

Podstawy teleinformatyki

**Rozpoznawanie obrazu z gry w warcaby oraz
wizualizacja stanu gry na komputerze.**

Bartosz Kaaz I21 106085

Spis treści

1 Wprowadzenie.....	3
1.1 Wstęp.....	3
1.2 Etapy projektu.....	3
1.3 Przyjęte założenia.....	3
1.4 Zasady gry.....	3
2 Realizacja.....	4
2.1 Aplikacja testowa.....	4
2.1.1 Opis aplikacji.....	4
2.1.2 Funkcjonalność.....	5
2.1.3 Implementacja aplikacji testowej.....	8
2.2 Logika aplikacji.....	10
2.2.1 Opis logiki.....	10
2.2.2 Funkcjonalność.....	12
2.2.3 Implementacja logiki.....	18
3 Wnioski.....	21

1 Wprowadzenie

1.1 Wstęp

Celem naszego projektu była wizualizacja na komputerze stanu gry w warcaby. Aby zrealizować nasze zadanie wydzieliliśmy w projekcie trzy podstawowe etapy. Ich realizacją odpowiednio podzieliliśmy się w naszej grupie, a następnie efekty pracy zostały połączone w jeden spójny projekt.

1.2 Etapy projektu

Podział zadań w projekcie wyglądał następująco:

- rozpoznawanie obrazu przechwyconego z kamery
- logika gry
- aplikacja WPF łącząca całość

Moim zadaniem była implementacja logiki gry polegająca na wykrywaniu błędów podczas ruchu pionkami oraz zgłaszanie poprawnie wykonanych ruchów. Pozostałą częśćią zajęli się pozostali dwaj koledzy z naszej grupy.

1.3 Przyjęte założenia

W celu bezbłędnego rozpoznawania poszczególnych stanów na szachownicy przyjęliśmy następujące założenia:

- szachownica jest wymiaru 8x8
- szachownica jest ustawiona w taki sposób, że w lewym górnym rogu znajduje się białe pole
- pionki są reprezentowane przez różne kolory:
 - kolor czerwony: pionek pierwszego gracza
 - kolor żółty: kolor drugiego gracza
 - kolor niebieski: kolor damki pierwszego gracza
 - kolor zielony: kolor damki drugiego gracza
- przy szachownicy znajduje się stojak z kamerą, która znajduje się bezpośrednio nad szachownicą i rejestruje przebieg gry

1.4 Zasady gry

Jak wiadomo gra ta posiada wiele odmian. W celu implementacji musieliśmy zdecydować jakie zasady gry będą obowiązywać w trakcie rozgrywki na naszej szachownicy. Poniżej została przedstawiona lista zasad:

- każdy z graczy rozpoczyna grę posiadając 12 pionów

- gracz pierwszy posiada czerwone piony i rozpoczyna grę w górnej części szachownicy (czyli tej gdzie w lewym narożniku znajduje się białe pole)
- gracz drugi posiada żółte piony i rozpoczyna grę w dolnej części szachownicy (czarne pole w lewym narożniku)
- pierwszy ruch wykonuje gracz pierwszy – czerwone piony
- następnie gracze wykonują ruchy na zmianę chyba, że po udanym biciu poruszany pionek znajduje się w takiej pozycji, że musi wykonać kolejne obowiązkowe bicie. Wtedy wykonuje je.
- Piony mogą poruszać się o jedno pole do przodu po przekątnej (na ukos) na wolne pola.
- Bicie pionem następuje przez przeskoczenie sąsiedniego pionu (lub damki) przeciwnika na pole znajdujące się tuż za nim po przekątnej (pole to musi być wolne). Zbite piony są usuwane z planszy po zakończeniu ruchu.
- Piony mogą bić zarówno do przodu, jak i do tyłu
- W jednym ruchu wolno wykonać więcej niż jedno bicie tym samym pionem, przeskakując przez kolejne piony (damki) przeciwnika.
- Bicia są obowiązkowe.
- Pion, który dojdzie do ostatniego rzędu planszy, staje się damką, przy czym jeśli znajdzie się tam w wyniku bicia i będzie mógł wykonać kolejne bicie (do tyłu), to będzie musiał je wykonać i nie staje się wtedy damką.
- Kiedy pion staje się damką, kolej ruchu przypada dla przeciwnika.
- Damki mogą poruszać się w jednym ruchu o dowolną liczbę pól do przodu lub do tyłu po przekątnej, zatrzymując się na wolnych polach.
- Bicie damką jest możliwe z dowolnej odległości po linii przekątnej i następuje przez przeskoczenie pionu (lub damki) przeciwnika, za którym musi znajdować się co najmniej jedno wolne pole -- damka przeskakuje na dowolne z tych pól i może kontynuować bicie (na tej samej lub prostopadłej linii).

2 Realizacja

2.1 Aplikacja testowa

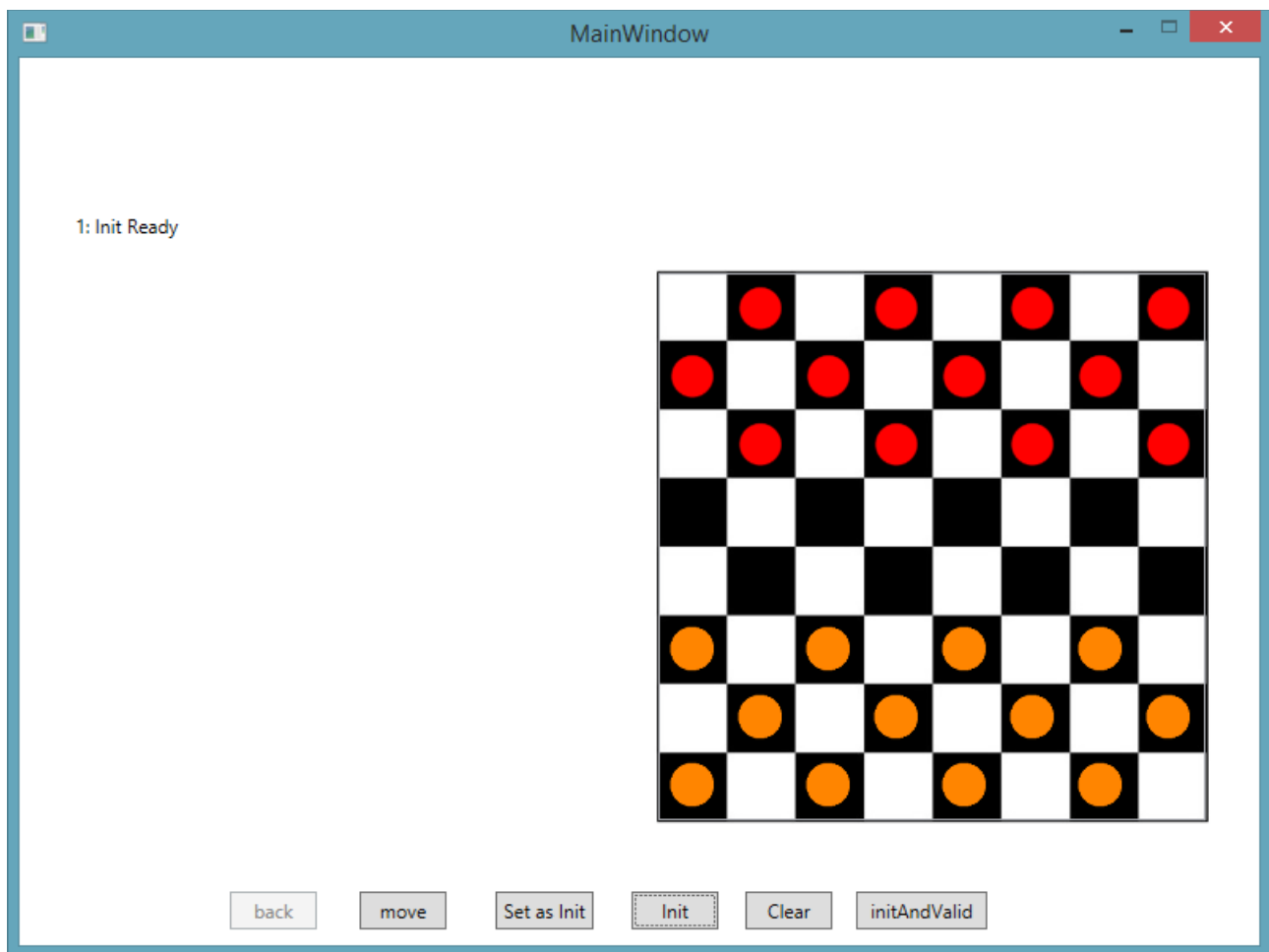
2.1.1 Opis aplikacji

Z powodu podziału zadań konieczne było utworzenie aplikacji testowej na której mógłbym testować efekty swojej pracy. Było to konieczne abym mógł pracować już w tym czasie kiedy koledzy z grupy zajmowali się pracą nad przechwytywaniem obrazu z kamery.

Jak później się okazało, pisanie logiki opartej o aplikację testową pozwoliło przewidzieć problemy o jakich nie pomyśleliśmy na wstępnych spotkaniach. Oraz rozwiązać je dzięki czemu w momencie łączenia efektów naszych prac nie mieliśmy praktycznie żadnych problemów.

Aplikacja ta umożliwia stawianie pionów w dowolnym miejscu oraz symulowanie różnych sytuacji do jakich może dojść podczas gry.

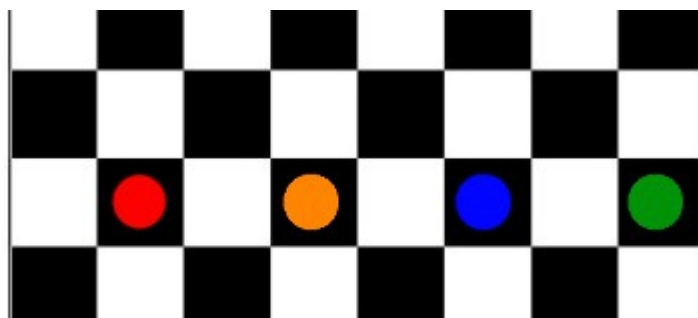
Wygląda ona wygląda następująco:



2.1.2 Funkcjonalność

Jak zostało wcześniej wspomniane aplikacja ta pozwala na symulację dowolnej sytuacji jaka może wystąpić podczas gry w warcaby. Wybraną sytuację możemy symulować ustawiając wybrany pion na dowolnym polu. Możemy tego dokonać klikając na pole myszką.

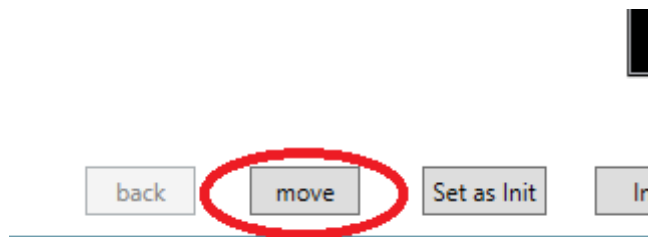
- jedno kliknięcie powoduje pojawienie się czerwonego piona
- dwa kliknięcia to pojawienie się żółtego piona
- trzy kliknięcia to pion niebieski
- cztery kliknięcia to pion zielony



Tak jak wcześniej zostało to opisane, każdy kolejny kolor jest przypisany do konkretnego piona na szachownicy oraz do poszczególnych graczy.

Naciśnięcie po raz piąty na pole powoduje wyczyszczenie pola.

Po ustawieniu w odpowiedni sposób pionów należy wykonać ruch klikając myszką na przycisk *move* przedstawionego na poniższym zrzucie ekranu



Po wykonaniu ruchu po lewej stronie okna aplikacji pojawiają się logi z informacją pochodzącą z części projektu odpowiedzialnej za logikę gry

```
6: Error: Player1- Obligatory Capture
5: OK: Player2- Move Success
4: OK: Player1- Move Success
3: OK: Player2- Move Success
2: OK: Player1- Move Success
1: Init Ready
```

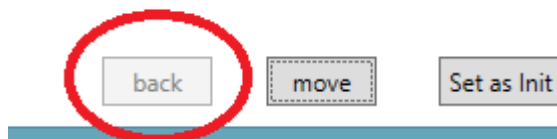


Komunikaty te zostaną omówione w dalszej części dokumentacji.

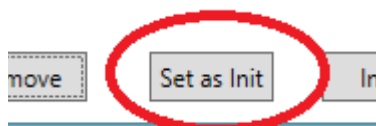
Poza ustawianiem pionów, poruszaniem się oraz logowaniem komunikatów aplikacja posiada również szereg przycisków ułatwiających pracę użytkownika podczas testowania modułu.

Między innymi są to:

- przycisk *back* – pozwala on wycofać błędny stan szachownicy w przypadku jeśli użytkownik wykonał zły ruch:



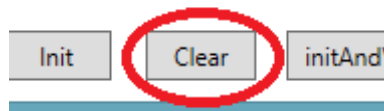
- przycisk *Set as Init* – za jego pomocą możliwe jest ustawienie dowolnej konfiguracji szachownicy i zmuszenia części logiki do przyjęcia jej jako poprawnej i startowej. Właśnie za pomocą tego przycisku możliwa jest symulacja niemal każdej sytuacji.



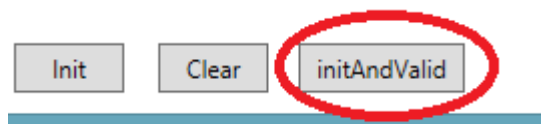
- przycisk *Init* służy do ustawienia poprawnego, początkowego stanu szachownicy. Czyli po 12 pionów dla każdego gracza rozstawionych w odpowiednich pozycjach.



- przycisk *Clear* czyści szachownicę



- *initAndValid* działa podobnie jak *Init* z tą różnicą, że zaraz po początkowym rozstawieniu pionów na szachownicy zostaje wywołana funkcja sprawdzająca poprawność rozstawienia.



2.1.3 Implementacja aplikacji testowej

Aplikacja ta w całości została napisana w technologii WPF.

Zostały też zaimplementowane funkcje łączące się z logiką (która zostanie opisana poniżej) i pozwalające a wypisywanie efektów działania aplikacji. Są to między innymi:

```
private void Button_Init_Click(object sender, RoutedEventArgs e)
{
    Clear();
    DrawPawnsLine(0, 1, 1, RedPawn);
    DrawPawnsLine(1, 0, 1, RedPawn);
    DrawPawnsLine(2, 1, 1, RedPawn);
    DrawPawnsLine(5, 0, 2, YellowPawn);
    DrawPawnsLine(6, 1, 2, YellowPawn);
    DrawPawnsLine(7, 0, 2, YellowPawn);
    _checkCheckers.InitBoard(boardArrayToFieldState());
    InitConfiguration();
}
```

Funkcja odpowiedzialna za początkowe ustawienie szachownicy

```
private void DrawPawnsLine(int start, int offset, int pawnType, BitmapImage
pwanImage)
{
    for (var i = offset; i < 8; i += 2)
```

```

    {
        var index = start * 8 + i;
        _boardArray[start, i] = pawnType;
        Images[index].Visibility = Visibility.Visible;
        Images[index].Source = pwanImage;
    }
}

```

Funkcja ta ma za zadanie rysowanie rzędu pionów we wskazanym miejscu na szachownicy.

```

private void Clear()
{
    for (var i = 0; i < 8; i++)
        for (var j = 0; j < 8; j++)
        {
            var index = i * 8 + j;
            _boardArray[i, j] = 0;
            Images[index].Visibility = Visibility.Hidden;
        }
}

```

Czyści całą szachownicę

```

private void Button_SetInit_Click(object sender, RoutedEventArgs e)
{
    _checkCheckers.InitBoard(boardArrayToFieldState());
    InitConfiguration();
}

```

Ustawia aktualny stan szachownicy jako początkowy

```

private void Button_Move_Click(object sender, RoutedEventArgs e)
{
    var board = boardArrayToFieldState();
    TestLog(_checkCheckers.UpdateAndValidBoard(board));
}

```

Wykonanie ruchu

```

private void Button_InitAndValid_Click(object sender, RoutedEventArgs e)
{
    var initAndValid = _checkCheckers.InitAndValid(boardArrayToFieldState());
    if (initAndValid) InitConfiguration();
    TestLog(initAndValid ? "InitAndValid Success" : "InitAndValid Failed");
}

```

Ustawienie stanu początkowego szachownicy oraz sprawdzanie poprawności wykonania tej operacji

Stan pionów na szachownicy jest reprezentowany w tablicy 8x8

```
private readonly int[,] _boardArray = new int[8, 8];
```

Która po aktualizacji jest odpowiednio konwertowana z wartości int na strukturę `FieldState`, a następnie przekazywana do części projektu zajmującego się logiką

```
private FieldState[,] boardArrayToFieldState()
{
    var result = new FieldState[8, 8];
    for (int i = 0; i < 8; i++)
    {
        for (int j = 0; j < 8; j++)
        {
            switch (_boardArray[i, j])
            {
                case 0:
                    result[i, j] = FieldState.Empty;
                    break;
                case 1:
                    result[i, j] = FieldState.RedPawn;
                    break;
                case 2:
                    result[i, j] = FieldState.YellowPawn;
                    break;
                case 3:
                    result[i, j] = FieldState.BluePawn;
                    break;
                case 4:
                    result[i, j] = FieldState.GreenPawn;
                    break;
                default:
                    result[i, j] = FieldState.Empty;
                    break;
            }
        }
    }
    return result;
}
```

2.2 Logika aplikacji

2.2.1 Opis logiki

Logika aplikacji została zrealizowana i dołączona do głównej aplikacji jako projekt typu `ClassLibrary`. Dzięki czemu mogła działać zarówno w aplikacji testowej jak i w miejscu docelowym.

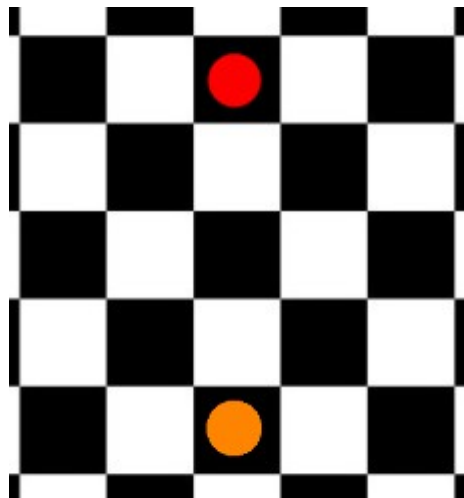
Głównym jej zadaniem jest przyjmowanie na wejściu tablicy reprezentującej stan szachownicy i zwracanie odpowiednich komunikatów w zależności od podanej zmiany.

2.2.2 Funkcjonalność

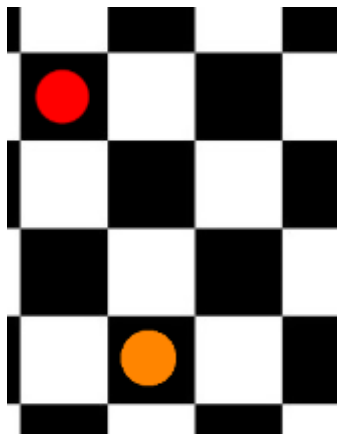
Aplikacja logiki została przygotowana na odpowiedź spowodowaną szeregiem sytuacji występujących na szachownicy. Są to między innymi:

Wykrywanie poprawnie wykonanego ruchu:

- w sytuacji kiedy przychodzi kolej na gracza, który musi wykonać ruch i nie występują żadne szczególne okoliczności (takie jak konieczne bicie lub zmiana piona na damę). Jest on zobowiązany do wykonania ruchu. Może to przedstawiać poniższa sytuacja:



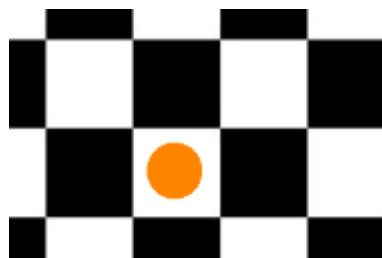
Gracz czerwony w tej chwili wykonuje ruch, po czym logowana jest informacja o powodzeniu.



2: OK: Player1- Move Success
1: Init Ready

Zgłaszanie błędnego ruchu

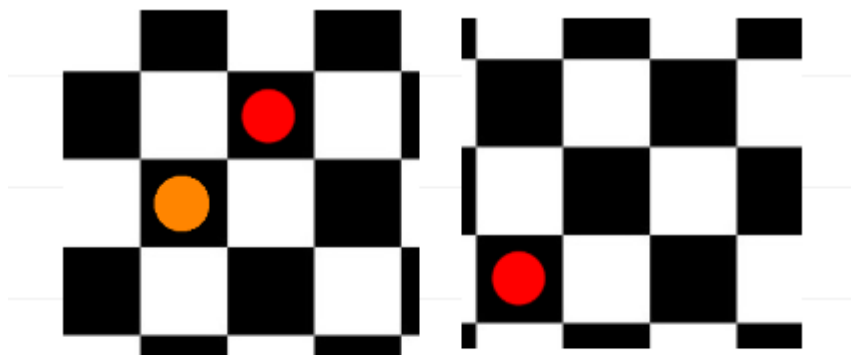
- Jeśli użytkownik wykona ruch niepoprawny, np. takie jaki został pokazany na poniższym zrzucie ekranu. Wtedy aplikacja loguje odpowiednią informację:



3: Error: Player2- Invalid move
2: OK: Player1- Move Success
1: Init Ready

Wykrywanie obowiązkowego bicia

- w momencie kiedy pion gracza aktualnie wykonującego ruch znajduje się w takiej pozycji, że możliwe jest bicie. Zgodnie z zasadami musi je wykonać. Sytuacja ta została przedstawiona poniżej:



Logowana jest także odpowiednia informacja:

2: OK: Player1- Capture success
1: Init Ready

Pominięcie obowiązkowego bicia

- Aplikacja nie może dopuścić do sytuacji, w której użytkownik mając możliwość wykonania bicia nie wykona go. W takim przypadku logowany jest błąd i aplikacja czeka aż użytkownik wróci do poprzedniego stanu.

Logowana jest następująca informacja:

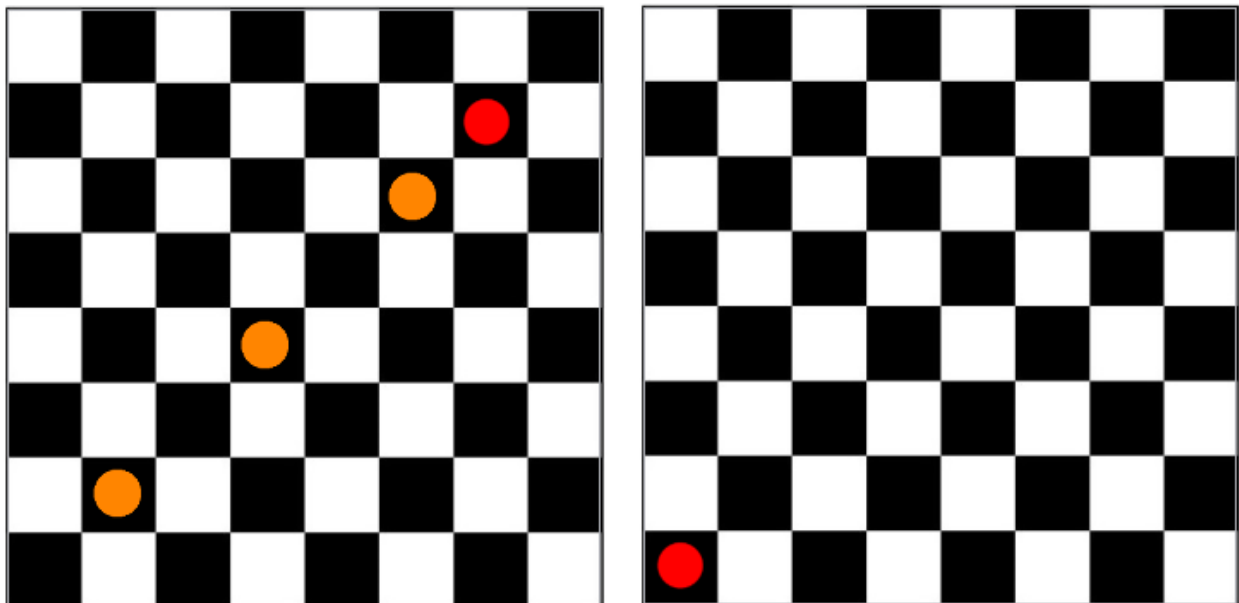
2: Error: Player1- Obligatory Capture
1: Init Ready

Bicie kilka razy pod rząd

Na szachownicy może dojść do takiej sytuacji, że po wykonanym biciu przez gracza znajduje się on w takiej pozycji, że jest on zmuszony wykonać kolejne bicie.

W takiej sytuacji aplikacja pilnuje aby po wykonaniu takiego ruchu nie zaktualizowała się kolejność ruchów graczy (jedzie ponownie ten sam gracz, który wykonał bicie)

Sytuacja ta może przedstawiać się następująco:



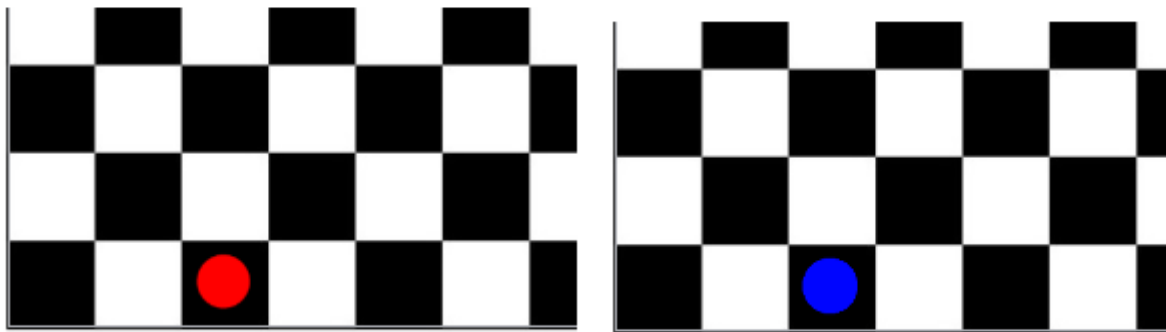
Logowany jest także odpowiedni komunikat, tyle razy ile zostało wykonane bicie.

1: OK: Player1- Capture success
2: OK: Player1- Capture success
3: OK: Player1- Capture success
1: Init Ready

Zmiana piona na damkę

- zgodnie z przyjętymi zasadami w sytuacji kiedy pion, który dojdzie do ostatniego rzędu planszy, staje się damką, przy czym jeśli znajdzie się tam w wyniku bicia i będzie mógł wykonać kolejne bicie (do tyłu), to będzie musiał je wykonać i nie staje się wtedy damką. Aplikacja zapewnia taki scenariusz.

Możemy zauważyć to na poniższym zrzucie ekranu:



logowana jest także odpowiednia informacja:

```
9: OK: Player1- Changed pawn to dame  
8: OK: Player1- Move Success  
7: OK: Player2- Move Success  
6: OK: Player1- Move Success
```

w przypadku kiedy na końcu swojego ruchu gracz zapomni o zmianie swojego piona na damkę również logowana jest informacja dla użytkownika, lecz tym razem jest to informacja o błędzie, a gra zostaje wstrzymana

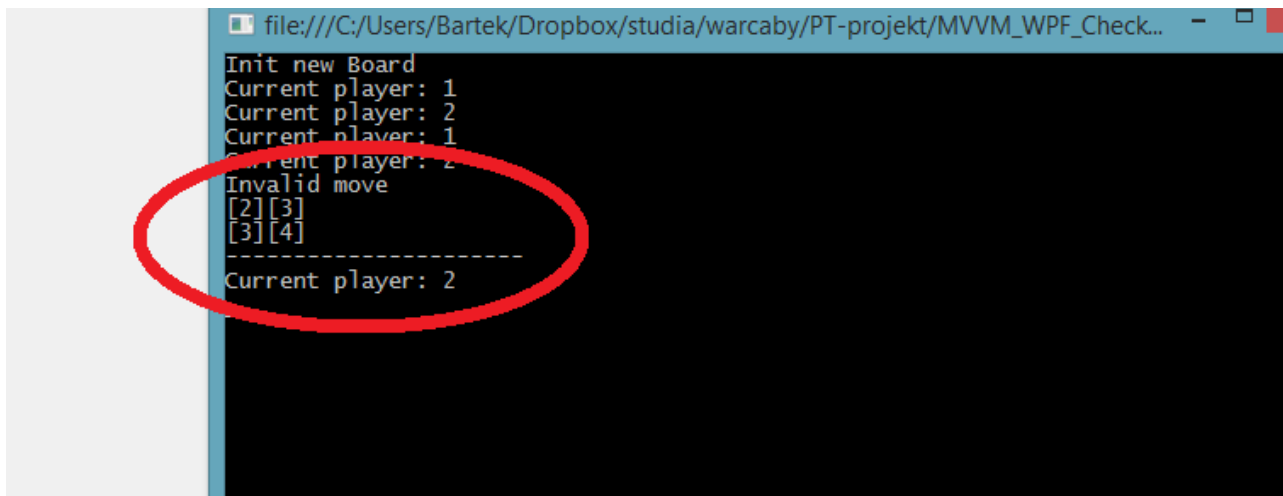
```
3: Error: Player1- Failed Change pawn to dame  
2: OK: Player1- Move Success  
1: Init Ready
```

Logowanie informacji o błędach

- poza informacjami dla użytkownika zaczynających się od słów *Error* lub *OK* logowana jest także informacja do konsoli
- w przypadku kiedy wykonywane są poprawny ruch konsola informuje jedynie o aktualnie poruszającym się graczu

```
file:///C:/Users/Bartek/Dropbox/studia/warcaby/PT-projekt/MVVM_W  
Init new Board  
Current player: 1  
Current player: 2  
Current player: 1  
Current player: 2
```

- natomiast w przypadku kiedy zostaje wykonany niepoprawny ruch logowane są także informacje w jakich miejscach na szachownicy doszło do zmiany, oraz informacja o typie błędu:



```
file:///C:/Users/Bartek/Dropbox/studia/warcaby/PT-projekt/MVVM_WPF_Check...
Init new Board
Current player: 1
Current player: 2
Current player: 1
Current player: 2
Invalid move
[2][3]
[3][4]
-----
Current player: 2
```

2.2.3 Implementacja logiki

- Tak jak zostało to wcześniej wspomniane logika jest zaimplementowana jako osobny projekt typu ClassLibrary.
- Główną klasą odpowiedzialną za zbieranie danych oraz zwracanie komunikatów jest klasa CheckCheckers.
- W jej wnętrzu możemy wyróżnić między innymi następujące metody:

```
public void InitBoard(FieldState[,] boardArray)
{
    _gameState = new GameState(boardArray);
    Console.WriteLine("Init new Board");
    Console.WriteLine("Current player: {0}", _gameState.CurrentPlayer + 1);
}
```

Ustawia ona początkowy obiekt `_gameState` odpowiedzialny za przechowywanie wszystkich informacji o stanie gry. Loguje także informacje na konsoli

```
public string UpdateAndValidBoard(FieldState[,] boardArray)
{
    if (!BoardIsNew(boardArray))
        return String.Empty;

    if (IsError && GetDiff(boardArray, _gameState.PreviousBoardArray).Count == 0)
    {
        IsError = false;
        _gameState.Restore();
        return "Now is OK";
    }
}
```

```

    }

    if (!IsError)
        UpdateBoard(boardArray);

    return Message;
}

```

Funkcja ta jest jedną z najczęściej wykorzystywanych. Służy do zwracania odpowiednich informacji oraz do aktualizacji stanu szachownicy reprezentowanej przez poniższą tablicę

```
private FieldState[,] _boardArray;
```

Struktura znajdująca się w tablicy prezentuje się następująco

```

public enum FieldState
{
    Empty,
    RedPawn,
    YellowPawn,
    BluePawn,
    GreenPawn
}

```

Przedstawia ona wszystkie stany jakie może przyjąć każde z pól na szachownicy

- jako przykład jednej z podstawowych funkcji odpowiedzialnych za sprawdzanie wykonanego ruchu może posłużyć poniższa:

```

private static void CapturePawn(FieldState[,] boardState, int currentX, int
currentY, int moveX, int moveY, GameState gameState, FieldState currentPawnType)
{
    var opponentPawnX = currentX + moveX;
    var opponentPawnY = currentY + moveY;
    var opponentPawn = gameState.CurrentPawn == FieldState.RedPawn ?
FieldState.YellowPawn : FieldState.RedPawn;
    var opponentDame = gameState.CurrentDame == FieldState.BluePawn ?
FieldState.GreenPawn : FieldState.BluePawn;
    var newX = currentX + (moveX > 0 ? moveX + 1 : moveX - 1);
    var newY = currentY + (moveY > 0 ? moveY + 1 : moveY - 1);

    if ((boardState[currentX, currentY] != currentPawnType) ||
        (opponentPawnX < 0 || opponentPawnX > 7 || opponentPawnY < 0 ||
opponentPawnY > 7) ||
        (newX < 0 || newX > 7 || newY < 0 || newY > 7) ||
        !((boardState[opponentPawnX, opponentPawnY] == opponentPawn) ||
(boardState[opponentPawnX, opponentPawnY] == opponentDame)) ||
        (boardState[newX, newY] != FieldState.Empty))
        return;

    var tmpBoard = (FieldState[,])boardState.Clone();
    tmpBoard[newX, newY] = tmpBoard[currentX, currentY];
    tmpBoard[currentX, currentY] = FieldState.Empty;
    tmpBoard[opponentPawnX, opponentPawnY] = FieldState.Empty;
    gameState.PossibleCapture.Add(tmpBoard);
}

```

Można zauważyć, że wykonywane ruchy są sprawdzane w taki sposób, że logika zapamiętuje dwa stany szachownicy, aktualny oraz poprzedni poprawny.

Funkcje sprawdzające ruch porównują te dwie tablice w poszukiwaniu zmian, następnie jeśli zmiana została wykryta sprawdzają czy podana tablica, która chce być zaakceptowana jako poprawna znajduje się w jednej z list

```
public List<FieldState[,]> PossibleMoves;  
public List<FieldState[,]> PossibleCapture;
```

Reprezentują one wszystkie możliwe ruchy oraz bicia jakie może wykonać aktualnie poruszający się gracz. Z powodu obowiązkowych bić lista PossibleMoves sprawdzana jest tylko w wypadku jeśli lista PossibleCapture jest pusta. Zapewnia to wykonanie obowiązkowego bicia. Jeśli stan szachownicy nie zostanie znaleziony w żadnej z list wtedy zwracany jest odpowiedni błąd.

Poniżej funkcja odpowiedzialna za dobór komunikatów:

```
private void Validete()  
{  
    _playerCapture = false;  
    if (!MoveHelper.ChangedPawns(_gameState.BoardArray,  
_gameState.PreviousBoardArray))  
    {  
        IsError = true;  
        const string message = "Failed Change pawn to dame";  
        SetMessage(message);  
        ConsoleHelper.ShowBoardChanges(message, GetDiff(_gameState.BoardArray,  
_gameState.PreviousBoardArray));  
        return;  
    }  
    if (MoveHelper.ChangePawnToDame(_gameState.PreviousBoardArray) &&  
MoveHelper.ChangedPawns(_gameState.BoardArray, _gameState.PreviousBoardArray))  
    {  
        IsError = false;  
        SetMessage("Changed pawn to dame");  
        return;  
    }  
    if (_gameState.PossibleCapture.Any(capture => GetDiff(_gameState.BoardArray,  
capture).Count == 0))  
    {  
        IsError = false;  
        SetMessage("Capture success");  
        _playerCapture = true;  
        return;  
    }  
    if (_gameState.PossibleCapture.Any())  
    {  
        IsError = true;  
        const string message = "Obligatory Capture";  
        SetMessage("Obligatory Capture");  
        ConsoleHelper.ShowBoardChanges(message, GetDiff(_gameState.BoardArray,  
_gameState.PreviousBoardArray));  
        return;  
    }  
    if (_gameState.PossibleMoves.Any(move => GetDiff(_gameState.BoardArray,  
move).Count == 0))  
    {
```



```
        IsError = false;
        SetMessage("Move Success");
        return;
    }
    IsError = true;
    SetMessage("Invalid move");
    ConsoleHelper.ShowBoardChanges("Invalid move", GetDiff(_gameState.BoardArray,
_gameState.PreviousBoardArray));
}
```

3 Wnioski

Jak się okazało z powodu konieczności implementacji takich mechanizmów jak obowiązkowe bicie czy zmiana piona na damkę implementacja gry zajęła więcej czasu niż mógłbym się tego spodziewać. Mimo to jestem zadowolony z efektów prac. Pozostali koledzy w grupie wykonali swoje zadania rzetelnie. Dzięki czemu z łatwością i bez problemów mogliśmy połączyć efekty naszych prac.