**פרוייקט גמר**

הנושא: **משחק רברסי(revesi)**

**המגיש: רועי גיאת**

**המנחה: מיכאל צ'רנובילסקי**

**מאי 2018 תשע"ח**

# תוכן עניינים

[תוכן עניינים 2](#_Toc481448555)

[מבוא 3](#_Toc481448556)

[מטרה 3](#_Toc481448557)

[תיאור המערכת 3](#_Toc481448558)

[שפת התכנות ופירוט סביבת העבודה והכלים 3](#_Toc481448559)

[מפרטי תוכנה 4](#_Toc481448560)

[תיאור כללי 4](#_Toc481448561)

[מהלך המשחק 4](#_Toc481448562)

[התנהלות המשתמש במערכת 4](#_Toc481448563)

[ניסוח וניתוח הבעיה האלגוריתמית 5](#_Toc481448565)

[פיתוח הפתרון ויישומו 6](#_Toc481448566)

[תיאור אלגוריתמים 8](#_Toc481448567)

[מבנה נתונים 10](#_Toc481448568)

[תכנון 11](#_Toc481448569)

[חלוקה למודולים 11](#_Toc481448570)

[רשימת הנתונים 15](#_Toc481448571)

[ביבליוגרפיה 19](#_Toc481448573)

[קוד התוכנית 20](#_Toc481448575)

[GameForm 20](#_Toc481448577)

[Board 21](#_Toc481448578)

[Computer 23](#_Toc481448579)

[Move](#_Toc481448580) 26

[Piece 27](#_Toc481448581)

[Player 28](#_Toc481448582)

# **מבוא**

## מטרה

לבחון את הידע שברשותי שצברתי במהלך השלוש שנים האחרונות במגמת מחשבים.

## תיאור המערכת

המערכת הינה משחק בשם "רברסי", משחק אשר משוחק על לוח בסדר גודל של 8 שורות על 8 עמודות.

המשחק כולל 2 שחקנים אשר משחקים אחד נגד השני בצורה כזו שכל משתתף מניח בתורו דיסקית אחת שצבעה הוא הצבע המזוהה עמו. כאשר הוא מניח דיסקית זו, כל הדיסקיות מהצבע הנגדי הנמצאות בין דיסקית זו לבין דיסקית נוספת מהצבע שלו הופכות גם כן לצבע שמזוהה עם השחקן.

תזוזה:

* הצבת דיסקית תקינה רק כאשר הדיסקית תוחמת דיסקית אחת או יותר של השחקן הנגדי.

אפשרויות משחק:

1. שחקן אנושי נגד שחקן אנושי.

## שפת התכנות ופירוט סביבת העבודה והכלים

פיתוח המערכת נעשה בשפת התכנות C#.  
סביבת העבודה היאFramework - Microsoft .NET ,  
תוך כדי שימוש ב- Microsoft Visual Studio 2017.

פיתוח המערכת נעשה תוך כדי שימוש בכלי Windows Application שסביבת העבודה מציעה למשתמש.

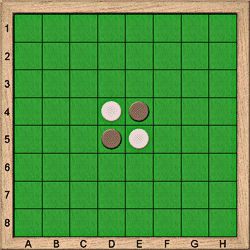
# מפרטי תוכנה

## תיאור כללי

המשחק רברסי הוא משחק חשיבה המשוחק על לוח ריבועים בסדר גודל 8x8.

המצב ההתחלתי הוא כאשר ישנם 2 כלים שחורים ו2 כלים לבנים הממוקמים במרכז הלוח.

הראשון שמשחק הוא בעל הכלים השחורים.

 (מצב התחלתי).

### מהלך המשחק

במהלך המשחק כל שחקן בתורו מזיז את אחד מכליו תוך שמירה על החוקים:

1. תנועה יכולה להתבצעה לכל כיוון כאשר שחקן מניח דיסקית התוחמת דיסקית אחת או יותר של השחקן היריב עם דיסקית ישנה של אותו השחקן.
2. כאשר על משבצת היעד מונח כלי יריב, ומעבר לו יש מקום פנוי, התנועה תהיה מעין דילוג של הכלי למשבצת הריקה, תוך "אכילה" (החלפה) של כלי היריב לצבע של השחקן האוכל.
3. ניצחון מושג אם מתקיים אחד מהבאים:
   1. כל הכלים על הלוח הם באותו הצבע(גם אם הלוח לא מלא) השחקן בצבע הזה מנצח.
   2. כאשר כל הלוח מתמלא או כשאין מהלכים אפשריים נוספים לאף אחד מהצדדים, המנצח הוא מי שיש יותר עיגולים בצבע שלו.
4. קיימת דרך אחת להשגת תיקו:
   1. אם בסיום המשחק מספר הכלים השחורים שווה למספר הכלים הלבנים.

### התנהלות המשתמש במערכת

כאשר מגיע תורו של השחקן הוא בוחר במשצבת בה הוא רוצה להניח את הכלי שלו.

* הבחירה מתבצעת על ידי לחיצה על הלחצן השמאלי בעכבר.
* אפשרויות האכילה תהיינה מוצגות על ידי מסגרת צהובה.

## ניסוח וניתוח הבעיה האלגוריתמית

* בעיה מספר 1 - ביצוע מהלך

במשחק "רברסי" יש בכל מהלך מספר אפשרויות אכילה, כל שחקן בתורו צריך להזיז את אחד מכליו לאחת המשבצות שמוגדרות כמהלך חוקי (כלומר - בתוך אפשרויות האכילה). המערכת צריכה לבדוק האם המהלך שהשחקן מעוניין לבצע חוקי או לא, במידה והמהלך חוקי אז המערכת צריכה לבצע אותו ואם הוא אינו חוקי היא צריכה לתת לו להכניס מהלך חדש(עד שיכניס מהלך חוקי).

* בעיה מספר 2 – אלגוריתם של שחקן ממוחשב

המשחק בפרוייקט שלי, רברסי, משוחק בצורה של שחקן אנושי נגד שחקן ממוחשב, כלומר שחקן שעושה מהלכים לבד ואינו מקבל קלט מהמשתמש. השחקן ההממוחשב מחליט על מהלכיו לפי אלגוריתם שבעזרתו הוא מחשב את המהלך הטוב ביותר עבורו מספר צעדים קדימה שהוגדרו מראש (שלושה צעדים קדימה) (בתוך אלגוריתם זה הוא מתחשב גם במהלך הטוב ביותר של השחקן הנגדי ואז יכול פעול בהתאם).

## פיתוח הפתרון ויישומו

1. ביצוע מהלך

במשחק "רברסי" ישנה אפשרות אכילה אחת ויחידה והיא כאשר הנחת הדיסקית החדשה תוחמת דיסקית אחת או יותר של השחקו השני עם דיסקית ישנה של אותו השחקן שתורו לבצע מהלך(האכילה יכולה להתצבעה בכל הכיוונים).

המערכת צריכה למצוא את כל המהלכים האפשריים של אותו השחקן ולהציג לשחקן את האפשרויות אכילה שלו. אם השחקן אינו לוחץ על משצבת שנכללת בתוך אפשרויות האכילה המערכת תתן לשחקן לבחור שוב במהלך עד שמהלכו יהיה חוקי.

אלגוריתם – בדיקת אפשרויות מהלך:  
האלגוריתם מוצא מהלכים אפשריים בצורה הבאה, הוא בודק כל כלי(של אותו השחקן) על הלוח ומסתכל מה יש בכל כיוון של אותו כלי, אם הוא מוצא כלי של השחקן הנגדי באחד מהכיוונים הוא סופר כמה כלים נמצאים באותו הכיוון ובסופו של רצף הכלים של השחקן הנגדי הוא מוסיף לרשימת המהלכים האפשריים את המשצבת הפנויה הראשונה של אותו הכיוון.  
לאחר פעולות אלה המחשב או השחקן יכולים לבחור את התנועה הרצויה.

1. אלגוריתם של שחקן ממוחשב

האלגוריתם של שחקן ממוחשב עובד בצורה הבאה, הוא רץ על רשימת המהלכים האפשריים ומבצע כל מהלך אפשרי, לכל מהלך אפשרי הוא מבצע גם את המהלך(התגובה) הטוב ביותר של השחקן הנגדי כך עד שהוא מגיע לעומק מסוים(שמוגדר מראש). בסוף כל תור האלגוריתם מחשב את הציון של הלוח לפי המהלך שהתבצע. המחשב בוחר את הציון לפי כל רמה, כלומר אם תורו של השחקן הממוחשב הציון הגבוה ביותר(מקסימלי) יבחר ואם תורו של השחקן אז הציון הנמוך ביותר(מינימלי) יבחר, עד שלבסוף האלגוריתם יגיע בדרך זו למהלך הטוב ביותר שהוא יכול לבצע וכך ייעשה.

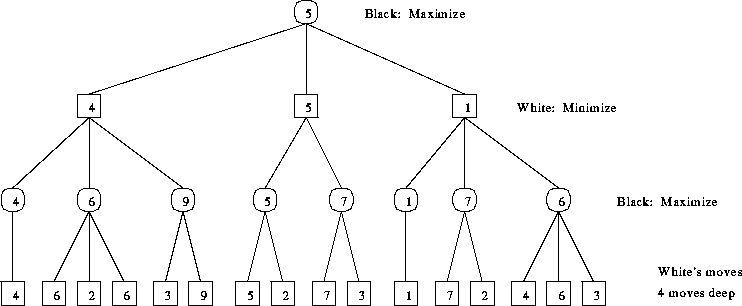
אלגוריתם זה נקרא MiniMax.

עץ MinMax הוא עץ אשר פורס את אפשרויות מהלכי שחקן א' ,ולכל אפשרות מהלך שחקן א' הוא פורס את אפשרויות מהלכי שחקן ב', ולכל אפשרות מהלך של שחקן ב' הוא פורס את את אפשרויות מהלכי שחקן א' וכך האלה, עד רמת עומק מסויימת(הגדרה מויקפדיה).

המשחק איקס-עיגול.

עץ המינימקס(MinMax) עובד בצורה הבאה, בכל עלה בעץ יבחר העלה בעל הציון המקסימלי או המינמלי לפי השחקן שהמהלך הוא עבורו, כלומר אם העלים מייצגים את מהלכי השחקן הממוחשב אז ניקח את הציון המקסימלי ואם הם מייצגים את מהלכי השחקן האנושי ניקח את הציון המינמלי.

בצורה הזאת נוכל לקבל את המהלך הטוב ביותר של השחקן הממוחשב בהנחה כמובן שהשחקן גם משחק בצורה הטובה ביותר עבורו.



(דוגמא לעץ מינימקס)

## 

## תיאור אלגוריתמים

תאור אלגוריתם מספר **1: ביצוע מהלך**

האלגוריתם אוסף את כל המהלכים האפשריים לפי חוקי המשחק ונותן לשחקן לבחור באחד ממהלכים אלה.

**ביצוע מהלך()**

1. קבלת כל המהלכים האפשריים מפונקציה חיצונית.
2. מציאת הדיסקית השנייה התוחמת את הדיסקיות של השחקן היריב עם הדיסקית מהמהלכים האפשריים.
3. עבור אינדס שווה לאפס עד אינדקס קטן ממספר הכלים התחומים בין 2 הדיסקיות של השחקן שתורו לשחק.
   1. מחיקת הדיסקית הנמצאת באותו משצבת מרשימת הכלים הנמצאים על הלוח של היריב.
   2. הוספת הדיסקית אל רשימת הכלים הנמצאים על הלוח של השחקן שתורו תוך החלפת הצבע לצבע הדיסקית שמזוהה עם השחקן שתורו.
   3. התקדמות למשבצת הבאה באותו הכיוון.
4. הוספת הדיסקית(המהלך) אל הלוח.

תאור אלגוריתם מספר 2**: אלגוריתם לשחקן ממוחשב**

האלגורתים בונה עץ משחק שבעזרתו הוא מחשב את המהלך הטוב ביותר עבור השחקן הממוחשב 3 צעדים קדימה

**מינימקס(עומק)**

1. אם עומק שווה לאפס
   1. תחזיר ציון לוח
2. אם עומק אי זוגי אז תורו של השחקן הממוחשב אחרת תורו של השחקן האנושי
3. קבל את כל המהלכים האפשריים
4. הגדרת המשתנים מקסימום ומינימום
5. לכל מהלך בתוך מהלכים אפשריים
   1. אם המהלך נמצא בתוך תחומי הלוח(שורה, עמודה)
      1. בצע מהלך(מהלך, שחקן שתורו)
      2. מינימקס(1 - עומק)
      3. אם עומק אי זוגי
         1. אם הציון החדש על הלוח גדול יותר מהציון המקסימלי עד כה
            1. הציון המקסימלי שווה לציון החדש
            2. המהלך הטוב ביותר שווה למהלך ששוחק עכשיו
      4. אם עומק זוגי
         1. אם הציון החדש על הלוח נמוך יותר מהציון המינמלי עד כה
            1. הציון המקסימלי שווה לציון החדש
            2. המהלך הטוב ביותר שווה למהלך ששוחק עכשיו
      5. מחק מהלך
6. אם עומק זוגי תחזיר ציון מקסימלי אחרת תחזיר ציון מינמלי

## 

## מבנה נתונים

**מטריצה (matrix)**

במהלך הפרויקט עשיתי שימוש במטריצה. לדוגמא כאשר עשיתי מטריצה שמכילה את כל הכיוונים האפשרים שאפשר לנוע אליהם על הלוח.

**רשימה (list)**

במהלך הפרויקט עשיתי שימוש ברשימה. לדוגמא, כאשר עשיתי רשימה שמכילה את כל המהלכים האפשריים של השחקן שתורו לשחק.

**מילון (dictionary)**

במהלך הפרוייקט עשיתי שימוש במילון, לדוגמא, כאשר עשיתי מילון שמכיל את השורה, הטור והצבע של כל כלי על הלוח.

# תכנון

## חלוקה למודולים

Gameform

|  |  |
| --- | --- |
| שם הפעולה | תיעוד הפעולה |
| public GameForm() | בנאי המחלקה. יוצר טופס מטיפוס GameForm |
| private void pictureBox1\_MouseMove(object sender, MouseEventArgs e) | פעולת אירוע, מתבצעת כאשר המשתמש מעביר את העכבר מעל הלוח. הפעולה מסמנת, אם צריך, את הכלי המסומן על ידי העכבר. |
| private void pictureBoxBox1\_MouseClick(object sender, MouseEventArgs e) | פעולת אירוע, מתבצעת כאשר השחקן לוחץ על הלוח וחלקיו, הפעולה מבצעת מהלך בהתאם לבקשת המשתמש. |
| private void pictureBox1\_Paint(object sender, PaintEventArgs e) | פעולת אירוע, מתבצעת כאשר הלוח מתעדכן. הפעולה צובעת את הלוח. |

Board

|  |  |
| --- | --- |
| שם הפעולה | תיעוד הפעולה |
| public Board(GameForm gameForm) | בנאי המחלקה. יוצר טופס מטיפוס Board |
| internal void Paint(System.Drawing.Graphics graphics) | הפעולה מקבלת את הגרפיקה של הלוח, ומציירת, על הלוח, את הכלים בהתאם למטריצה |
| internal void Click(System.Drawing.Point point) | הפעולה מקבלת מיקום על הלוח בו השחקן לחץ על העכבר, ובודקת האם מהלך זה חוקי או לא(אם כן מוסיפה אותו ללוח אם לא אז לא) בתנאי שהמשחק אינו נגמר. |
| private bool Endgame() | פעולה בוליאנית שבודקת האם המשחק נגמר או לא ומחזירה תשובה בהתאם. |

Computer

|  |  |
| --- | --- |
| שם הפעולה | תיעוד הפעולה |
| public Computer(Board board, TypePiece typePiece)  : base(board, typePiece) | בנאי המחלקה. יוצר טופס מטיפוס Computer |
| private int MiniMax(int level) | הפעולה מקבלת "עומק" ומחשבת את המהלך הטוב ביותר כמה מהלכים (ששווה לעומק) קדימה |
| public void DoStep() | הפעולה קוראת לפונקציה CompStep בThread |
| private int Evaluate() | הפעולה מחשבת ציון של מהלך לפי הכלים הנמצאים על הלוח |
| private void UnDoMove(Move move, Player player) | הפעולה מקבלת מהלך ושחקן מהטיפוסים "Move" ו-"Player" בהתאמה ומוחקת את אותו המהלך של השחקן |
| private void DoMove(Move move, Player player) | הפעולה מקבלת מהלך ושחקן מהטיפוסים "Move" ו-"Player" בהתאמה ומוסיפה את אותו המהלך של השחקן. |
| private bool Endgame() | פעולה בוליאנית שבודקת האם המשחק נגמר או לא ומחזירה תשובה בהתאם. |
| private bool IsLegal(int newrow, int newcol) | פעולה בוליאנית המקבלת שורה ועמודה ובודקת האם המהלך החדש שרוצים לבצע נמצא בתוך הלוח המוגדר ומחזירה תשובה בהתאם. |
| private void CompStep() | הפעולה מוסיפה את המהלך הטוב ביותר לרשימה של הכלים ומשנה את הלוח בהתאם בנוסף היא מציגה את כמות הכלים בצבע הלבן וכמות הכלים בצבע השחור בכל תור |

Move

|  |  |
| --- | --- |
| שם הפעולה | תיעוד הפעולה |
| public Move(Piece source, Point dest, int dir, int count, Player player) | בנאי המחלקה. יוצר טופס מטיפוס Move |

Piece

|  |  |
| --- | --- |
| שם הפעולה | תיעוד הפעולה |
| public Piece(int row, int col, TypePiece typePiece) | בנאי המחלקה. יוצר טופס מטיפוס Piece |
| internal void Paint(System.Drawing.Graphics graphics) | הפעולה מקבלת את הגרפיקה של הלוח, ומציירת, על הלוח, את הכלים בהתאם למטריצה |

Player

|  |  |
| --- | --- |
| שם הפעולה | תיעוד הפעולה |
| public Player(Board board, TypePiece typePiece) | בנאי המחלקה. יוצר טופס מטיפוס Player |
| internal void Paint(System.Drawing.Graphics graphics) | הפעולה מקבלת את הגרפיקה של הלוח, ומציירת, על הלוח, את הכלים בהתאם למטריצה |
| internal bool Check(int row, int col) | פעולה בוליאנית המקבלת שורה ועמודה ומחזירה אמת אם המהלך שהשחקן רוצה לבצע הוא חוקי שקר אחרת. אם המהלך חוקי הפעולה משנה את הלוח בהתאם למהלך. |
| public int CheckDir(int row, int col, int i,Player player) | פעולה המקבלת שורה, עמודה, כיוון ושחקן וסופרת כמה כלים נאכלו במהלך המתבקש להתבצע. |
| private bool IsLegal(int newrow, int newcol) | פעולה בוליאנית המקבלת שורה ועמודה ובודקת האם המהלך הוא בתוך הלוח המוגדר מראש ומחזירה תשובה בהתאם. |
| internal void Add(int row, int col) | פעולה המקבלת שורה ועמודה ומוסיפה את אותו המהלך לרשימת הכלים מסוג Piece |
| internal void Remove(int row, int col) | פעולה המקבלת שורה ועמודה ומוחקת את אותו המהלך מרשימת הכלים מסוג Piece |
| public List<Move> GetPossibleMoves() | פעולה המוסיפה את כל המהלכים האפשריים של שחקן באותו התור |
| internal bool NotExist(int row, int col) | פעולה המקבלת שורה ועמודה ובודקת האם המהלך הזה נמצא על הלוח |
| internal int GetGrade() | פעולה המחשבת את הציון של כל הכלים הנמצאים על הלוח |

## רשימת הנתונים

Board

|  |  |
| --- | --- |
| **שם הנתון** | **פירוט הנתון** |

|  |  |
| --- | --- |
| public static int N = 8; | משתנה סטטי המייצג את מספר השורות/עמודות בלוח |
| protected List<Move> possibleMoves; | רשימה מסוג "move" המייצגת את המהלכים האפשריים של כל שחקן |
| int PlayerWins = 0; | משתנה המייצג את מספר הניצחונות של השחקן |
| int ComputerWins = 0; | משתנה המייצג את מספר הניצחונות של המחשב |
| public GameForm gameForm; | משתנה המייצג את הטופס בו נמצא הלוח |
| private Player player1; | משתנה המייצג את השחקן הראשון |
| private Player player2; | משתנה המייצג את השחקן השני |
| int CountPlayer = 0; | משתנה המייצג את מספר הכלים(השחורים) של השחקן |
| int CountComputer = 0; | משתנה המייצג את מספר הכלים(הלבנים) של המחשב |
| Rectangle rect; | משתנה מסוג " Rectangle " המייצג מלב |

Computer

|  |  |
| --- | --- |
| private Move bestMove; | משתנה המייצג את המהלך הטוב ביותר בכל תור |
| int CountPlayer = 0; | משתנה המייצג את מספר הכלים(השחורים) של השחקן |
| int CountComputer = 0; | משתנה המייצג את מספר הכלים(הלבנים) של המחשב |

Move

|  |  |
| --- | --- |
| public Piece source; |  |
| public Point dest; |  |
| public int dir; | משתנה המייצג את כיוון האכילה |
| public int count; | משתנה המייצג כמה כלים צריכים להאכל |
| public Player player; | משתנה המייצג את השחקן שתורו לשחק |

Piece

|  |  |
| --- | --- |
| public static int PieceSize = 40; | משתנה המייצג את גודל הכלי שמונח על המשבצת |
| public int row; | משתנה המייצג שורה על הלוח |
| public int col; | משתנה המייצג עמודה על הלוח |
| public TypePiece typePiece; | משתנה המייצג צבע כלי |

Player

|  |  |
| --- | --- |
| public Dictionary<int, Piece> pieces; | מילון המייצג את כל הכלים על הלוח |
| protected List<Move> possibleMoves;  ; | רשימה מסוג "move" המייצגת את המהלכים האפשריים של כל שחקן |
| private TypePiece typePiece; | משתנה המייצג צבע כלי |
| protected Board board; | משתנה מסוג "Board" המייצג לוח |
| protected Player enemy; | משתנה המייצג את השחקן הנגדי(זה שאינו תורו) |
| public int[,] Dir; | מטריצה המייצגת את כל הכיוונים האפשרים על הלוח |

Gameform

|  |  |
| --- | --- |
| Board board; | משתנה לוח מסוג "Board" |

# ביבליוגרפיה

אתרי אינטרנט:

* [https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/](about:blank)
* [https://he.wikipedia.org/wiki/%D7%A2%D7%A5\_%D7%9E%D7%99%D7%A0%D7%99%D7%9E%D7%A7%D7%A1](about:blank)

# קוד התכנית

Gameform

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

namespace MyOtGame

{

public partial class GameForm : Form

{

Board board;

public GameForm()

{

InitializeComponent();

board = new Board(this);

toolStripStatusLabel1.Text = "Start game";

}

private void pictureBox1\_Paint(object sender, PaintEventArgs e)

{

board.Paint(e.Graphics);

}

private void pictureBox1\_MouseClick(object sender, MouseEventArgs e)

{

board.Click(e.Location);

pictureBox1.Invalidate();

}

private void pictureBox1\_MouseMove(object sender, MouseEventArgs e)

{

pictureBox1.Invalidate();

}

}

}

Board

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Drawing;

using System.Threading;

using MyOtGame.GUI;

namespace MyOtGame

{

class Board

{

int PlayerWins = 0;

int ComputerWins = 0;

int CountPlayer = 0;

int CountComputer = 0;

public static int N = 8;

public Player player1;

public Player player2;

// int turn = 0;

public static GameForm gameForm;

//public static Menu menu;

protected List<Move> possibleMoves;

Rectangle rect;

public Board(GameForm form)

{

gameForm = form;

player1 = new Player(this, TypePiece.BLACK);

player2 = new Computer(this, TypePiece.WHITE);

}

internal void Paint(System.Drawing.Graphics graphics) //paint

{

possibleMoves = player1.GetPossibleMoves();

foreach (Move move in possibleMoves)

{

rect = new Rectangle(move.dest.X \* Piece.PieceSize, move.dest.Y\* Piece.PieceSize, Piece.PieceSize-4, Piece.PieceSize-4);

graphics.FillRectangle(new SolidBrush(Color.Gold), rect);

}

player1.Paint(graphics);

player2.Paint(graphics);

}

internal void Click(System.Drawing.Point point)

{

int row = point.Y / Piece.PieceSize;

int col = point.X / Piece.PieceSize;

// Player player = turn % 2 == 0 ? player2 : player1;

if (Endgame())

{

string winner = "";

if (CountComputer > CountPlayer)

{

ComputerWins++;

winner = "COMPUTER WIN";

}

else if (CountComputer < CountPlayer)

{

PlayerWins++;

winner = "YOU WIN!!";

}

else

{

winner = "DRAW";

PlayerWins++;

ComputerWins++;

}

Board.gameForm.toolStripStatusLabel1.Text = winner;

Thread.Sleep(1000);

}

else

{

if (player1.Check(row, col))

{

player1.Add(row, col);

(player2 as Computer).DoStep();

}

}

}

private bool Endgame() //check if game over

{

CountPlayer = player1.pieces.Values.Count; CountComputer = player2.pieces.Values.Count;

return ((CountPlayer + CountComputer) == 64 || CountPlayer == 0 || CountComputer == 0);

}

}

}

Computer

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

namespace MyOtGame

{

class Computer : Player

{

private Move bestMove;

int CountPlayer = 0;

int CountComputer = 0;

public Computer(Board board, TypePiece typePiece)

: base(board, typePiece)

{

}

private int MiniMax(int level) //take the best move for computer

{

if (level == 0)

return Evaluate();

Player player = level % 2 != 0 ? board.player2 : board.player1;

possibleMoves = GetPossibleMoves();

int max = int.MinValue, min = int.MaxValue;

foreach (Move move in possibleMoves)

{

if (IsLegal(move.dest.Y, move.dest.X))

{

DoMove(move, player);

int grade = MiniMax(level - 1);

if (level % 2 != 0)

{

if (grade > max)

{

max = grade;

bestMove = move;

}

}

else

{

if (grade < min)

{

min = grade;

bestMove = move;

}

}

UnDoMove(move, player);

}

}

return level % 2 != 0 ? max: min;

}

public void DoStep()

{

Thread compStep = new Thread(new ThreadStart(CompStep));

compStep.Start();

}

private int Evaluate()

{

int grade = board.player2.GetGrade() - board.player1.GetGrade();

return grade \* 2 + (board.player2.pieces.Count - board.player1.pieces.Count);

}

private void UnDoMove(Move move, Player player) //delete the move from "DoMove" function

{

enemy = (player is Computer) ? board.player1 : board.player2;

int newrow = move.source.row + Dir[move.dir, 0];

int newcol = move.source.col + Dir[move.dir, 1];

for (int j = 0; j < move.count; j++)

{

int key\_enemy = newrow \* Board.N + newcol;

this.pieces.Remove(key\_enemy);

enemy.pieces.Add(key\_enemy, new Piece(newrow, newcol, player == this ? TypePiece.BLACK : TypePiece.WHITE));

newrow += Dir[move.dir, 0];

newcol += Dir[move.dir, 1];

}

this.Remove(move.dest.Y, move.dest.X);

}

private void DoMove(Move move, Player player) //add new pawn and delete the enemy pawn

{

enemy = player is Computer ? board.player1 : board.player2;

int newrow = move.source.row + Dir[move.dir, 0];

int newcol = move.source.col + Dir[move.dir, 1];

for (int j = 0; j < move.count; j++)

{

int key\_enemy = newrow \* Board.N + newcol;

enemy.pieces.Remove(key\_enemy);

this.pieces.Add(key\_enemy, new Piece(newrow, newcol, player == this ? TypePiece.BLACK : TypePiece.WHITE));

newrow += Dir[move.dir, 0];

newcol += Dir[move.dir, 1];

}

this.Add(move.dest.Y, move.dest.X);

}

private bool Endgame() //check if game over

{

CountPlayer = enemy.pieces.Values.Count; CountComputer = pieces.Values.Count;

return ((CountPlayer + CountComputer) == 64 || CountPlayer == 0 || CountComputer == 0);

}

private bool IsLegal(int newrow, int newcol)

{

return newrow >= 0 && newrow < Board.N && newcol >= 0 && newcol < Board.N;

}

private void CompStep() // add best move to piece

{

MiniMax(1);

string dot = ".";

for (int i = 1; i < 4; i++)

{

Board.gameForm.toolStripStatusLabel1.Text = "Computer think" + dot;

Thread.Sleep(500);

dot += ".";

}

Check(bestMove.dest.Y, bestMove.dest.X);

Add(bestMove.dest.Y, bestMove.dest.X);

CountPlayer = enemy.pieces.Values.Count; CountComputer = pieces.Values.Count;

Board.gameForm.toolStripStatusLabel1.Text = "Computer : " + CountComputer + " You : " + CountPlayer;

Board.gameForm.pictureBox1.Invalidate();

}

}

}

Move

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Drawing;

namespace MyOtGame

{

class Move

{

public Piece source;

public Point dest;

public int dir;

public int count;

public Player player;

public Move(Piece source, Point dest, int dir, int count, Player player)

{

this.source = source;

this.dest = dest;

this.dir = dir;

this.count = count;

this.player = player;

}

}

}

Piece

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace MyOtGame

{

public enum TypePiece { WHITE, BLACK,POSSIABLE };

class Piece

{

public static int PieceSize = 40;

public int row;

public int col;

public TypePiece typePiece;

public Piece(int row, int col, TypePiece typePiece)

{

// TODO: Complete member initialization

this.row = row;

this.col = col;

this.typePiece = typePiece;

}

internal void Paint(System.Drawing.Graphics graphics)

{

switch (typePiece)

{

case TypePiece.BLACK:

graphics.DrawImage(Properties.Resources.Pawn\_Black, col \* PieceSize, row \* PieceSize, PieceSize, PieceSize);

break;

case TypePiece.WHITE:

graphics.DrawImage(Properties.Resources.Pawn\_White, col \* PieceSize, row \* PieceSize, PieceSize, PieceSize);

break;

}

}

}

}

Player

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Drawing;

namespace MyOtGame

{

class Player

{

public Dictionary<int, Piece> pieces;

protected List<Move> possibleMoves;

private TypePiece typePiece;

protected Board board;

protected Player enemy;

public int[,] Dir =

{

{-1,-1}, //UpLeft

{-1,0}, //Up

{-1,1}, //UpRight

{0,-1}, //Left

{0,1}, //Right

{1,-1}, //DownLeft

{1,0}, //Down

{1,1} //DownRight

};

public Player(Board board, TypePiece typePiece)

{

this.board = board;

this.typePiece = typePiece;

enemy = board.player1 == this ? board.player2 : board.player1;

pieces = new Dictionary<int, Piece>();

if (typePiece == TypePiece.BLACK)

{

pieces.Add(3 \* Board.N + 3, new Piece(3, 3, TypePiece.BLACK)); //starting position

pieces.Add(4 \* Board.N + 4, new Piece(4, 4, TypePiece.BLACK)); //starting position

}

else

{

pieces.Add(3 \* Board.N + 4, new Piece(3, 4, TypePiece.WHITE)); //starting position

pieces.Add(4 \* Board.N + 3, new Piece(4, 3, TypePiece.WHITE)); //starting position

}

}

internal void Paint(System.Drawing.Graphics graphics)

{

List<Piece> pieces = this.pieces.Values.ToList();

foreach (Piece piece in pieces)

piece.Paint(graphics);

}

internal bool Check(int row, int col)

{

enemy = board.player1 == this ? board.player2 : board.player1;

bool approval = false;

int index = 0;

while (index != 8) //Get \*all\* possible moves

{

for (int i = 0; i < 8; i++)

{

index++;

int newrow = row + Dir[i, 0];

int newcol = col + Dir[i, 1];

int count = CheckDir(row, col, i, enemy);

if (count > 0)

{

for (int j = 0; j < count; j++)

{

int key\_enemy = newrow \* Board.N + newcol;

enemy.pieces.Remove(key\_enemy); //Deletes enemy pawn

pieces.Add(key\_enemy, new Piece(newrow, newcol, this.typePiece)); //Add new pawn

newrow += Dir[i, 0];

newcol += Dir[i, 1];

}

approval = true;

}

}

}

if (approval)

return true;//A legal move

return false;

}

public int CheckDir(int row, int col, int i,Player player)

{

int count = 0;

int key\_new = 0;

int newrow = row + Dir[i, 0];

int newcol = col + Dir[i, 1];

key\_new = newrow \* Board.N + newcol;

while (IsLegal(newrow, newcol) && player.pieces.ContainsKey(key\_new))

{

newrow += Dir[i, 0];

newcol += Dir[i, 1];

key\_new = newrow \* Board.N + newcol;

count++; //How many pawns eaten

}

if (IsLegal(newrow, newcol) && !NotExist(newrow, newcol) && count > 0)

return count;

return 0;

}

private bool IsLegal(int newrow, int newcol)

{

return newrow >= 0 && newrow < Board.N && newcol >= 0 && newcol < Board.N;

}

internal void Add(int row, int col) // add to pices new pawn

{

int key\_new = row \* Board.N + col;

pieces.Add(key\_new, new Piece(row, col, this.typePiece));

}

internal void Remove(int row, int col) //remove from pices pawn

{

int key\_new = row \* Board.N + col;

pieces.Remove(key\_new);

}

public List<Move> GetPossibleMoves() //get all possible moves

{

enemy = board.player1 == this ? board.player2 : board.player1;

List<Move> possibleMoves = new List<Move>();

foreach (Piece piece in pieces.Values)

{

for (int i = 0; i < 8; i++)

{

bool ok = false;

int newrow = piece.row + Dir[i, 0];

int newcol = piece.col + Dir[i, 1];

int key\_new = newrow \* Board.N + newcol;

int count = 0;

if (!NotExist(piece.row, piece.col))

{

while (IsLegal(newrow, newcol) && enemy.pieces.ContainsKey(key\_new))

{

newrow += Dir[i, 0];

newcol += Dir[i, 1];

key\_new = newrow \* Board.N + newcol;

ok = true;

count++;

}

}

if (ok && NotExist(newrow, newcol))

possibleMoves.Add(new Move(piece, new Point(newcol, newrow), i, count, this));

}

}

return possibleMoves;

}

internal bool NotExist(int row, int col) //Checks that the new move is in the limits of the board

{

int key = row \* Board.N + col;

return !pieces.ContainsKey(key);

}

internal int GetGrade()

{

int[,] mark = {

{ 1000, -100, 100, 100, 100, 100, -100, 1000 },

{ -100, -100, 50, 50, 50, 50, -100, -100 },

{ 100, 50, 80, 80, 80, 80, 50, 100 },

{ 100, 50, 80, 80, 80, 80, 50, 100 },

{ 100, 50, 80, 80, 80, 80, 50, 100 },

{ 100, 50, 80, 80, 80, 80, 50, 100 },

{ -100, -100, 50, 50, 50, 50, -100, -100 },

{ 1000, -100, 100, 100, 100, 100, -100, 1000 }

};

int grade = 0;

foreach (Piece piece in this.pieces.Values)

{

grade += mark[piece.row, piece.col];

}

return grade;

}

}

}