Szachy

Szymon Makulec Jan Moskal

5 lutego 2024

Cel i efekty pracy

Opis

Projekt zakładał stworzenie gry w szachy dla dwóch graczy. Kod oparty jest na bibliotece wxWidgets, a interfejs użytkownika wykorzystuje siatkę wxBitmapButton z obrazkami figur połączonymi z kolorem tła, do reprezentacji szachownicy i figur na niej położonych.

Tablice obiektów

Użycie obrazków (images[2][13])

```
// Black Pieces Images
images[0][0] = wxBitmap(wxImage("images/D.jpg"));
images[0][1] = wxBitmap(wxImage(_T("images/Pieces/bpB.png")));
images[0][2] = wxBitmap(wxImage(_T("images/Pieces/bpD.png")));
images[0][3] = wxBitmap(wxImage(_T("images/Pieces/brB.png")));
images[0][4] = wxBitmap(wxImage(_T("images/Pieces/brD.png")));
images[0][5] = wxBitmap(wxImage(_T("images/Pieces/bnB.png")));
images[0][6] = wxBitmap(wxImage(_T("images/Pieces/bnD.png")));
images[0][7] = wxBitmap(wxImage(_T("images/Pieces/bbB.png")));
images[0][8] = wxBitmap(wxImage(_T("images/Pieces/bbD.png")));
images[0][9] = wxBitmap(wxImage(_T("images/Pieces/bqB.png")));
images[0][10] = wxBitmap(wxImage(_T("images/Pieces/bqD.png")));
images[0][11] = wxBitmap(wxImage(_T("images/Pieces/bkB.png")));
images[0][12] = wxBitmap(wxImage(_T("images/Pieces/bkD.png")));

// White Pieces Images
images[1][0] = wxBitmap(wxImage("images/B.jpg"));
images[1][1] = wxBitmap(wxImage(_T("images/Pieces/wpB.png")));
images[1][2] = wxBitmap(wxImage(_T("images/Pieces/wpD.png")));
images[1][3] = wxBitmap(wxImage(_T("images/Pieces/wrB.png")));
images[1][4] = wxBitmap(wxImage(_T("images/Pieces/wrD.png")));
images[1][5] = wxBitmap(wxImage(_T("images/Pieces/wnB.png")));
images[1][6] = wxBitmap(wxImage(_T("images/Pieces/wnD.png")));
images[1][7] = wxBitmap(wxImage(_T("images/Pieces/wbB.png")));
images[1][8] = wxBitmap(wxImage(_T("images/Pieces/wbD.png")));
images[1][9] = wxBitmap(wxImage(_T("images/Pieces/wqB.png")));
images[1][10] = wxBitmap(wxImage(_T("images/Pieces/wqD.png")));
images[1][11] = wxBitmap(wxImage(_T("images/Pieces/wkB.png")));
images[1][12] = wxBitmap(wxImage(_T("images/Pieces/wkD.png")));
```

Do wyświetlania figur na przyciskach wxBitmapButton, używamy zestawu dwudziestu sześciu obrazków, reprezentujących figury szachowe. Dwa puste pola (czarne i białe) oraz sześć różnych typów figur (pion, wieża, skoczek, goniec, hetman, król) w dwóch kolorach (biały i czarny) na dwóch tłach (jasnym, ciemnym). images, jest to tablica, w której przechowywane są obrazy figur szachowych używane do wyświetlania na przyciskach wxBitmapButton na szachownicy. Umożliwia łatwe zarządzanie obrazkami figur szachowych w grze.

Użycie przycisków wxBitmapButton (board[8][8])

```
board[0][0] = BitmapButton1;
board[0][1] = new wxBitmapButton(this, wxNewId(), images[0][6],
    wxDefaultPosition, wxDefaultSize, wxBU_AUTODRAW,
    wxDefaultValidator);
board[0][2] = new wxBitmapButton(this, wxNewId(), images[0][7],
    wxDefaultPosition, wxDefaultSize, wxBU_AUTODRAW,
    wxDefaultValidator);
board[0][3] = new wxBitmapButton(this, wxNewId(), images[0][10],
    wxDefaultPosition, wxDefaultSize, wxBU_AUTODRAW,
    wxDefaultValidator);
board[0][4] = new wxBitmapButton(this, wxNewId(), images[0][11],
    wxDefaultPosition, wxDefaultSize, wxBU_AUTODRAW,
    wxDefaultValidator);
board[0][5] = new wxBitmapButton(this, wxNewId(), images[0][8],
    wxDefaultPosition, wxDefaultSize, wxBU_AUTODRAW,
    wxDefaultValidator);
board[0][6] = new wxBitmapButton(this, wxNewId(), images[0][5],
    wxDefaultPosition, wxDefaultSize, wxBU_AUTODRAW,
    wxDefaultValidator);
board[0][7] = new wxBitmapButton(this, wxNewId(), images[0][4],
    wxDefaultPosition, wxDefaultSize, wxBU_AUTODRAW,
    wxDefaultValidator);

for(int j = 0; j < 8; j++){
  if(j%2 == 0){
    board[1][j] = new wxBitmapButton(this, wxNewId(),images[0][2] ,
         wxDefaultPosition, wxDefaultSize, wxBU_AUTODRAW,
        wxDefaultValidator);
  }else{
    board[1][j] = new wxBitmapButton(this, wxNewId(),images[0][1] ,
         wxDefaultPosition, wxDefaultSize, wxBU_AUTODRAW,
        wxDefaultValidator);

  }
}

for(int i = 2; i < 6; i++){
  for(int j = 0; j < 8; j++){
    if((i+j)%2 == 0){
      board[i][j] = new wxBitmapButton(this, wxNewId(), images
          [1][0], wxDefaultPosition, wxDefaultSize, wxBU_AUTODRAW,
          wxDefaultValidator);
    }else{
      board[i][j] = new wxBitmapButton(this, wxNewId(), images
          [0][0], wxDefaultPosition, wxDefaultSize, wxBU_AUTODRAW,
          wxDefaultValidator);
    }
  }
}

for(int j = 0; j < 8; j++){
  if((j)%2 != 0){
    board[6][j] = new wxBitmapButton(this, wxNewId(), images[1][2],
         wxDefaultPosition, wxDefaultSize, wxBU_AUTODRAW,
        wxDefaultValidator);
```

```
32    }else{
33      board[6][j] = new wxBitmapButton(this, wxNewId(), images[1][1],
            wxDefaultPosition, wxDefaultSize, wxBU_AUTODRAW,
          wxDefaultValidator);
34
35    }
36  }
37
38  board[7][0] = new wxBitmapButton(this, wxNewId(), images[1][4],
      wxDefaultPosition, wxDefaultSize, wxBU_AUTODRAW,
      wxDefaultValidator);
39  board[7][1] = new wxBitmapButton(this, wxNewId(), images[1][5],
      wxDefaultPosition, wxDefaultSize, wxBU_AUTODRAW,
      wxDefaultValidator);
40  board[7][2] = new wxBitmapButton(this, wxNewId(), images[1][8],
      wxDefaultPosition, wxDefaultSize, wxBU_AUTODRAW,
      wxDefaultValidator);
41  board[7][3] = new wxBitmapButton(this, wxNewId(), images[1][9],
      wxDefaultPosition, wxDefaultSize, wxBU_AUTODRAW,
      wxDefaultValidator);
42  board[7][4] = new wxBitmapButton(this, wxNewId(), images[1][12],
      wxDefaultPosition, wxDefaultSize, wxBU_AUTODRAW,
      wxDefaultValidator);
43  board[7][5] = new wxBitmapButton(this, wxNewId(), images[1][7],
      wxDefaultPosition, wxDefaultSize, wxBU_AUTODRAW,
      wxDefaultValidator);
44  board[7][6] = new wxBitmapButton(this, wxNewId(), images[1][6],
      wxDefaultPosition, wxDefaultSize, wxBU_AUTODRAW,
      wxDefaultValidator);
45  board[7][7] = new wxBitmapButton(this, wxNewId(), images[1][3],
      wxDefaultPosition, wxDefaultSize, wxBU_AUTODRAW,
      wxDefaultValidator);
46
47  for(int i = 0;i < 8; i++){
48    for(int j = 0; j < 8; j++){
49      if(i + j == 0){
50        j++;
51      }
52      GridSizer1->Add(board[i][j], 1, wxALIGN_CENTER_HORIZONTAL|
          wxALIGN_CENTER_VERTICAL, 5);
53      Connect(board[i][j]->GetId(),wxEVT_COMMAND_BUTTON_CLICKED,(
          wxObjectEventFunction)&Chess2DDialog::OnBitmapButton1Click)
          ;
54
55
56    }
57  }
```

Każdy element board[i][j] (8x8) reprezentuje jedno pole z wyświetloną figurą danego koloru i tłem na szachownicy, zawiera przycisk do interakcji z użytkownikiem. Każdy przycisk jest obsługiwany przez tą samą funkcję OnBitmapButton1Click.

Użycie figur (pieces[2][7])

```
pieces[0][0] = new Pawn(false);
pieces[0][1] = new Rook(false);
pieces[0][2] = new Knight(false);
pieces[0][3] = new Bishop(false);
pieces[0][4] = new Queen(false);
pieces[0][5] = new King(false);
pieces[0][6] = new Rook(false);
pieces[1][0] = new Pawn(true);
pieces[1][1] = new Rook(true);
pieces[1][2] = new Knight(true);
pieces[1][3] = new Bishop(true);
pieces[1][4] = new Queen(true);
pieces[1][5] = new King(true);
pieces[1][6] = new Rook(true);
```

pieces[i][j] jest to dwuwymiarowa tablica wskaźników do obiektów klasy Piece. Jest to tablica, w której przechowywane są obiekty reprezentujące różne typy figur szachowych w dwóch kolorach: białym i czarnym. Każdy obiekt klasy Piece przechowuje informacje o tym czy figura już się ruszała. Dlatego tworzymy dwie wieże, żeby każdej z osobna można było sprawdzać czy się ruszała. W przypadku pionów uznaliśmy tworze oddzielnych Pawnów za zbyteczne, ponieważ każdy może poruszać się tylko w 1 kierunku.

Przechowywanie przycisków i figur w jednym obiekcie (squares[8][8])

```
for(int i = 0; i < 8; i++){
  for(int j = 0; j < 8; j++){
    if((i+j) % 2 == 0){
      squares[i][j] = new Square(i, j , true, board[i][j]);
    }else{
      squares[i][j] = new Square(i, j , false, board[i][j]);
    }
  }
}

for(int i = 0; i < 8; i++){
  squares[1][i]->setPiece(pieces[0][0]);
  squares[6][i]->setPiece(pieces[1][0]);
}

// Assigning other Pieces to Squares
squares[0][0]->setPiece(pieces[0][1]);
squares[0][1]->setPiece(pieces[0][2]);
squares[0][2]->setPiece(pieces[0][3]);
squares[0][3]->setPiece(pieces[0][4]);
squares[0][4]->setPiece(pieces[0][5]);
squares[0][5]->setPiece(pieces[0][3]);
squares[0][6]->setPiece(pieces[0][2]);
squares[0][7]->setPiece(pieces[0][6]);
squares[7][0]->setPiece(pieces[1][1]);
squares[7][1]->setPiece(pieces[1][2]);
```

```
27 squares[7][2]->setPiece(pieces[1][3]);
28 squares[7][3]->setPiece(pieces[1][4]);
29 squares[7][4]->setPiece(pieces[1][5]);
30 squares[7][5]->setPiece(pieces[1][3]);
31 squares[7][6]->setPiece(pieces[1][2]);
32 squares[7][7]->setPiece(pieces[1][6]);
```

squares[i][j] to dwuwymiarowa tablica wskaźników do obiektów typu Square, reprezentująca pola szachownicy. Każdy element tej tablicy to obiekt Square, który przechowuje informacje o polu (przycisk, figura oraz wiersz, kolumna i kolor pola).

Struktura klas

Projekt składa się z kilku klas, które są ze sobą powiązane w celu zorganizowania logiki gry w szachy. Poniżej przedstawiona jest struktura klas, ich relacje oraz krótkie opisy.

Chess2DDialog: Główna klasa projektu, dziedzicząca po wxDialog, reprezentująca okno gry w szachy. Odpowiada za obsługę interfejsu użytkownika i inicjalizację gry.

Board: Klasa reprezentująca szachownicę. Odpowiada za logikę gry, przechowuje obiekty klasy Square reprezentujące pola na szachownicy.

Square: Klasa reprezentująca pole na szachownicy. Zawiera informacje o kolorze pola, wierszu i kolumnie pola oraz referencje do przycisku wxBitmapButton i do figury Piece.

Piece: Klasa abstrakcyjna, po niej dziedziczą konkretne rodzaje figur szachowych. Zawiera podstawowe metody i pola, takie jak: kolor, czy to czy figura się ruszała. Wprowadza metode wirtualną dotyczącą zwracania typu figury.

Pawn, Rook, Knight, Bishop, Queen, King: Klasy dziedziczące po klasie Piece, reprezentujące poszczególne rodzaje figur szachowych. Każda z tych klas implementuje pola z informacjami na temat danej figury.

Relacje między klasami:

Chess2DDialog zawiera obiekt klasy **Board**, który odpowiada za logikę gry i przechowuje informacje o szachownicy.

Klasa **Board** zarządza tablicą obiektów **Square** reprezentujących pola na szachownicy.

Każdy obiekt **Square** zawiera referencję do obiektu wxBitmapButton oraz informacje o obecności figury (**Piece**).

Klasy **Pawn, Rook, Knight, Bishop, Queen, King** dziedziczą po klasie **Piece** i zawierają specyficzne implementacje metod związanych z ich ruchami.

Ta struktura klas pozwala na uporządkowanie kodu i łatwość rozbudowy gry.

Szczegóły konkretnych klas:

Chess2DDialog

```
wxBitmapButton* board[8][8];
wxBitmap images[2][13];

int counter = 1;
int whiteOrBlack = 1;

_B = new Board(1);
```

W konstruktorze klasy tworzony jest obiekt klasy Board, który będzie reprezentować szachownicę w grze. Tutaj są wczytywane obrazki do images[2][13] oraz tworzone przyciski umieszczane w board[8][8], a także liczniki odpowiadające za liczenie kliknięć i pilnowanie czyj mamy ruch. Zawiera również obsługę zdarzeń związanych z interakcją użytkownika (OnBitmapButtonClick()). Także w tym miejscu jest kod odpowiedzialny za wyświetlanie szachownicy.

Board

```
class Board
{
  public:
  Board(int i);
  ~Board();
  void cleaning();
  void creatingSquares();
  void creatingPieces();
  void assigningPieces();
  void restart();
  void setClicked(int _nrBB);
  void setDestination(int _nrBB);
  bool isClickedPiece();
  bool isDestinationPiece();
  bool ArePiecesSameColor();
  bool isGoodColorMoving();
  void updateSquares(Square* _clicked, Square* _destination);
  bool isKingInside();
  void wasKingMoving();
  int whereIsKing();
```

```
  bool castling();
  bool pawnBlockedByPieceInFront();
  void pawnMovesButNothingIsInFront();
  void pawnTakes();
  void pawnPromotion();
  void whereICanMove();
  bool isInSetOfMoves();
  bool isSomethingBetween(Square* _squareOne, Square* _squareTwo,
      int _typeInt);
  bool isBeatable(Square* _square);
  bool moveSimulation();
  Piece* pieces[2][7];
  Square* squares[8][8];
  Square* clicked;
  Square* destination;
  Square* whiteKing;
  Square* blackKing;
  set<int> setOfMoves;
  Square* squareBetween;
};
```

Klasa Board jest centralnym punktem logiki gry, zarządzającym szachownicą. Zawiera tablicę obiektów Square oraz tablicę obiektów Piece. Jest w niej kod odpowiedzialny za ustawienie początkowego rozmieszczenia figur na szachownicy. Klasa Board implementuje logikę ruchu figur, sprawdzając, czy dany ruch jest zgodny z zasadami gry w szachy. Odpowiada za obsługę zdarzeń związanych z interakcją użytkownika na poziomie szachownicy, takich jak kliknięcia na pola. Po wykonaniu ruchu odpowiednio aktualizuje widok.

Square

```
class Square
{
  public:
  friend class Board;
  Square(int _row, int _col, bool _backgroundColor, wxBitmapButton*
      _button, Piece* _piece = nullptr);
  ~Square();
  int getRow();
  int getCol();
  bool getBackgroundColor();
  wxBitmapButton* getButton();
  Piece* getPiece();
  void setPiece(Piece* _piece);

  protected:
  int row;
  int col;
  bool backgroundColor;
  wxBitmapButton* button;
  Piece* piece;
};
```

Klasa Square reprezentuje pojedyncze pole na szachownicy. Przechowuje informację o kolorze pola na szachownicy (jasne/ciemne), wierszu i kolumnie pola. Zawiera referencję do obiektu klasy Piece reprezentującego figurę. Jeśli pole jest puste, wartość ta to nullptr. Przechowuje referencję do przycisku, który jest powiązany z danym polem na szachownicy.

Piece

```
class Piece
{
  public:
  Piece(bool _color, bool _moved);
  virtual ~Piece();
  bool getColor();
  bool getMoved();
  void setMoved();
  virtual int getTypeInt()=0;

  protected:
  bool color;
  bool moved;
};
```

Klasa Piece stanowi abstrakcję dla poszczególnych rodzajów figur (piona, wieży, skoczka, gońca, hetmana, króla). Klasa posiada konstruktor, który inicjuje kolor figury oraz informację o tym, czy figura wykonała ruch. Posiada metode virtual int getTypeInt()=0: Jest to metoda czysto wirtualna, zwracająca typ figury jako wartość liczbową (przydatna do indeksach naszych tablic).

Pawn, Rook, Knight, Bishop, Queen, King

```
class Pawn:public Piece
{
  public:
  Pawn(bool _color, bool _moved = false);
  ~Pawn();
  int getTypeInt();

  private:
  int TypeInt = 1;
};
```

Klasy te dziedziczą po klasie Piece i reprezentują konkretne figury w grze. Poniżej przedstawiono kluczowe elementy tech klas: Posiada informację o tym, czy figura wykonała swój pierwszy ruch w grze. To istotne np. dla reguły pierwszego ruchu piona lub dla roszady Posiada informację (int) o typie konkretnej figury. Wszytskie te klasy są identyczne.

Metody w klasie Board:

Odpowiedzialne za restartowanie gry oraz usuwanie i tworzenie obiektów:

```
void cleaning();
void creatingSquares();
void creatingPieces();
void assigningPieces();
void restart();
```

• **cleaning()**: Usuwa zmienne tworzone dynamicznie (korzysta z niej destruktor i restart).
• **creatingSquares()**: Tworzy obiekty reprezentujące pola na szachownicy.
• **creatingPieces()**: Tworzy obiekty reprezentujące figury szachowe.
• **assigningPieces()**: Przypisuje figury do odpowiednich pól na szachownicy (pozycje startowe).
• **restart()**: Przywraca początkowe ustawienie figur na szachownicy.
Odpowiedzialne za obsługę interakcji użytkownika z szachownicą, kontrolujące kliknięcia, aby zapobiec nieoczekiwanym i niechcianym zachowaniom:

```
void setClicked(int _nrBB);
void setDestination(int _nrBB);
bool isClickedPiece();
bool isDestinationPiece();
bool arePiecesSameColor();
bool isGoodColorMoving();
```

• **setClicked(int _nrBB)**: ustawia kliknięte pole
• **setDestination(int _nrBB)**: ustawia docelowe pole
• **isClickedPiece()**: sprawdza czy w kliknięty polu jest jakaś figura (używana po to żeby sprawdzić czy w wybranym polu jest jakaś figura, którą można się ruszyć)
• **isDestinationPiece()**: sprawdza czy w docelowym polu jest jakaś figura
• **arePiecesSameColor()**: sprawdza czy w kliknięty i docelowym polu są figury tego samego koloru (używana żeby zapobiec ruszaniu się na pola gdzie mamy własne figury)
• **isGoodColorMoving()**: sprawdza czy ruch wykonuje kolor, którego jest teraz kolej
Odpowiedzialna za aktualizacje szachownicy po ruchu:

```
void updateSquares(Square* _clicked, Square* _destination);
```

• **updateSquares(Square* _clicked, Square* _destination)**: ustawia obrazki i figury na polu kliknięty i docelowym
Odpowiedzialne za lokalizację oraz zmianę pozycji króli królów:

```
bool isKingInside();
void wasKingMoving();
int whereIsKing();
```

• **isKingInside()**: sprawdza czy król jest w kliknięty polu
• **wasKingMoving()**: sprawdza na koniec ruchu czy król się ruszył i aktualizuje jego lokalizacje
• **whereIsKing()**: zwraca pozycje króla strony, która jest teraz na ruchu

Odpowiedzialna za roszadę

```
bool castling();
```

• **castling()**: sprawdza czy w danym ruchu roszada miała miejsce, jeśli nie miała lub się udała to zwraca false, jeśli nie można było jej zrobić zwraca true

Odpowiedzialne za ruch i bicie (przez niego) piona oraz jego promocje:

```
bool pawnBlockedByPieceInFront();
void pawnMovesButNothingIsInFront();
void pawnTakes();
void pawnPromotion();
```

• **pawnBlockedByPieceInFront()**: sprawdza czy ruch pion jest blokowany przez figury przed nim
• **pawnMovesButNothingIsInFront()**: wykonuje ruch piona, gdy nic przed nim nie stoi
• **pawnTakes()**: wykonuje bicie po przekątnej gdy są tam jakieś figury przeciwnika
• **pawnPromotion()**: sprawdza czy pion znajduje się w pierwszym rzędzie u przeciwnika wtedy zamienia go w hetmana

Odpowiedzialne za sprawdzanie legalnych ruchów dla każdej figury:

```
void whereICanMove();
bool isInSetOfMoves();
bool isSomethingBetween(Square* _squareOne, Square* _squareTwo, int
     _typeInt);
```

• **whereICanMove()**: tworzy zbiór legalnych ruchów dla figury wybranej przez gracza
• **isInSetOfMoves()**: sprawdza czy ruch wybrany przez gracza jest w zbiorze legalnych ruchów dla danej figury
• **isSomethingBetween(Square* _squareOne, Square* _squareTwo, int _typeInt)**: sprawdza czy między dwoma polami jest jakaś figura (używane do sprawdzenia czy nic nie stoi na drodze na dane pole)

Odpowiedzialne za szachowanie i symulacje ruchów

```
bool isBeatable(Square* _square);
bool moveSimulation();
```

• **isBeatable(Square* _square)**: sprawdza czy wybrana figura jest podbiciem przez jakąś figure (używane głównie do sprawdzania szachowania króla)
• **moveSimulation()**: przeprowadza symulacje ruchu (potem cofa wprowadzone zmiany) w sytuacji gdy król jest pod szachem, jeśli ruch gracza sprawia, że król nie jest już pod szachem zwraca true, w przeciwnym wypadku false

Obsługa przycisków (OnBitmapButton1Click)

```
void Chess2DDialog::OnBitmapButton1Click(wxCommandEvent& event){
  int nrBB = event.GetId() - 100;

  if(counter%2 != 0){
    _B->setClicked(nrBB);
    if(!(_B->isClickedPiece()) || !(_B->isGoodColorMoving())){
      return;
    }
    counter++;
    return;
  }

  if(counter%2 == 0){
    _B->setDestination(nrBB);
    if(_B->isDestinationPiece() && _B->arePiecesSameColor()){
      counter--;
      return;
    }
    _B->whereICanMove();

    if(!_B->isInSetOfMoves()){
      counter--;
      return;
    }
    if(_B->isSomethingBetween(_B->clicked, _B->destination, _B->
        clicked->getPiece()->getTypeInt())){
      counter--;
      return;
    }

    if(_B->moveSimulation()){
      wxLogMessage("Tutaj nie wolno");
      counter--;
      return;
    }

    if(_B->castling()){
      counter--;
      return;
    }
    _B->updateSquares(_B->clicked, _B->destination);

    _B->pawnPromotion();
    _B->wasKingMoving();

```

```
45      counter++;
46      whiteOrBlack = (whiteOrBlack + 1)%2;
47      return;
48    }
49  }
```

Funkcja OnBitmapButton1Click sprawdza, czy kliknięcia odpowiadają zasadom gry. Uniemożliwia nieprawidłowe ruchy oraz obsługuje specjalne przypadki, takie jak promocja piona na hetmana czy wykonanie roszady. Dodatkowo aktualizuje stan planszy po wykonaniu ruchu gracza.