, tura danego gracza jest pomijana z powodu braku dostępnych akcji. Pominięcie tury następuje poprzez funkcję opisaną w następnym podpunkcie.

```
void GameDialog::OnButton1Click(wxCommandEvent& event){
   if(gra.los==0){
      gra.losuj(); // Wywolanie funkcji losuj()
      int l = gra.los;
      StaticBitmap1->SetBitmap(kostka[l-1]);
      // Sprawdzenie dla kazdego gracza, czy wylosowal liczbe 6 oraz czy ma jakies pionki na planszy
      if((gra.gracz==0) && (gra.los != 6) &&
          ((gra.ile_w_bazie_g + gra.ile_skonczylo_g) == 4)){
         zmien_ture();
         gra.los=0;
      } else if((gra.gracz==1) && (gra.los != 6) && ((gra.ile_w_bazie_r + gra.ile_skonczylo_r) ==
          4)){
         zmien_ture();
         gra.los=0;
      } else if((gra.gracz==2) && (gra.los != 6) && ((gra.ile_w_bazie_y + gra.ile_skonczylo_y)==4)){
         zmien_ture();
         gra.los=0;
      } else if((gra.gracz==3) && (gra.los != 6) && ((gra.ile_w_bazie_b + gra.ile_skonczylo_b)==4)){
         zmien_ture();
         gra.los=0;
      }
   }else wxMessageBox( _("Juz wylosowales!"));

}
```

2.3.3 Funkcja zmiany tury

Funkcja ta zmienia turę na następnego gracza. Działa poprzez sprawdzenie którego gracza jest obecnie tura, a następnie dodanie do numeru gracza cyfry 1, oraz użycia metody modulo 4 (w razie gdyby indeks gracza był równy 4). Na koniec funkcja aktualizuje wskaźnik czyja jest tura na kolor gracza.

```
void GameDialog::zmien_ture()
{
   gra.gracz = (gra.gracz+1) % 4;
   if((gra.gracz==0)&&(gra.ile_skonczylo_g==4))
   gra.gracz = (gra.gracz+1) % 4;
   if((gra.gracz==1)&&(gra.ile_skonczylo_r==4))
   gra.gracz = (gra.gracz+1) % 4;
   if((gra.gracz==2)&&(gra.ile_skonczylo_y==4))
   gra.gracz = (gra.gracz+1) % 4;
   if((gra.gracz==3)&&(gra.ile_skonczylo_b==4))
   gra.gracz = (gra.gracz+1) % 4;
   StaticBitmap2->SetBitmap(rysunki[gra.gracz]);
}
```

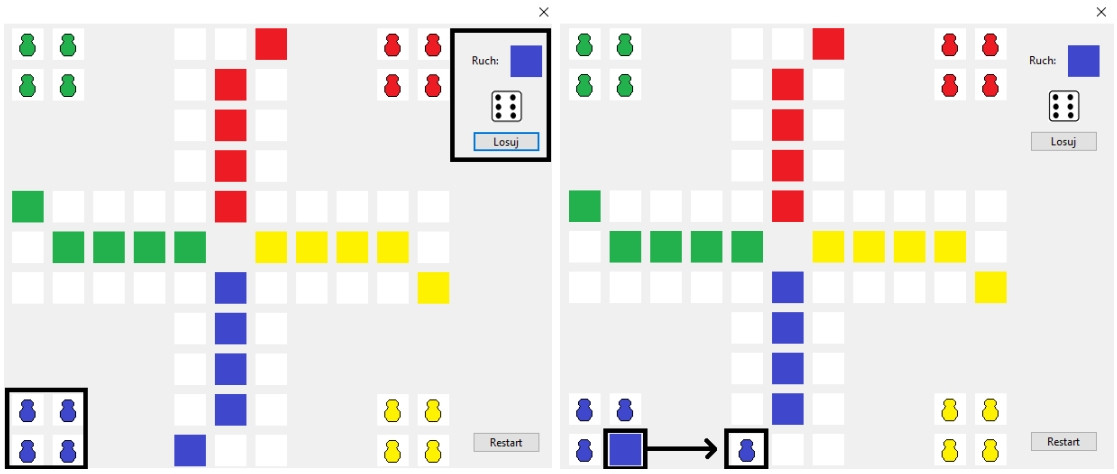
2.3.4 Funkcja do wstawiania pionka z bazy na planszę

Jak nazwa wskazuje, funkcja ta ma za zadanie ustawić pionka z bazy na pole startowe danego gracza. Działa ona według następującego schematu:

- sprawdzenie czyjego gracza jest obecnie tura
- ewentualne zbicie pionka, w razie gdyby pionek innego koloru znajdował się już na polu startowym
- aktualizacja Bitmap (aby w bazie nie został żaden pionek widmo)
- dostosowanie ilości pionków w bazie

Następujący przykład jest dla gracza o kolorze zielonym, ze względu na to, że pozostałe kolory działają analogicznie

```
void GameDialog::wstaw_piona()
{
   // Sprawdzenie, czy tura jest gracza z indeksem 0, analogicznie w pozostalych przypadkach
   if(gra.gracz == 0){
      // Ewentualne zbicie pionka na polu z indeksem 44 (pole startowe gracza zielonego)
      zbij(44);

      // Aktualizacja Bitmap
      pola[home_g[gra.ile_w_bazie_g-1]]->SetBitmap(rysunki[0]);

      // Sprawdzenie ktory pionek zostal wyciagniety z bazy
      for(int i=0; i<4; i++){
         if(current_g[i] == home_g[gra.ile_w_bazie_g-1]){
            current_g[i] = 44;
            i=4;
         }
      }
      auto it=curr_g.find(home_g[gra.ile_w_bazie_g-1]);
      curr_g.erase(it); // Usuniecie ID pionka z bazy
      curr_g.insert(44); // Ustawienie jego ID na pole startowe

      // Dostosowanie ilosci pionkow w bazie
      gra.ile_w_bazie_g = gra.ile_w_bazie_g-1;
      // Ewentualne dostosowanie Bitmap, gdyby na polu startowym znajdowal sie juz pionek zielonego
          gracza
      int ile2=ile_na_pol(44);
      if(ile2==1){
         pola[44]->SetBitmap(pionki[0]);}
      if(ile2==2){
         pola[44]->SetBitmap(pionki2[0]);}
      if(ile2==3){
         pola[44]->SetBitmap(pionki3[0]);}
      if(ile2==4){
         pola[44]->SetBitmap(pionki4[0]);}
   }
   // Analogicznie jak w przypadku gracza koloru zielonego
   if(gra.gracz == 1){
   .
   .
   .
}
```



Rysunek 8: Wyjście pionka z bazy

2.3.5 Funkcja ile na polu

Funkcja zwrcająca ilość pionków tego samego koloru na jednym polu. Działa na zasadzie prostej pętli, która dla każdego pionka sprawdza czy tam stoi. Deklaracja tej funkcji:

```
int GameDialog::ile_na_pol(int p)
{
   int k=0;
   for(int i=0;i<4;i++){
      if(current_g[i]==p)
      k++;
   }
   for(int i=0;i<4;i++){
      if(current_r[i]==p)
      k++;
   }
   for(int i=0;i<4;i++){
      if(current_y[i]==p)
      k++;
   }
   for(int i=0;i<4;i++){
      if(current_b[i]==p)
      k++;
   }
   return k;
}
```

2.3.6 Funkcja sprawdź

Funkcja sprawdzająca czy na kliknięty przez gracza polu stoi jego pionek. x to ID gracza ktory się rusza a p to pole na które klinął. Funkcja zwraca wartość TRUE jeśli gracz kliknął na pole swojego pionka.

```
bool GameDialog::sprawdz(int x,int p){
   if(x==0){
      return (curr_g.find(p) != curr_g.end());
   }
   if(x==1){
      return (curr_r.find(p) != curr_r.end());
   }
   if(x==2){
      return (curr_y.find(p) != curr_y.end());
   }
   if(x==3){
      return (curr_b.find(p) != curr_b.end());
   }
}
```

2.3.7 Funkcja do ruchu pionkiem

Jest to funkcja do ruchu pionkiem. W argumentach dostaje ona liczbę wylosowanych w rzucie kostką oczek oraz ID pola które wcisnął gracz. Jest to przykład dla gracza koloru zielonego, reszta jest napisana w sposób analogiczny

```
void GameDialog::ruch(int los,int pol) aktualne pole
{
   int ile=ile_na_pol(pol); // Sprawdzanie ile przed ruchem stoi pionkow na danym polu
   // Jesli na polu stoi jeden pionek to koeczna jest aktualizacja Bitmap
   if(ile==1){
      if(pol==44){
         pola[pol]->SetBitmap(rysunki[0]);
      }
      else if(pol==6){
         pola[pol]->SetBitmap(rysunki[1]);
      }
      else if(pol==76){
         pola[pol]->SetBitmap(rysunki[2]);
      }
      else if(pol==114){
         pola[pol]->SetBitmap(rysunki[3]);
      }
      else{
         pola[pol]->SetBitmap(rysunki[5]);
      }
   }
   int k;
   if(gra.gracz == 0){
      // ustawienie odpowiedniej grafiki jesli przed ruchem na polu znajdowal sie wiecej niz jeden
          pionek
      if(ile==2)
      pola[pol]->SetBitmap(pionki[0]);
      if(ile==3)
      pola[pol]->SetBitmap(pionki2[0]);
      if(ile==4)
      pola[pol]->SetBitmap(pionki3[0]);
      // Petla do sprawdzania ID pionka znajdujacego sie na danym polu
      for(int i=0;i<4;i++){
         if(pol==current_g[i]){
            k=i;
            i=4;
```

```
        }
    }
    for(int g=0;g<45;g++) // Wyszukanie numeru ID na trasie pionka
    {
        if(current_g[k]==valid_g[g]){
            // Sprawdzenie czy po dodaniu do indeksu aktualnego losu pionek wyszedl by poza plansze,
                jesli tak to gracz traci ture
            if(g+los>=44){
                wxMessageBox(_("Musisz wyrzucic mniejsza liczbe oczek, tracisz ture za zla decyzje"));
                // Ustawienie odpowiedniej grafiki do sytuacji
                if(ile==1)
                pola[pol]->SetBitmap(pionki[0]);
                if(ile==2)
                pola[pol]->SetBitmap(pionki2[0]);
                if(ile==3)
                pola[pol]->SetBitmap(pionki3[0]);
                if(ile==4)
                pola[pol]->SetBitmap(pionki4[0]);

                // Wyjscie z petli przeszukiwania i ustawienie liczby oczek na zero
                g=45;
                gra.los=0;
            }
            else{
                zbij(valid_g[g+los]); // Ewentualne zbicie wrogiego pionka/pionkow
                current_g[k]=valid_g[g+los]; // aktualizacja ID pionka
                auto it = curr_g.find(pol); // aktualizacja multizbioru curr_g
                curr_g.erase(it);
                curr_g.insert(valid_g[g+los]);

                // Sprawdzenie ile po ruchu stoi pionkow na tym polu i dostosowanie odpowiedniej
                    bitmapy
                int ile2=ile_na_pol(current_g[k]);
                if(ile2==1){
                    pola[current_g[k]]->SetBitmap(pionki[0]);}
                if(ile2==2){
                    pola[current_g[k]]->SetBitmap(pionki2[0]);}
                if(ile2==3){
                    pola[current_g[k]]->SetBitmap(pionki3[0]);}
                if(ile2==4){
                    pola[current_g[k]]->SetBitmap(pionki4[0]);}
                // Sprawdzenie, czy pionek znalazl sie w domku
                if(g+los>=40){
                    // Ewenatulne zwiekszenie atrybutu ile pionkow danego gracza skonczylo
                    gra.ile_skonczylo_g=gra.ile_skonczylo_g+1;
                    wxMessageBox(_("Doszedles pionkiem do bazy!"));
                    // Wywolanie funkcji konczacej gre jesli wszystkie pionki doszly do domku
                    if(gra.ile_skonczylo_g==4){
                        koniec();
                    }
                }
                // Wyjscie z petli
                g=45;
            }
        }
    }
}
```

2.3.8 Funkcja zbijania

Funkcja ta służy do zbijania wrogich pionków, jeśli ruch gracza trafi na pole, na którym stoi już jakiś pionek wrogiego gracza. Funkcja działa w następujący sposób:

- Ustalenie który gracz zbija
- Sprawdzenie czy pole, na które rusza się pionek jest zajęte przez pionek/pionki wrogiego gracza
- Zmiana Bitmapy pola w bazie na pole pionka (Pola na którym stał zmieniać nie trzeba, gdyż będzie stał tam pionek, który zbija, co jest zawarte w funkcji "ruch")
- Zwiększenie atrybutu "ile-w-bazie" gracza, którego pionek został zbity

Przykład funkcji dla jednego gracza (zielonego) ze względu na analogię pozostałych kolorów

```
void GameDialog::zbij(int p)
{
   // Ustalenie jakiego koloru jest bijacy pionek (w tym przypadku zielony)
   if(gra.gracz==0)
   {
      // Sprawdzenie, czy na polu, na ktore idzie pionek znajduje sie pionek/pionki innego koloru
      for(int i=0;i<4;i++){
         if(current_b[i]==p){
            current_b[i]=home_b[gra.ile_w_bazie_b];
            pola[current_b[i]]->SetBitmap(pionki[3]);
            auto it = curr_b.find(p);
            curr_b.erase(it);
            curr_b.insert(home_b[gra.ile_w_bazie_b]);
            // Zwiekszenie ilosci pionkow w bazie zbitego gracza
            gra.ile_w_bazie_b= gra.ile_w_bazie_b+1; ile_w_bazie
         }
      }
      // Analogicznie dwa dwoch pozostalych przeciwnikow
      for(int i=0;i<4;i++){
         if(current_y[i]==p){
            current_y[i]=home_y[gra.ile_w_bazie_y];
            pola[current_y[i]]->SetBitmap(pionki[2]);
            auto it = curr_y.find(p);
            curr_y.erase(it);
            curr_y.insert(home_y[gra.ile_w_bazie_y]);
            gra.ile_w_bazie_y= gra.ile_w_bazie_y+1;
         }
      }
      for(int i=0;i<4;i++){
         if(current_r[i]==p){
            current_r[i]=home_r[gra.ile_w_bazie_r];
            pola[current_r[i]]->SetBitmap(pionki[1]);
            auto it = curr_r.find(p);
            curr_r.erase(it);
            curr_r.insert(home_r[gra.ile_w_bazie_r]);
            gra.ile_w_bazie_r= gra.ile_w_bazie_r+1;
         }
      }
   }
```

2.3.9 Funkcja kończąca grę

Funkcja używana w funkcji "ruch", która sumuje ilość graczy, którzy skończyli rozgrywkę oraz informuje ich o zajętym miejscu.

```
void GameDialog::koniec()
{

   gra.ile_graczy_skonczylo++;
   switch (gra.ile_graczy_skonczylo)
   {
      case 1:
      wxMessageBox(_("Wygrales!!"));
      break;
      case 2:
      wxMessageBox(_("Gratujacje! Zajales drugie miejsce!"));
      break;
      case 3:
      wxMessageBox(_("Skonczyles gre! Zajales 3 miejsce"));
      // Po zdobyciu przez ktoregos z graczy 3. miejsca gra sie konczy
      wxMessageBox("Koniec gry!");
      break;
   }
}
```

2.3.10 Funkcjonalność przycisku Reset

Przycisk Reset został stworzony z myślą o przywróceniu gry do etapu początkowego, aby na przykład po skończonej rozgrywce można było zagrać jeszcze raz. Nadpisuje on zmienne takie jak: położenie poszczególnych pionków, wynik rzutu kostką, liczbę graczy, którzy skończyli, ilość pionków w każdej z baz oraz ilość poszczególnych pionków, które doszły do domku, a także aktualizuje grafiki wszystkich pól Bitmapy. Funkcjonalność tego przycisku jest określona następująco:

```
void GameDialog::OnButton2Click(wxCommandEvent& event)
{
   current_g[0] = 0;
   current_g[1] = 1;
   current_g[2] = 11;
   current_g[3] = 12;

   current_r[0] = 9;
   current_r[1] = 10;
   current_r[2] = 20;
   current_r[3] = 21;

   current_y[0] = 108;
   current_y[1] = 109;
   current_y[2] = 119;
   current_y[3] = 120;

   current_b[0] = 99;
   current_b[1] = 100;
   current_b[2] = 110;
   current_b[3] = 111;


   for (auto it=curr_g.begin(); it!=curr_g.end(); ++it)
   curr_g.erase (it);

   for (auto it=curr_r.begin(); it!=curr_r.end(); ++it)
   curr_r.erase (it);
```

```
for (auto it=curr_y.begin(); it!=curr_y.end(); ++it)
curr_y.erase (it);

for (auto it=curr_b.begin(); it!=curr_b.end(); ++it)
curr_b.erase (it);


curr_g.insert(0);
curr_g.insert(1);
curr_g.insert(11);
curr_g.insert(12);

curr_r.insert(9);
curr_r.insert(10);
curr_r.insert(20);
curr_r.insert(21);

curr_y.insert(108);
curr_y.insert(109);
curr_y.insert(119);
curr_y.insert(120);

curr_b.insert(99);
curr_b.insert(100);
curr_b.insert(110);
curr_b.insert(111);


for(int i=0; i<121; i++)
{
   if (krzyz.find(i) != krzyz.end()){
      pola[i]->SetBitmap(rysunki[5]);
   }
   if(base_g.find(i) != base_g.end()){
      pola[i]->SetBitmap(rysunki[0]);
   }
   if(base_r.find(i) != base_r.end()){
      pola[i]->SetBitmap(rysunki[1]);
   }
   if(base_y.find(i) != base_y.end()){
      pola[i]->SetBitmap(rysunki[2]);
   }
   if(base_b.find(i) != base_b.end()){
      pola[i]->SetBitmap(rysunki[3]);
   }

   // pionki
   if(square_g.find(i) != square_g.end()){
      pola[i]->SetBitmap(pionki[0]);
   }
   if(square_r.find(i) != square_r.end()){
      pola[i]->SetBitmap(pionki[1]);
   }
   if(square_y.find(i) != square_y.end()){
      pola[i]->SetBitmap(pionki[2]);
   }
   if(square_b.find(i) != square_b.end()){
      pola[i]->SetBitmap(pionki[3]);
   }
}
```

```
    gra.gracz = 0;

    gra.ile_w_bazie_r = 4;
    gra.ile_w_bazie_y = 4;
    gra.ile_w_bazie_g = 4;
    gra.ile_w_bazie_b = 4;

    gra.ile_skonczylo_r = 0;
    gra.ile_skonczylo_y = 0;
    gra.ile_skonczylo_g = 0;
    gra.ile_skonczylo_b = 0;

    gra.ile_graczy_skonczylo = 0;
    gra.los = 0;
}
```

3 Podział pracy

Projekt ten miał również za zadanie nauczyć nas pracy zespołowej i dzielenia obowiązków. Końcowy podział pracy prezentuje się następująco:

- Interfejs graficzny i utworzenie planszy:
  - Jakub Przemski - 65%
  - Igor Rybiński - 15%
  - Mateusz Rosiński - 20%
- Funkcjonalność przycisku Losuj:
  - Jakub Przemski - 70%
  - Igor Rybiński - 20%
  - Mateusz Rosiński - 10%
- Funkcja do wstawiania pionka z bazy na planszę i do zmiany tury:
  - Jakub Przemski - 55%
  - Igor Rybiński - 35%
  - Mateusz Rosiński - 10%
- Funkcje ”sprawdź” i ”ile na polu”:
  - Jakub Przemski - 45%
  - Igor Rybiński - 45%
  - Mateusz Rosiński - 10%
- Funkcja do ruchu pionkiem:
  - Jakub Przemski - 30%
  - Igor Rybiński - 65%
  - Mateusz Rosiński - 5%
- Funkcja do zbijania:
  - Jakub Przemski - 25%
  - Igor Rybiński - 65%
  - Mateusz Rosiński - 10%

- Funkcja kończąca grę:
  - Jakub Przemski - 40%
  - Igor Rybiński - 30%
  - Mateusz Rosiński - 30%
- Przycisk do resetu gry:
  - Jakub Przemski - 80%
  - Igor Rybiński - 10%
  - Mateusz Rosiński - 10%
- Grafiki:
  - Jakub Przemski - 10%
  - Igor Rybiński - 10%
  - Mateusz Rosiński - 80%
- Pisanie dokumentacji:
  - Igor Rybiński - 10%
  - Mateusz Rosiński - 90%