**עבודת גמר לקבלת תואר**

**טכנאי תוכנה**



**הנושא: משחק פנטגו/Pentago**

**המגיש: שקד ברססט**

**מס. ת.ז.: 214997900**

**המנחה: שרה פולק**

**תאריך הגשה: 20/04/2023**

**תוכן עניינים**

[**תקציר** 3](#_Toc133586317)

[**מושגים** 4](#_Toc133586318)

[**תיאור הנושא** 5](#_Toc133586319)

[תכולת המשחק 5](#_Toc133586320)

[מטרת המשחק 5](#_Toc133586321)

[סיום המשחק 5](#_Toc133586322)

[**רקע תיאורטי** 6](#_Toc133586323)

[**תיאור הבעיה האלגוריתמית** 7](#_Toc133586324)

[**סקירת אלגוריתמים בתחום הבעיה** 8](#_Toc133586325)

[**אסטרטגיה נבחרת לפתרון- תיאור האסטרטגיה והנימוק לבחירתה** 10](#_Toc133586326)

[**ארכיטקטורה של הפתרון המוצע בפורמט של Design Level down-Top** 13](#_Toc133586327)

[**תרשים מקרי-שימוש cases Use UML** 14](#_Toc133586328)

[**מבנה נתונים** 15](#_Toc133586329)

[**תיאור סביבת העבודה ושפת התכנות** 17](#_Toc133586330)

[**אלגוריתם ראשי (בפורמט פסיאודו קוד)** 18](#_Toc133586331)

[**תרשים מחלקות- UML Class Diagram** 24](#_Toc133586332)

[**תיאור המחלקות הראשיות בפרויקט** 27](#_Toc133586333)

[**התכנית הראשית** 55](#_Toc133586334)

[**מדריך למשתמש** 56](#_Toc133586335)

[**סיכום אישי / רפלקציה** 60](#_Toc133586336)

[**ביבליוגרפיה** 61](#_Toc133586337)

[**נספח א – קוד הפרויקט** 62](#_Toc133586338)

[מחלקת DemoDriver: 62](#_Toc133586339)

[מחלקת Instructions: 63](#_Toc133586340)

[מחלקת OpeningScreen: 66](#_Toc133586341)

[מחלקת TheBoard: 69](#_Toc133586342)

[מחלקת MiniBoard: 80](#_Toc133586343)

[מחלקת Board: 85](#_Toc133586344)

[מחלקת ControllerClass: 96](#_Toc133586345)

[מחלקת AiPlayer: 102](#_Toc133586346)

# **תקציר**

בניתי משחק פנטגו, משחק לוח אסטרטגי וחשיבתי, לשני שחקנים בלבד. משחק זה מורכב מעשרות מיליארדים של אפשרויות, ומצריך חשיבה מורכבת. המשחק מאפשר מספר מצבים, אשר מאפשר בין היתר שימוש בבינה מלאכותית (AI), כאשר בעת תחילת המשחק ניתן לבחור את אופן פעולת המשחק:

* שחקן אנושי מול שחקן ממוחשב
* שחקן אנושי מול שחקן אנושי

בנוסף, ישנה אפשרות לראות את חוקי המשחק לפני תחילתו.

המשחק ידידותי למשתמש (GUI), כאשר נתן להתחיל משחק, ולסיימו (כאשר שחקן אחד מנצח או תיקו). המשחק מוגדר כמשחק "פתוח"/ידע מלא- בכל רגע נתון במשחק שני השחקנים יודעים את מצב הלוח.

מטרתו של המשחק היא ליצור רצף של חמישה חיילים מהצבע שלך- מתוך הלוח המורכב מ 6x6. הרצף יכול להיות אופקי, מאונך או אלכסוני.

המערכת שפיתחתי משתמשת באלגוריתמים ומבני נתונים יעילים והיא מבוססת תכנות מונחה עצמים בשפת JAVA. כמובן שבחירה של מבנה נתונים מתאים ויעיל תשפיע על זמן הריצה ולכן נרצה לבחור מבנה נתונים אשר מתאים למשחק ויעיל ככל הניתן.

# **מושגים**

**משחק פתוח/ידע מלא**: משחק מסוג ידע מלא הוא משחק אשר הכל "פתוח", ידוע וחשוף לכל השחקנים. כלומר, כל הכלים על הלוח חשופים וידועים, כך שניתן לדעת בדיוק מה המצב הנתון במשחק בכל נקודת זמן.

**בינה מלאכותית – AI:** בינה מלאכותית היא שם דמוי למצב שבו מנסים לדמות את יכולות החשיבה האנושית על ידי שימוש באמצעים טכנולוגיים. למשל, במשחק שבניתי, המחשב מנסה לדמות את החשיבה האנושית וזאת על ידי אלגוריתם שפיתחתי.

# **תיאור הנושא**

## תכולת המשחק

המשחק מורכב מלוח בגודל של 6x6 משבצות, אשר מורכב מארבע מיני לוחות בגודל 3x3 כל אחד. כל מיני לוח ניתן לסיבוב ימינה או שמאלה. בנוסף, מכיל 36 חיילי משחק, כאשר 18 מתוכם בצבע לבן (שחקן א'), ו 18 האחרים בצבע שחור (שחקן ב').

כל משתתף בתורו שם חייל על הלוח בצבע שניתן לו. לאחר מכן, יהיה עליו לסובב את אחד ממיני הלוחות ימינה או שמאלה בתשעים מעלות.

ארבע מיני הלוחות יהיו מסומנים כך:

|  |  |
| --- | --- |
| 2 | 1 |
| 4 | 3 |

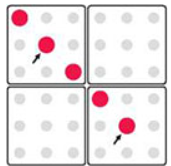
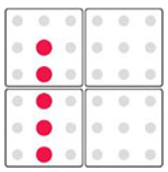
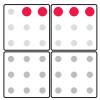
## מטרת המשחק

מטרת המשחק היא לנסות להגיע לחמישה חיילים ברצף במאונך, במאוזן או באלכסון, כל זאת תוך שמירה על חוקי המשחק וניסיון לתקוף את היריב ולמנוע ממנו לנצח.

## סיום המשחק

המשחק מסתיים כאשר אחד השחקנים שם מצליח לעשות רצף של חמישה חיילים באופן אנכי, אופקי או אלכסוני. בנוסף, המשחק מכיל את האפשרות לתיקו (מאוד נדיר), כאשר אף אחד מהשחקנים לא הצליח הגיע לניצחון.

דוגמאות לניצחון במשחק:

# **רקע תיאורטי**

המשחק שפיתחתי נשען על תורת המשחקים ושימוש בבינה מלאכותית לתכנון שחקן ממוחשב. המשחק מורכב מהרבה מאוד מצבים אפשריים, כאשר לבסוף מובילים לניצחון. כל שלב שבו משחקים משפיע על מהלך המשחק כולו, וכל תכנון ששחקן מסוים עושה יכול להשתנות בכל עת (עקב תגובה נגדית).

על השחקן הממוחשב למעשה לבחון את המצבים האפשריים ואת יריבו, בכך שיבדוק אם היריב שמולו יכול לנצחו בתור הבא שיגיע, אם כן ינסה לחסום אותו מכך, ואם לא ינסה אסטרטגיית ניצחון שמתאימה למצב הלוח.

ניתן לממש את השחקן הממחשב במספר דרכים שונות, וביניהן עץ החלטה, מכונת מצבים ועוד. אני בחרתי לעבוד עם מכונת מצבים, אשר לאחר שקלול המצבים תבחר אך הצעד הבא הכי טוב לשחקן הממוחשב (הרחבה בהמשך). זה יקרה באמצעות פונקציית הערכה שתדע לתת הערכה שלכל מהלך שרלוונטי ללוח. למשל, עבור 4 חיילים רצופים היא תחזיר 100, עבור 3 חיילים רצופים תחזיר 70 וכו.

כפי שכבר ציינתי, המשחק נועד לשני שחקנים בלבד, כאשר הוא משוחק בתורות- המהלכים לא מתנהלים בו זמנית, אלא אחד אחרי השני. כך, בכל תור ותור יהיה עלינו לבדוק מספר דברים חשובים:

* שחקן יהיה חייב לסובב מיני-לוח, גם אם הוא ריק.
* שחקן אחד לא יוכל להמשיך במשחק עד אשר השחקן שלפניו סיים לשים חייל וגם לסובב מיני-לוח כרצונו.
* שחקן לא יוכל לשים חייל על חייל של שחקן אחר (לתפוס מקום שכבר תפוס), וכמובן שגם לא יוכל לשים חייל של חייל שלו שכבר קיים.
* בדיקת ניצחון/תיקו.

# **תיאור הבעיה האלגוריתמית**

הפרויקט מכיל 2 בעיות עיקריות:

* הבעיה האלגוריתמית והחשובה ביותר בפרויקט שלי היא **בניית שחקן ממוחשב אשר ידע לשחק בצורה יעילה ולשחק את המהלך הטוב עבורו**. כלומר, בניית פונקציית הערכה טובה.

על מנת שהשחקן יפעל בצורה המיטבית, עלינו לבחור אלגוריתם יעיל, אשר יבדוק את המקרים האפשריים ויחליט את הצעד הבא שלו בצורה מיטבית, כמובן תוך שמירה על ביצועים טובים ללא הגעה לזמן ריצה ארוך מדי. כפי שציינת מקודם, הפרויקט שלי מכיל המון מצבים, ולכן בחירת אלגוריתם מתאים היא מהותית.

האלגוריתם יצטרך להתחשב בשני דברים עיקרים בכל הנוגע לבחירת הצעד הבא- קירוב לניצחון והפסלת היריב.

* בעיה נוספת בה נתקלתי היא **בחירת מבנה נתונים** מתאים בו אשתמש למימוש מבנה לוח המשחק. כיוון שהמשחק חייב להיות יעיל ומהיר, היה עליי לחשוב על מבנה נתונים מתאים. לאחר שבחרתי את מבנה הנתונים – מימוש על ידי סיביות (אפרט בהמשך), היה צורך להתאים אותו לפרויקט, דבר אשר היווה סיבוך למימוש המשחק.

# **סקירת אלגוריתמים בתחום הבעיה**

בעת העבודה על השחקן הממוחשב הפועל במסגרת בינה מלאכותית יש למצוא אלגוריתם מתאים אשר ידע לפעול בצורה יעילה ומיטבית.

כעת אסקור אלגוריתמים אפשריים בתחום זה:

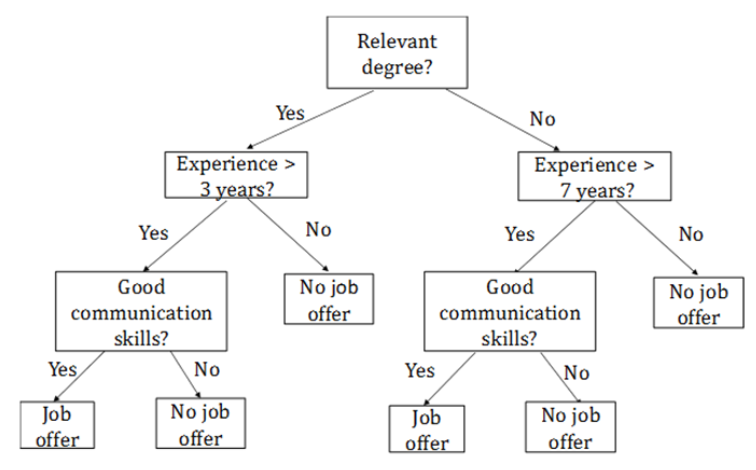
עץ החלטה-

עץ החלטה הוא אלגוריתם חיזוי השייך לתחום הלמידה החישובית, אשר מציג תהליך של שלב-אחר-שלב לביצוע חיזוי מסוים.

עץ החלטה ממפה תצפיות על פריט ויוצר מסקנות על ערך היעד של הפריט. במבנה של עץ זה, עלים מייצגים סיווגים אפשריים (אופציות סיווג נוספות) וענפים מייצגים צירופים של תכונות אשר יובילו למחלקות הסיווג.

כך למשל ניתן לראות בדוגמה באה עץ החלטה האם להעסיק עובד:

ניתן לראות כי ההחלטה מתקבלת על ידי ההחלטה הבאה בעץ, ולא כל העץ בבת אחת. כלומר, מאפיין אחד בכל פעם. תחילה, נלקח בחשבון המאפיין החשוב ביותר, (יותר למעלה בעץ ההחלטה- האם יש תואר) לאחר מכן נלקחים שנות הניסיון, וכן הלאה.



האלגוריתם בוחר במאפיין (שעדיין לא נבחר) בעל רווח המידע הצפוי הגבוה ביותר- כאשר קיימת רמת סף מסוימת (מה המקסימום האפשרי), האלגוריתם קובע את רמת הסף האופטימלית עבור כל אחד מהמאפיינים ברמה (כלומר, את רמת הסף שמביאה את רווח המידע הצפוי עבור אותו מאפיין) ומשווה בין השניים לידיעת התוצאה האופטימלית ברמה הנוכחית.

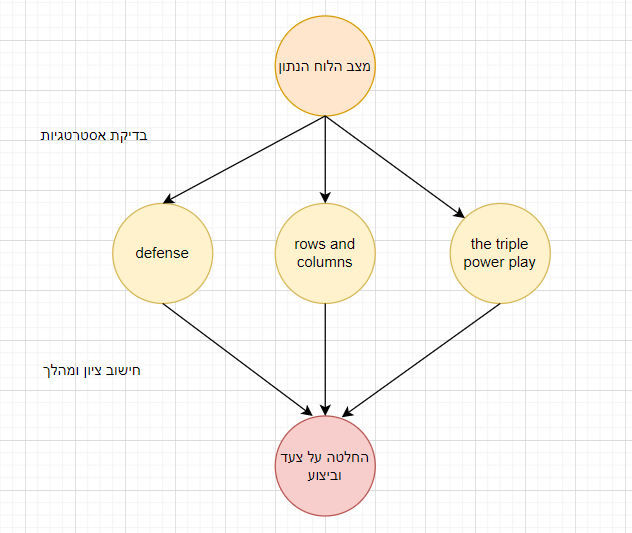
הכרעת מצבים-

הכרעת מצבים היא מצב שבו ישנם מצבים אפשריים, ובהתאם למצב הנוכחי בוחרים את המצב המתאים ביותר. בכל עת, רק מצב מסוים יכול להיות פעיל, כך שיש לעבור ממצב אחד לאחר על מנת לבצע פעולות שונות, כאשר כל מצב מייצג פעולה מסוימת, כמו התקפה או הגנה. זהו מודל השימושי למשחקי AI. כך למשל, החשיבה של האויב במשחק יכול להיות ממומשת במודל זה. למעשה, ישנם מספר מצבים אפשריים ומתוכם, נבחר אחד המתאים ביותר. הבחירה תיעשה על ידי ניקוד מתאים לכל מצב, כאשר המצב הכי טוב ינוקד בציון הכי גבוהה, ולפיו נבחר את התור הבא.

כך, אוכל במשחק לשלב את האסטרטגיות המתאימות (התקפה, הגנה...)

במשחק, הכרעת המצבים פועלת כך:

קריאה לפונקציות האסטרטגיה, כל אחת מחשבת מציון ומהלך, וממנה מחליטים על צעד יעיל וביצועו.



# **אסטרטגיה נבחרת לפתרון- תיאור האסטרטגיה והנימוק לבחירתה**

האסטרטגיה הנבחרת על השחקן הממוחשב היא הכרעת מצבים. למשחק ישנם מספר אסטרטגיות אשר מתארות מצבים אופטימליים לניצחון. לפי מצב הלוח הנתון נבחר את המהלך הכי טוב שיכול השחקן הממוחשב לעשות על מנת לקרבו לניצחון, כאשר לכל מהלך ניתן ניקוד- ניקוד גבוהה=מהלך טוב המקרב לניצחון, ניקוד נמוך=מהלך לא כדאי.

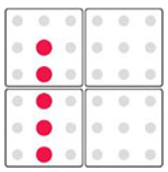
כלומר, פונקציית ההערכה תדע לעבור על כל אחת מהאסטרטגיות, תקבל ציון מכל אחת והמלצה לצד הבא שעליה לעשות. היא תבחר את האסטרטגיה עם הציון הכי גבוהה, וכך תעשה כל תור ותור.

למשחק ישנן מספר אסטרטגיות שהגדרתי:

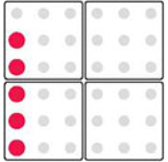
**Rows and columns**- יכול להיות במאונך או במאוזן, באמצעותו ניתן לגשת לניצחון בהרבה דרכים, כאשר מייצרים שלישייה בלוח אחד (שיכולה לעבור במרכז) וזוג בלוח אחר (שיכול לעבור במרכז). אסטרטגיה מתאימה גם להתקפה וגם להגנה.

מבדילים בין המקרים, כאשר השורה או העמודה לא עוברים במרכז, ולמעשה קלים יותר להתקפה, כך שליריב קשה יותר להגיב בחזרה.

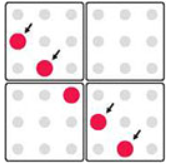
עובר במרכז, מנוקד פחות טוב:



לא עובר במרכז, מנוקד יותר טוב:



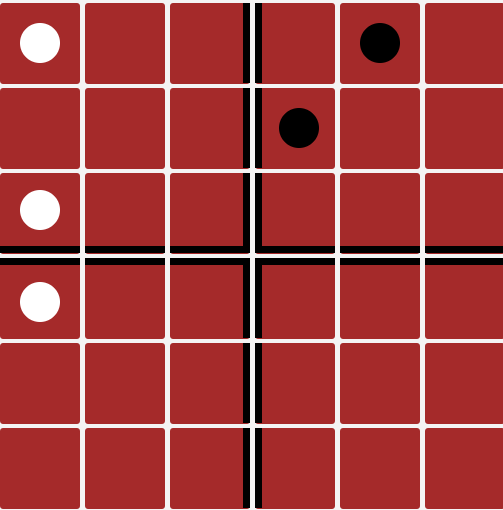
**The triple power play**- אלכסון אשר לא עובר במרכזים כלל. האלכסון עובר בשלושה בלוקים שונים. קשה מאוד לעלות על ניצחון מסוג זה, ולכן זהו יתרון גדול שלו. אסטרטגיה מתאימה בעיקר להתקפה, אך גם להגנה.



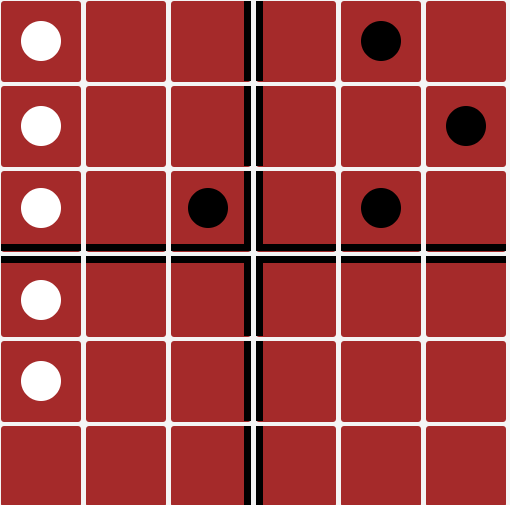
כאמור, אסטרטגית ההגנה כוללת את כל אחד מהמצבים שהצגתי לעיל, כאשר ההגנה מתחילה במצב שבו ישנם שלושה חיילים מהשחקן הנגדי אשר מתאימים לתבנית ניצחון מסוימת. תבנית ניצחון הכוונה לחמישה חיילים ברצף היוצרים ניצחון. בעת זיהוי של מקרה כזה, אסטרטגית ההתקפה במצב "hold", עד אשר הצלחת ההגנה ויציאה ממצב סכנה.

פונקציית ההגנה פועלת כך למשל בדוגמה לעיל:

זיהוי סכנה עתידית- זיהוי של שלושה חיילים מצבע היריב אשר מתאימה לתבנית ניצחון:

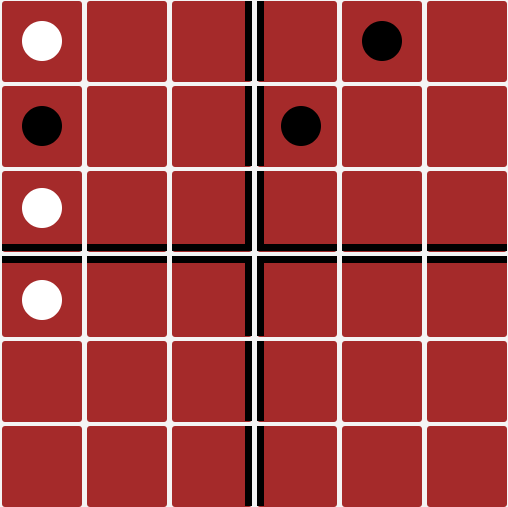


תבנית הניצחון המהווה סכנה עתידית:



כך למשל, היא תפעל:

תשים את החייל שלה בלוח מספר 1, ותסובב את הלוח הנגדי- במקרה הזה, לוח מספר 3.



אסטרטגית ההתקפה לעומת זאת, מורכבת משילוב אסטרטגיות עיקריות:

* The triple power play
* Rows and columns

שתי האסטרטגיות הללו מרכיבות יחדיו את ההתקפה, כאשר האסטרטגיה The triple power play מוגדרת באופן דיפולטי והתחלתי עם ציון טוב יותר, כיוון שקשה יותר לעלות עליה והיא קשה להגנה. כך, מכונת המצבים תריץ את שתי האסטרטגיות הללו, ותחזיר לנו ציון טוב יותר, לפיו נדע איזה התקפה עדיפה כרגע.

הכרעת המצבים היא שעשיתי מוגדרת כך:

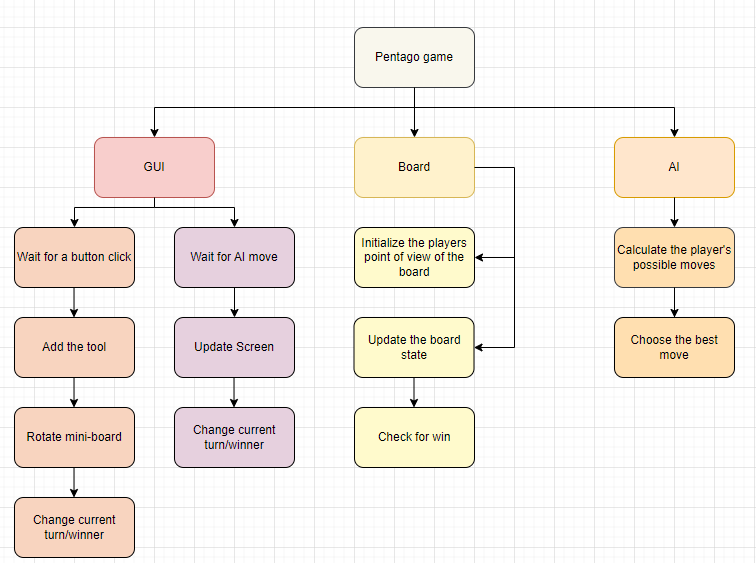
* ניקוד לאסטרטגיה the triple power play היא 80.
* ניקוד לאסטרטגיה rows and columns כאשר לא עובר במרכזים היא 60.
* ניקוד לאסטרטגיה rows and columns כאשר כן עובר במרכזים היא 40.
* ניקוד להגנה היא 100.
* במידה ויש מהלך אחד לפני ניצחון מובטח, ניקוד יהיה 120.

בחרתי באסטרטגיה זו, כיוון שהיא בודקת את כל המקרים האפשריים בהן היריב יכול לנצח (אסטרטגית הגנה), פועלת באסטרטגית התקפה יעילה, ולמעשה "ממליצה" לשחקן הממוחשב את הדרך היעילה ביותר לאחר שסקרה את המקרים הרצויים.

למעשה, באסטרטגיות אלו אוכל להשתמש בשני מונחים חשובים:

* מיני הלוחות הם **סימטריים**, ולכן פעולה במיני לוח אחד תיראה אותו דבר בדיוק בלוח אחר.
* בעת **סיבוב** מיני לוחות, נוצר מצב של **מראה והשתקפות**, וכך נוכל לבצע פעולה שתהיה זהה בכל מיני הלוחות.

# **ארכיטקטורה של הפתרון המוצע בפורמט של Design Level down-Top**



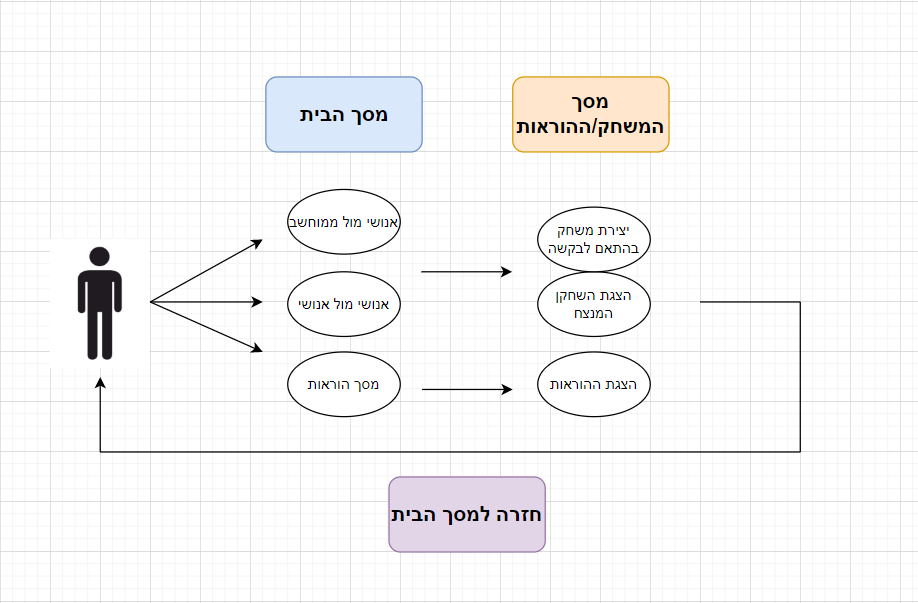
GUI- אחראי על תצוגת המשחק, אחראי על שינוי הלוח וסיבובו בעת תור של שחקן, אחראי על הצגת התור הנוכחי ועל המנצח, ואחראי על בחירת סוג משחק (אנושי מול אנושי או אנושי מול ממוחשב).

Board- אחראי על אתחול הלוח, ושינוי הלוח עבור שני השחקנים (לפי נקודת מבטם על המשחק). אחראי לבדוק ניצחון או תיקו.

AI- אחראי על חישוב האסטרטגיות עבור השחקן הממוחשב, כאשר בחירת האסטרטגיה הכי טובה תיקבע בהתאם לציונים שקיבלה כל אסטרטגיה.

# **תרשים מקרי-שימוש cases Use UML**

תרשים המתאר את האינטראקציה של הלקוח עם המערכת שבניתי בפרויקט:



# **מבנה נתונים**

בחירת מבנה הנתונים הוא חלק חשוב מאוד בעת מימוש המשחק, ובפרט לשם ביצועים טובים של השחקן הממוחשב- זמנים יעילים.

גודל לוח הפנטגו הוא 6x6. אממש את הלוח באמצעות bit map. כיוון שבמשחק יש שני שחקנים, אשמור את מצב הלוח בשני משתנים בגודל long (64 ביטים). משתנה ראשון יהיה עבור השחקן הראשון (צבע לבן) והמשתנה השני יהיה עבור השחקן השני/AI (צבע שחור). המשחק הוא מסוג ידע מלא, כלומר כל שחקן יודע באופן מלא את מהלכיו של השחקן השני והכל חשוף לו.

המשחק עובד על סיבוב הלוח- אשתמש בפעולות בינאריות על המשתנה- פעולות and, shift, not, or. באמצעות מצב המשתנה אוכל לדעת את מצב הלוח בכל רגע נתון, ולדעת אם הגענו במשחק לניצחון או תיקו.

עבור משתנה של שחקן שחור, כל חייל שחור שיש עליו ייוצג באמצעות הביט 1, כל שאר הביטים יהיו 0, ועבור משתנה של שחקן לבן, כל חייל לבן שיש עליו ייוצג באמצעות הביט 1, וכל השאר 0.

לדוגמה, עבור הלוח הבא אייצג:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 1 |

כך יישמר המידע על מצב הלוח (כאשר אחרי ישנם אפסים).

100011011 111001010 110001100 100111111

* אם ארצה לראות איך שינוי מתבצע בלוח יחיד- לוח מספר 1 (3x3) עבור סיבוב **ימינה**:

100 011 011

001 110 110

* אם ארצה לראות איך שינוי מתבצע בלוח יחיד- לוח מספר 1 (3x3) עבור סיבוב **שמאלה**:

100 011 011

011 011 100

מבנה נתונים זה הוא יעיל מאוד כיוון שכל המידע על הלוח נשמר בביטים ולא תופס הרבה מקום. בנוסף, הפעולות השונות המתרחשות בלוח, מתבצעות גם הן על ידי פעולות על סיביות- יעיל מאוד. מבנה נתונים זה מתאים לי לפרויקט כיוון שישנם רק שני מצבים עבור כל חייל- יש שחקן (1) או אין שחקן (0).

# **תיאור סביבת העבודה ושפת התכנות**

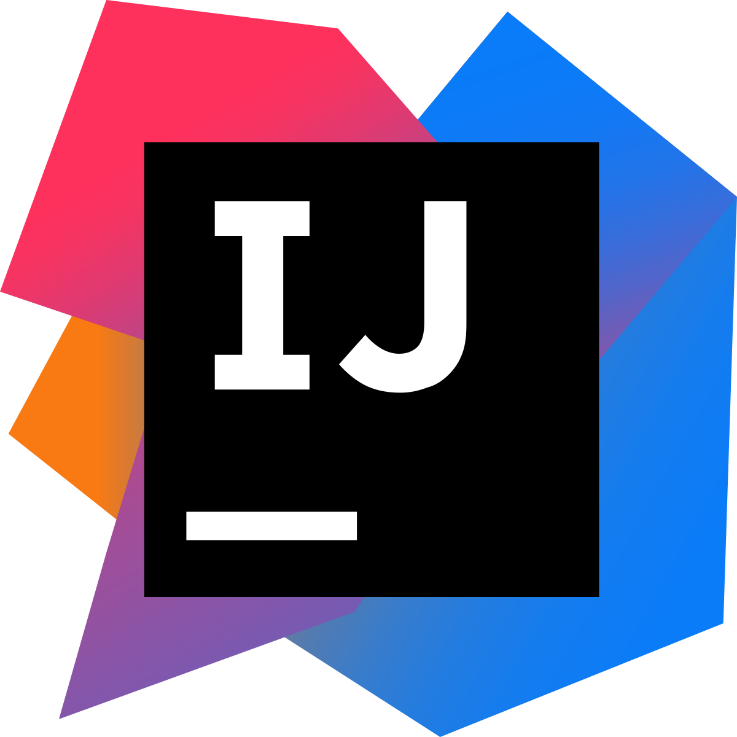
**עמדת פיתוח:** הפרויקט מיישם את המשחק פנטגו בשפת JAVA.

**מערכת הפעלה:** Microsoft – Windows 11

**שפת תכנות:** Java JDK JavaSe – 16

**סביבת עבודה:** Intellij IDEA Community Edition 2022.2.1

**מימוש גרפי:** חבילת JAVAFX



# **אלגוריתם ראשי (בפורמט פסיאודו קוד)**

האלגוריתם הראשי למעשה מורכב משלוש אסטרטגיות ראשיות:

* Rows and columns.
* The triple power play.
* Defense.

כל אחת מהאסטרטגיות למעשה מחזירה ציון והמלצה, למיקום החייל, איזה מיני לוח לסובב ולאיזה כיוון לסובב אותו. כאלגוריתם למעשה, יבחר באסטרטגיה עם הציון הכי טוב, ולפיה תפעל.

**אסטרטגית rows and columns:**

אסטרטגיה זו למעשה, פועלת על חמישייה באורך או ברוחב, ומאפשרת שלושה מצבים שונים אפשריים לניצחון:

**הראשונה**, מקבלת ציון 60 כיוון שלא עוברת במרכזים:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |

היא מורכבת משמונה מצבים שונים:

ארבעה מתוכם מסודרים כך:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 |

והארבעה הנוספים כך:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 |

למעשה, אם אוכל לחשב את הצעד הבא בצורה הכי טובה עבור מיני לוח 1, 2 למשל, אשר מהווים יחדיו 2 מסיכות, אוכל לחשב את כל המסכות האפשריות. אעשה זאת באמצעות סיבוב של הלוח כולו ימינה ושמאלה ובאמצעות תמונת מראה.

כך למעשה, אעשה תמיד את הבדיקה על מיני הלוחות1, 2 ואקבל אינדקס מתאים וסיבוב מתאים אותו אצטרך להתאים בחזרה למיני הלוחות המקוריים.

2 מסכות בצבע תכלת, עבור מיני לוחות 1,2.

2 מסכות בצבע אפור, עבור מיני לוחות 1,3, כמו מיני לוחות 1,2 עם סיבוב ימינה של כל הלוח.

2 מסכות בצבע ירוק, עבור מיני לוחות 3, 4, כמו מיני לוחות 1, 2 עם תמונת מראה.

2 מסכות בצבע כחול, עבור מיני לוחות 2,4, כמו מיני לוחות 1,2 עם סיבוב שמאלה של כל הלוח.

**השנייה**, מקבלת ציון 40 כיוון שכן עוברת במרכזים:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |

גם היא מורכבת משמונה מצבים שונים לניצחון.

**השלישית** והאחרונה, מקבלת ציון 60 כיוון שלא עוברת במרכזים:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |

גם היא מורכבת משמונה מצבים שונים לניצחון.

האלגוריתם פועל כך:

1. תחילה, נבחר באופן רנדומלי מסיכה אחת מבין השמונה.
2. שים חייל ראשון באחד מהמרכזים של מיני הלוחות, למשל הביטים שצבעתי באדום בדף הקודם.
3. כעת, שים חייל שני באחד המרכזים השניים, שלא תפוס.
4. לאחר מכן, שים באופן רנדומלי באחד משלושת המקומות האחרים, תוך כדי בדיקה שלא תפוס.
5. למעשה, לכל מסכה יש 5 אופציות לשים חיילים ולסובב בהתאם. כך, נבדוק אם אחת המסכות מתאימה ולפיה נפעל.

לדוגמה, במצב כזה, נדע שזוהי אחת מהמסכות האפשריות לסיבוב ולניצחון, כפי שציינתי בשלב 5, ולכן נשים את החייל כעת בצבע הצהוב ואסובב את מיני לוח 2 ימינה.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |

כך שנקבל את הלוח הבא:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |

**אסטרטגית Defense:**

אסטרטגיית ההגנה פועלת על כל האסטרטגיות הקיימות שציינתי. היא למעשה בוחנת את כל האסטרטגיות שבהן היריב יכול לנצח, מזהה אם יש איזשהו דפוס של ניצחון מצידו ופועלת בהתאם. למעשה הפונקציה פועלת רק כאשר יש 3 או 4 חיילים שמתאימים לאסטרטגיה מסוימת, ולא לפני.

כפי שציינתי ב rows and columns, ניתן לחשב על מיני לוח מסוים, ועל פיו על כל השאר. כך למעשה, נבדוק את כל המסכות על מיני לוחות 1, 2 ומשם נדע את כולם. אלגוריתם זה למעשה, פועל בדיוק כמו אסטרטגיית התקפה, רק שבודקים את היריב, ומנסים להפסיל אותו. כך, אם אראה מצב בתור אסטרטגית הגנה אפעל אותו דבר אם הוא אסטרטגית התקפה, רק על היריב.

האלגוריתם פועל כך:

1. עבור על כל אחת מאסטרטגיות הניצחון:
   1. בדוק האם באסטרטגיה יש ארבעה חיילים שמקרבים לניצחון היריב.
      1. אם כן, בדוק לפי המסכות לאן לסובב ואיזה חייל לשים איפה והחזר.
   2. בדוק האם באסטרטגיה יש שלושה חיילים שמקרבים לניצחון היריב.
      1. אם כן, בדוק לפי המסכות לאן לסובב ואיזה חייל לשים איפה, אך תמשיך לבדוק, כך שאם נמצא אסטרטגיה שמקרבת לניצחון של היריב עם ארבעה חיילים תהיה לה עדיפות.

**אסטרטגית the triple power play:**

אסטרטגיה זו פועלת בצורה אלכסונית אשר משתרכת על גבי שלושה מיני לוחות. למעשה, יש לה ארבע אופציות לניצחון, כאשר היא מדורגת הכי גבוה באסטרטגיות ההתקפה, כיוון שהכי קשה לעלות עליה ולעשות אותה.

למשל עבור אסטרטגית ניצחון זו:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |

לחלק האפור ישנן 4 אופציות לשים בהן 2 חיילים, כך גם לירוק, ולכחול יש 4 אופציות עבור חייל בודד. כך למעשה, נוכל לבחור בכל אחת מהדרכים הללו וכמובן לסובב בהתאם על מנת שיגיע ליעד שנרצה. לחלק הכחול יש יותר אופציות למקמו ולכן אותו נשאיר לסוף, לאחר שמיקמו 4 חיילים על הלוח, 2 אפורים ו 2 ירוקים.

בנוסף, שני האלכסונים שצבעתי למעשה פועלים אותו הדבר, חוץ מהחייל הבודד, ולכן בקוד התייחסתי אליהם כאל יחידה אחת, וכך גם לאלכסונים מימין.

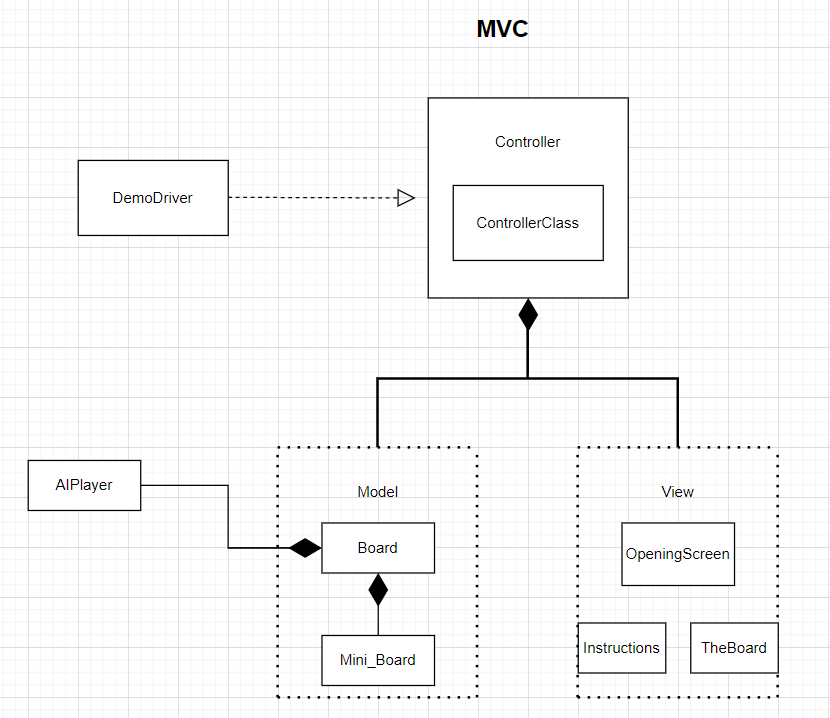
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |

האלגוריתם פועל כך:

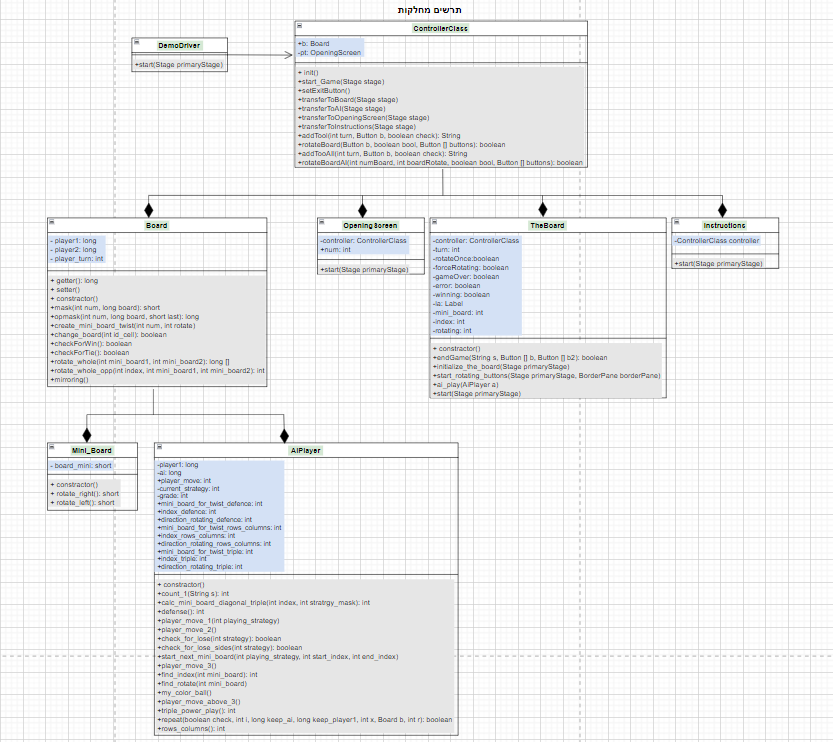
1. אם כעת אתה שם חייל ראשון על הלוח:
   1. בחר רנדומלית באחת משתי היחידות שציינתי- ימין או שמאל לאלכסונים.
   2. שים חייל ראשון אחד משמונה המקומות- ארבע אפורים או ארבע ירוקים וסובב בהתאם.
2. אחרת אם כעת אתה שם חייל שני על הלוח:
   1. בדוק שאין חייל ובחר בהתאם לחייל קודם ששמת חייל נוסף, כלומר מימינו או משמאלו אך לא ממולו וסובב בהתאם.
3. אחרת אם כעת אתה שם חייל שלישי על הלוח:
   1. בחר במיני הלוח השני- זה שלא שמת עליו עדיין חיילים.
   2. בדוק שאין חייל, ושים חייל ראשון באחד מארבע המקומות וסובב בהתאם.
4. אחרת אם כעת אתה שם חייל רביעי על הלוח:
   1. בדוק שאין חייל ובחר בהתאם לחייל הקודם ששמת חייל נוסף, וסובב בהתאם.
5. אחרת אם כעת אתה שם חייל חמישי ואחרון על הלוח:
   1. בדוק אם החייל שמשלים את המסכה תפוס.
      1. אם הוא לא תפוס שים וסובב בהתאם.
      2. אם הוא תפוס שים באחד משאר המקומות הכחולים וסובב בהתאם.
6. אם חל הפסד ביחידה הנוכחית איתה עובדים, עבור ליחידה השנייה והתחל מההתחלה.

# **תרשים מחלקות- UML Class Diagram**

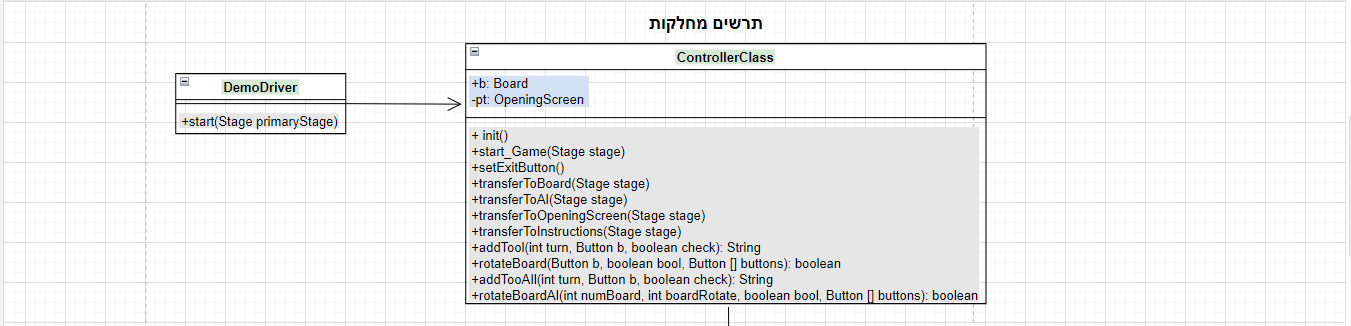
תרשים MVC

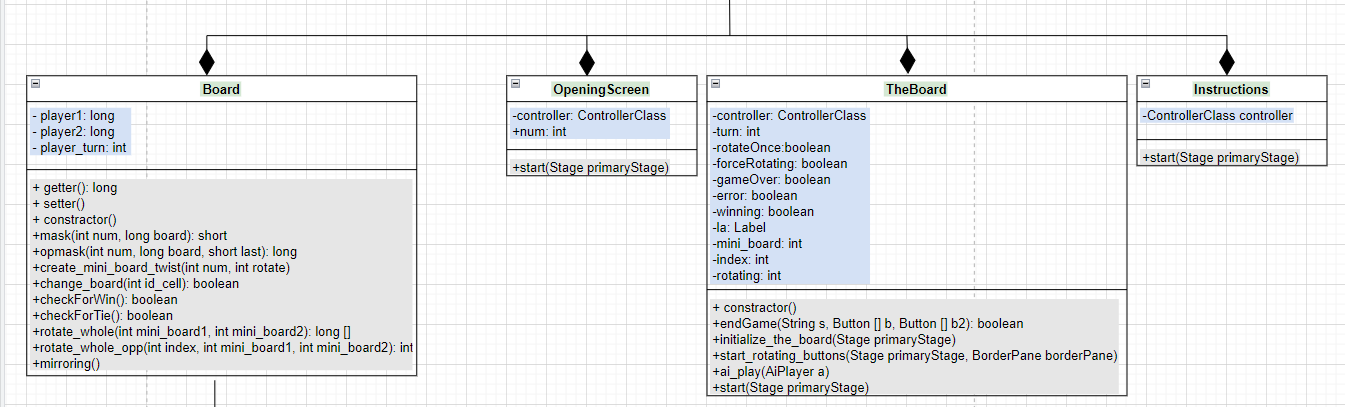


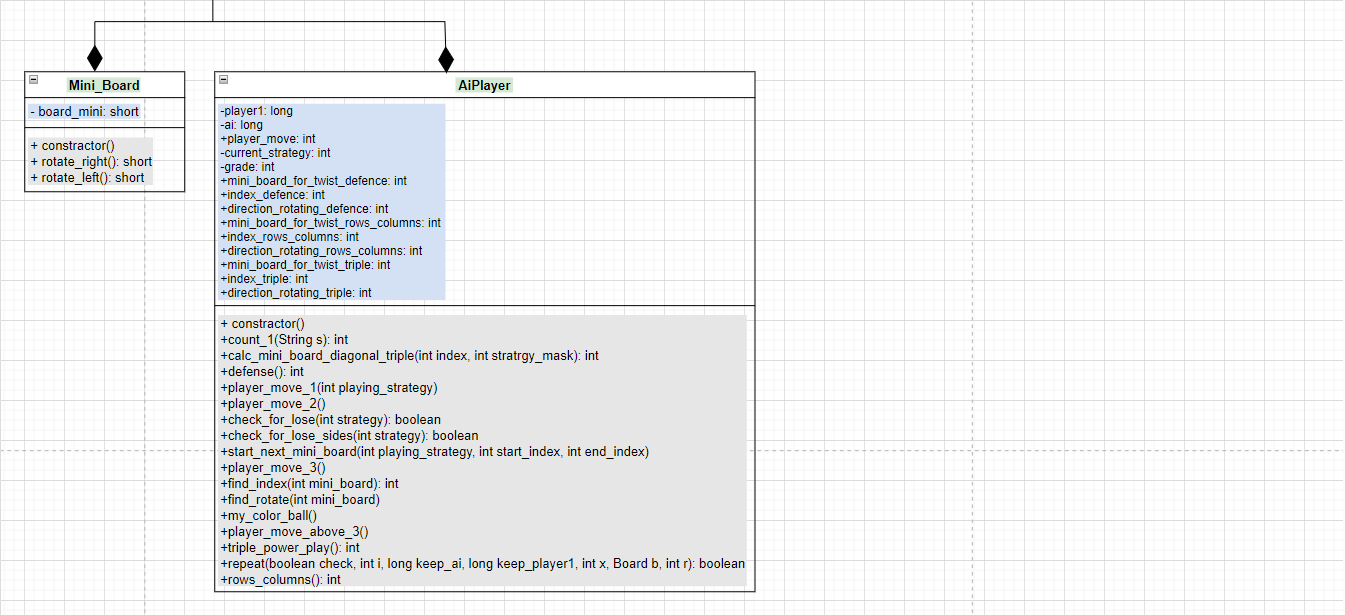
תרשים UML



התרשים מקרוב







# **תיאור המחלקות הראשיות בפרויקט**

הנחה: n הוא מספר השורות והעמודות במשחק. במקרה שלנו n=6.

מחלקת DemoDriver

מחלקה זו מריצה את התכנית. כל מטרתה היא ליצור מופע של ה controller ולאתחל את ה model וה view שבתוכו. בנוסף, היא שולחת את ה stage הראשי.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| שם הפונקציה | מה הפונקציה מבצעת | מה הפונקציה מקבלת | מה הפונקציה מחזירה | סיבוכיות |
| start | הפונקציה יוצרת מופע של ה controller ושולחת אליו את הstage . | primaryStage- stage ראשי שישמש את כל התכנית ליצירת ה view. |  | O(1) |

מחלקת MiniBoard

מחלקה זו מנהלת את סיבובי מיני הלוחות, ימינה ושמאלה.

שדות עיקריים:

* Board\_mini, משתנה מסוג short אשר מייצג את מיני הלוח איתו אנו עובדים.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| שם הפונקציה | מה הפונקציה מבצעת | מה הפונקציה מקבלת | מה הפונקציה מחזירה | סיבוכיות |
| MiniBoard | פעולה בונה. | Board\_mini- מסוג short מייצג את מיני הלוח איתו נעבוד. |  | O(1) |
| rotate\_right | מבצעת סיבוב ימינה של מיני הלוח שקיבלנו. |  | את מיני הלוח לאחר הסיבוב ימינה. | O(n) |
| rotate\_left | מבצעת סיבוב שמאלה של מיני הלוח שקיבלנו. |  | את מיני הלוח לאחר הסיבוב שמאלה. | O(n) |

**הסבר על הסיבובים:**

סיבוב ימינה-

למשל על הלוח הבא:

|  |  |  |
| --- | --- | --- |
| 0 | 0 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

נקבל את הלוח הבא:

|  |  |  |
| --- | --- | --- |
| 1 | 1 | 0 |
| 0 | 0 | 1 |
| 0 | 1 | 1 |

|  |  |
| --- | --- |
| מיקום לפי צבע | הזזת הביט |
|  | הזזה ימינה ב 2 |
|  | הזזה ימינה ב 4 |
|  | הזזה ימינה ב 6 |
|  | הזזה שמאלה ב 2 |
|  | ללא הזזה |
|  | הזזה ימינה ב 2 |
|  | הזזה שמאלה ב 6 |
|  | הזזה שמאלה ב 4 |
|  | הזזה שמאלה ב 2 |

סיבוב שמאלה-

למשל על הלוח הבא:

|  |  |  |
| --- | --- | --- |
| 0 | 0 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

נקבל את הלוח הבא:

|  |  |  |
| --- | --- | --- |
| 1 | 1 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 1 |

|  |  |
| --- | --- |
| מיקום לפי צבע | הזזת הביט |
|  | הזזה ימינה ב 6 |
|  | הזזה ימינה ב 2 |
|  | הזזה שמאלה ב 2 |
|  | הזזה ימינה ב 4 |
|  | ללא הזזה |
|  | הזזה שמאלה ב 4 |
|  | הזזה ימינה ב 2 |
|  | הזזה שמאלה ב 2 |
|  | הזזה שמאלה ב 6 |

פסיאודו קוד עבור סיבוב ימינה/שמאלה:

1. נשמור את הערך של הלוח במשתנה.
2. עבור השורה הראשונה בלוח (מורכב משלוש שורות, כל שורה שלושה חיילים):

2.1. הזז את הלוח בהתאם להזזת הביט שציינתי למעלה ושמור זאת במשתנה.

2.2. חבר את המשתנה ששמרת עם העתק של הלוח ושמור זאת במשתנה.

2.3. התקדם אל החייל הבא בשורה.

1. שמור את העתק הלוח לאחר השינוי במשתנה level1.
2. עבור השורה השנייה בלוח:

4.1. הזז את הלוח בהתאם להזזת הביט שציינתי למעלה ושמור זאת במשתנה.

4.2. חבר את המשתנה ששמרת עם העתק של הלוח ושמור זאת במשתנה.

4.3. התקדם אל החייל הבא בשורה.

1. שמור את העתק הלוח לאחר השינוי במשתנה level2.
2. עבור השורה השלישית בלוח:

6.1. הזז את הלוח בהתאם להזזת הביט שציינתי למעלה ושמור זאת במשתנה.

6.2. חבר את המשתנה ששמרת עם העתק של הלוח ושמור זאת במשתנה.

6.3. התקדם אל החייל הבא בשורה.

1. שמור את העתק הלוח לאחר השינוי במשתנה level3.
2. עבור על כל השורות(שלוש שורות):

8.1. הוסף כל חייל בשלושת משתני ה level שהגדרנו אל משתנה שיחזיק את מצב הלוח לאחר השינוי- last.

1. החזר את last.

מחלקת Instructions

מחלקה זו יוצרת את מסך ההוראות אליו יכול לעבור המשתמש. מכיל את ה label של החוקים, תמונה להמחשה וכפתור יציאה מהמסך או כפתור חזרה למסך הבית. בנוסף, מאפשר לגלול את המסך.

שדות עיקריים:

* controller, משתנה המאפשר גישה אל ה controller- כמו חזרה לתפריט הראשי או יציאה מהמסך.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| שם הפונקציה | מה הפונקציה מבצעת | מה הפונקציה מקבלת | מה הפונקציה מחזירה | סיבוכיות |
| start | פעולה המגדירה את מסך ההוראות של המשתמש. כלומר, חוקי המשחק. | Stage ראשי שקיבלנו מה controller. |  | O(1) |

מחלקת OpeningScreen

מחלקה זו יוצרת את המסך הראשוני אליו חשוף המשתמש. במסך זה למשתמש יש שלוש אופציות עיקריות לבחירה:

* העברה למסך של שחקן אנושי מול שחקן אנושי.
* העברה למסך של שחקן אנושי מול שחקן ממוחשב.
* העברה למסך של הוראות למשחק- חוקי המשחק.

שדות עיקריים:

* controller, משתנה המאפשר גישה אל ה controller- כמו חזרה לתפריט הראשי או יציאה מהמסך.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| שם הפונקציה | מה הפונקציה מבצעת | מה הפונקציה מקבלת | מה הפונקציה מחזירה | סיבוכיות |
| start | פעולה המגדירה את מסך הפתיחה של המשתמש. | Stage ראשי שקיבלנו מה controller. |  | O(1) |

מחלקת Board

מחלקה זו אחראית לכל הפעולות הנוגעות בשינוי הלוח- הוספת כלי ללוח, בדיקת ניצחון, בדיקת תיקו, בקשה לסיבוב מיני לוח או בקשה לסיבוב כל הלוח.

שדות עיקריים:

* player1- שחקן מספר 1, אשר מייצג את הלוח מזווית הראייה שלו. מסוג long וכאשר יש סיבית 1 משמע יש כלי בלוח, וסיבית 0 משמע הכלי ריק או מכיל כלי של שחקן 2.
* Player2- שחקן מספר 2, אשר מייצג את הלוח מזווית הראייה שלו. מסוג long וכאשר יש סיבית 1 משמע יש כלי בלוח, וסיבית 0 משמע הכלי ריק או מכיל כלי של שחקן 1.
* Player\_turn- מכיל את השחקן הנוכחי שמשחק:

אנושי מול אנושי- 1 – שחקן 1, 0- שחקן 2.

אנושי מול ממוחשב- 1- שחקן ממוחשב, 0-שחקן 1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| שם הפונקציה | מה הפונקציה מבצעת | מה הפונקציה מקבלת | מה הפונקציה מחזירה | סיבוכיות |
| Board | פעולה בונה – אשר מאתחלת את השחקן הנוכחי להיות 1, את הלוחות להיות 0. |  |  | O(1) |
| Mask | ממירה לוח של שחקן מסוים מ long ל short, בהתאם למיני לוח הרצוי לסיבוב. | Num- מיני הלוח שנרצה לסובב  board- הלוח לפי נקודת מבטו של שחקן מסוים. | מחזירה את הלוח ב short. | O(1) |
| Opmask | משנה את הלוח בהתאם למיני לוח שקיבלנו לאחר הסיבוב. ממירה ל long. | Num- מיני הלוח שנרצה לסובב  board- הלוח לפי נקודת מבטו של שחקן מסוים.  Last- מיני הלוח לאחר הסיבוב. | מחזירה את הלוח ב long לאחר הסיבוב. | O(1) |
| Create\_mini\_board\_twist | מבצעת את הסיבוב ומשנה את הלוחות בהתאם. | Num- מיני הלוח שנרצה לסובב.  Rotate- כיוון הסיבוב. |  | O(1) |
| Change board | מוסיפה כלי ללוח. | Id\_cell- ה id של הכלי אותו נרצה להוסיף. | מחזירה אמת אם הוספת הכלי היא חוקית, אחרת מחזירה שקר. | O(1) |
| checkForWin | הפונקציה בודקת ניצחון. |  | מחזירה אמת אם יש ניצחון, שקר אם אין. | O(n) |
| checkForTie | הפונקציה בודקת תיקו. |  | מחזירה אמת אם יש תיקו, שקר אם אין. | O(1) |
| rotate\_whole | הפונקציה מסובבת את כל הלוח ימינה או שמאלה. על מנת שיהיו כמו מיני לוחות 1,2. | Mini\_board1- מיני הלוח הראשון אליו נתייחס.  Mini\_board2- מיני הלוח השני אליו נתייחס. | מערך מסוג long בגודל 2, שכן מייצג את שני השחקנים לאחר הסיבוב. | O(n^2) |
| rotate\_whole\_opp | הפונקציה משנה את האינדקס שקיבלנו בהתאם ללוחות 1, 2 עבור המיני לוחות איתם התעסקנו לפני הסיבוב. | Index- האינדקס המומלץ בהתאם ללוחות 1, 2  Mini\_board1- מיני הלוח הראשון אליו נתייחס.  Mini\_board2- מיני הלוח השני אליו נתייחס. | מחזירה את האינדקס החדש, בהתאם למיני הלוחות אותם קיבלנו. | O(n^2) |
| mirroring | הפונקציה מבצעת תמונת מראה עבור מיני לוח 3,4 להגעה למיני לוח 1,2. |  |  | O(n^2) |

פסיאודו קוד עבור פונקציית mirroring:

**נעשה קטע קוד זה פעמיים- עבור השחקן הממוחשב ועבור השחקן האנושי.**

1. חזור פעמיים:
   1. חזור תשע פעמים(עבור כל ביט במיני לוח):
      1. אם הגעת לביט באינדקס שקטן משלוש:
         1. אם לשחקן הממוחשב יש חייל במיקום זה:
            1. הזז אותו ימינה שש צעדים- המיקום המקביל אליו כמו מראה.
      2. אם הגעת לביט באינדקס שגדול מחמש:
         1. אם לשחקן הממוחשב יש חייל במיקום זה:
            1. הזז אותו שמאלה שש צעדים- המיקום המקביל אליו כמו מראה.
      3. עבור לביט הבא.

פסיאודו קוד עבור פונקציית rotate\_whole\_opp:

1. אם מיני הלוחות הם 2,4 (כדי לחזור לצורתם המקורית ולא 1,2 יש לסובב ימינה):
   1. נוסיף את מיקום החייל אל הלוח.
   2. נחזיר את הלוחות למקומם המקורי- מיני לוח 1 ל 4 ומיני לוח 2 ל 1.
   3. נסובב ימינה בחזרה את מיני הלוחות 2, 4.
   4. כעת החייל יהיה במיקום הנכון ונרצה לברר את ה id שלו, ולכן חזור כל עוד לא הגעת לחייל(לא הגעת לסיבית שערכה 1):
      1. תגדיל ב 1 את ה id.
      2. תזיז את הלוח בצעד אחד ימינה על מנת להמשיך לחייל הבא.
   5. החזר את ה id.
2. אם מיני הלוחות הם 1,3 (כדי לחזור לצורתם המקורית ולא 1,2 יש לסובב שמאלה):
   1. נוסיף את מיקום החייל אל הלוח.
   2. נחזיר את הלוחות למקומם המקורי- מיני לוח 1 ל 3 ומיני לוח 2 ל 1.
   3. נסובב שמאלה בחזרה את מיני הלוחות 1,3.
   4. כעת החייל יהיה במיקום הנכון ונרצה לברר את ה id שלו, ולכן חזור כל עוד לא הגעת לחייל(לא הגעת לסיבית שערכה 1):
      1. תגדיל ב 1 את ה id.
      2. תזיז את הלוח בצעד אחד ימינה על מנת להמשיך לחייל הבא.
   5. החזר את ה id.
3. אם מיני הלוחות הם 3,4:
   1. נשמור את מצב הלוח לפני, ונזמן את הפעולה mirroring, אשר תבצע חזרה המרה ללוחות 3, 4.
   2. כעת החייל יהיה במיקום הנכון ונרצה לברר את ה id שלו, ולכן חזור כל עוד לא הגעת לחייל(לא הגעת לסיבית שערכה 1):
      1. תגדיל ב 1 את ה id.
      2. תזיז את הלוח בצעד אחד ימינה על מנת להמשיך לחייל הבא.
   3. החזר את ה id.
4. החזר 1(לא יגיע אף פעם).

פסיאודו קוד עבור פונקציית rotate\_whole:

1. אם מיני הלוחות הם 2,4(סיבוב שמאלה על מנת להגיע ללוחות 1,2):
   1. סובב שמאלה את לוחות 2, 4
   2. בצע פעולות הזזה על מנת שנקבל את לוח 2 במקום של לוח 1, תוך כדי מחיקת כל שאר הלוחות, ושמור זאת במשתנה. נעשה זאת עבור שני השחקנים.
   3. בצע פעולות הזזה על מנת שנקבל את לוח 4 במקום של לוח 2, תוך כדי מחיקת כל שאר הלוחות, ושמור זאת במשתנה. נעשה זאת עבור שני השחקנים.
   4. חבר את שני המשתנים הללו כך שלוח 1,2 יהיו מלאים כמו 2,4 בהתאמה, ולוחות 1,3 יהיו ריקים.
2. אחרת אם מיני הלוחות הם 1,3(סיבוב ימינה על מנת להגיע ללוחות 1,2):
   1. סובב ימינה את לוחות 1,3
   2. בצע פעולות הזזה על מנת שנקבל את לוח 3 במקום של לוח 1, תוך כדי מחיקת כל שאר הלוחות, ושמור זאת במשתנה. נעשה זאת עבור שני השחקנים.
   3. בצע פעולות הזזה על מנת שנקבל את לוח 1 במקום של לוח 2, תוך כדי מחיקת כל שאר הלוחות, ושמור זאת במשתנה. נעשה זאת עבור שני השחקנים.
   4. חבר את שני המשתנים הללו כך שלוח 1,2 יהיו מלאים כמו 3,1 בהתאמה, ולוחות 2,4 יהיו ריקים.
3. אחרת אם מיני הלוחות הם 3,4:
   1. הזז את שני המשתנים 2 לוחות שמאלה.
   2. קרא לפונקציית mirroring, אשר תבצע תמונת מראה של לוחות 3,4 עבור לוחות 1,2.
4. שמור את ערכי לוחות השחקנים, במערך בגודל 2 והחזר אותו.

מחלקת ControllerClass

מחלקה זו אחראית על ניהול הקשר בין ה view ל model, למעשה משמשת כמעיין "מתווכת", אשר מקבלת בקשות ומעדכנת בהתאם את הצדדים.

שדות עיקריים:

* B, מופע של המחלקה Board, אשר מאפשר גישה ושינוי של הלוחות של השחקנים.
* Pt , מופע של המחלקה OpeningScreen, אשר מאפשרת גישה למסכי המשחק.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| שם הפונקציה | מה הפונקציה מבצעת | מה הפונקציה מקבלת | מה הפונקציה מחזירה | סיבוכיות |
| init | מאתחלת מופע של ה Board ושל ה OpeningScreen. |  |  | O(1) |
| Start\_Game | מעלה את מסך הפתיחה של המשחק. | Stage, הראשי איתו נפעיל את המסך. |  | O(1) |
| setExitButton | יוצאת ממסך המשחק |  |  | O(1) |
| transferToBoard | מעביר למסך של משחק אנושי מול אנושי. | Stage, הראשי איתו נפעיל את המסך. |  | O(1) |
| transferToAI | מעביר למסך של משחק אנושי מול ממוחשב. | Stage, הראשי איתו נפעיל את המסך. |  | O(1) |
| transferToOpeningScreen | מעביר למסך הפתיחה. | Stage, הראשי איתו נפעיל את המסך. |  | O(1) |
| transferToInstructions | מעביר למסך ההוראות- חוקי המשחק. | Stage, הראשי איתו נפעיל את המסך. |  | O(1) |
| addTool | מוסיפה חייל ללוח, עבור שחקן אנושי, ובודקת ניצחון או תיקו. מוסיפה גם ויזואלית וגם ללוח. | Turn, תור של השחקן הנוכחי.  B, כפתור שעליו לחץ השחקן.  Check,  מאפשר לשחק רק אם השחקן הקודם סובב. | מחזירה מחרוזת, אשר יכולה להכיל את השחקן המנצח, תיקו או error במידה ומהלך לא חוקי. | O(1) |
| rotateBoard | הפונקציה מבצעת סיבוב של מיני הלוח, עבור שחקן אנושי. | B, כפתור הסיבוב עליו לחץ השחקן.  Bool, נותן אינדיקציה אם כבר נעשה סיבוב.  Buttons, מערך כפתורי הסיבוב. | מחזירה שקר אם סובבה את המיני לוח. | O(n^2) |
| addToolAI | מאפשר הוספת חייל ללוח, עבור משחק של שחקן אנושי מול ממוחשב. בנוסף, בודקת ניצחון או תיקו. | Turn, תור של השחקן הנוכחי.  B, כפתור שעליו לחץ השחקן.  Check,  מאפשר לשחק רק אם השחקן הקודם סובב. | מחזירה מחרוזת, אשר יכולה להכיל את השחקן המנצח, תיקו או error במידה ומהלך לא חוקי. | O(1) |
| rotateBoardAI | הפונקציה מאפשרת סיבוב של מיני הלוח, עבור שחקן ה AI. | numBoard, מיני הלוח הנועד לסיבוב.  boardRotate, כיוון הסיבוב של מיני הלוח.  Bool, נותן אינדיקציה אם כבר נעשה סיבוב.  Buttons, מערך כפתורי הסיבוב. | מחזירה שקר אם סובבה את המיני לוח. | O(n^2) |

פסיאודו קוד עבור פונקציית addToolAI:

1. אם השחקן הקודם סיים את תורו(שם חייל וסובב):
   1. אם יש תיקו:
      1. החזר "תיקו"
   2. אחרת אם יש ניצחון:
      1. החזר "שחקן x ניצח" בהתאם לבדיקת השחקן הנוכחי.
   3. אם צבע הכפתור שקוף(אין חייל מונח עליו):
      1. בהתאם לשחקן הנוכחי, קבע את צבע הכפתור.
      2. שנה את התור הנוכחי.
      3. הוסף את החייל למודל, כלומר ללוח עצמו בביטים.
   4. אחרת החזר "טעות", כיוון שהמיקום כבר תפוס.
2. החזר ""

פסיאודו קוד עבור פונקציית rotateBoard:

1. אם השחקן הנוכחי עדיין לא סובב את הלוח:
   1. קבע בהתאם ל id של הכפתור שנלחץ, מהו מיני הלוח אותו מסובבים.
   2. קבע בהתאם ל id של הכפתור שנלחץ, לאיזה כיוון נסובב את הלוח.
   3. סובב את הלוח, וקבע את השחקנים בהתאם.
   4. מיין את מערך הכפתורים של החיילים לפי id.
   5. עבור על כל הכפתורים במיני הלוח שנבחר לסיבוב:
      1. קבע את צבע הכפתור בהתאם ללוחות השחקנים.
      2. עבור לביט הבא בלוח.
   6. החזר שקר, על מנת שבפעם הבאה שתזומן הפונקציה יירשם שהשחקן סובב ולכן לא יוכל לסובב שוב.

פסיאודו קוד עבור פונקציית rotateBoardAI:

1. אם השחקן הנוכחי עדיין לא סובב את הלוח:
   1. סובב את הלוח, וקבע את השחקנים בהתאם.
   2. מיין את מערך הכפתורים של החיילים לפי id.
   3. עבור על כל הכפתורים במיני הלוח שנבחר לסיבוב:
      1. קבע את צבע הכפתור בהתאם ללוחות השחקנים.
      2. עבור לביט הבא בלוח.
   4. החזר שקר, על מנת שבפעם הבאה שתזומן הפונקציה יירשם שהשחקן סובב ולכן לא יוכל לסובב שוב.

מחלקת AIPlayer

מחלקה זו מרכיבה את ה AI, מכילה שלושה חלקים עיקריים, אשר כל אחד מהם מחזיר ציון לפי עדיפות, אינדקס מומלץ לשים בו את החייל, איזה מיני לוח לסובב ולאיזה כיוון.

שלושת החלקים מהם מורכב ה AI:

* אסטרטגית ההגנה- defense
* אסטרטגית התקפה- the triple power play
* אסטרטגית ההתקפה- rows and columns

שדות עיקריים וכלליים:

* Grade, ציון הניתן לכל אסטרטגיה, לפיו נדע את המהלך הכי טוב הבא.
* Player1, משתנה מסוג long, המכיל את הלוח מנוקדת המבט של השחקן האנושי.
* ai, משתנה מסוג long, המכיל את הלוח מנוקדת המבט של השחקן הממוחשב.

פונקציות כלליות:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| שם הפונקציה | מה הפונקציה מבצעת | מה הפונקציה מקבלת | מה הפונקציה מחזירה | סיבוכיות |
| AiPlayer | פעולה בונה |  |  | O(1) |
| setPlayers | מעדכן את מצב השחקנים בכל תור שמשתנה. |  |  | O(1) |
| Count\_1 | הפונקציה מחשבת כמה ביטים שערכת הוא 1 קיימים במשתנה שקיבלנו. | S, משתנה מסוג מחרוזת המייצג משתנה המייצג את הלוח, אשר הומר למחרוזת. | מחזירה את מספר המופעים של הביט 1 במחרוזת. | O(n^2) |

**אסטרטגית ההגנה- defense**

הפונקציה מבצעת הגנה עבור האסטרטגיות עליהן דיברתי למעלה.

שדות עיקריים:

* Mini\_board\_for\_twist\_defense, מיני הלוח עליו ממליצה הפונקציה עבור אסטרטגיה זו.
* Index\_defense, האינדקס עליו ממליצה הפונקציה עבור אסטרטגיה זו.
* Direction\_rotating\_defense, כיוון סיבוב מיני הלוח עליו ממליצה הפונקציה עבור אסטרטגיה זו.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| שם הפונקציה | מה הפונקציה מבצעת | מה הפונקציה מקבלת | מה הפונקציה מחזירה | סיבוכיות |
| calc\_mini\_board\_diagonal\_triple | הפונקציה בודקת מה הוא מיני הלוח הרצוי לסיבוב. יחליט לפי האינדקס שיקבל, ידע לסובב מיני לוח עליו לא שם את האינדקס. | Index, האינדקס עליו המליצה הפונקציה.  Strategy\_mask, נותן אינדיקציה לגבי על איזה אלכסון מדובר מבין 4 האופציות האפשריות. | מספר מיני הלוח הרצוי לסיבוב. | O(1) |
| defense | הפונקציה על כל אחת מהאסטרטגיות ומחפשת החל משלושה מופעים של היריב, תבניות ניצחון מתאימות. כאשר חלה עדיפות כאשר קיימת תבנית אשר מכילה ארבעה מופעים. |  | מחזירה ציון מתאים- 100 אם מצאה שיש להגן או 0 אם אין צורך בהגנה. | O(n) |

פסיאודו קוד עבור פונקציית defense:

1. עבור על שלושת דרכי הניצחון של האסטרטגיה rows and columns:
   1. עבור כל דרך ניצחון, עבור על המסכות לניצחון- 4 בכל אחד:
      1. סובב את הלוחות בהתאם כך שיהיו כמו לוחות 1,2
      2. עבור כל מסכת ניצחון, יש 10 דרכים לנצחה כולל סיבובים:
         1. אם קיים 4 חיילים משל השחקן האנושי ואפס ביטים משל הממוחשב
            1. עבור על כל אחד מהביטים במסכה בדוק אם קיים של חייל.
            2. קבע בהתאם את האינדקס, מיני הלוח וכיוון הסיבוב.
            3. החזר 100.
2. עבור על שלושת דרכי הניצחון של האסטרטגיה rows and columns:
   1. עבור כל דרך ניצחון, עבור על המסכות לניצחון- 4 בכל אחד:
      1. סובב את הלוחות בהתאם כך שיהיו כמו לוחות 1,2
      2. עבור כל מסכת ניצחון, יש 10 דרכים לנצחה כולל סיבובים:
         1. אם קיים 3 חיילים משל השחקן האנושי ואפס ביטים משל הממוחשב
            1. עבור על כל אחד מהביטים במסכה בדוק אם קיים של חייל.
            2. קבע בהתאם את האינדקס, מיני הלוח וכיוון הסיבוב.
3. אם לא היה צורך בהגנה עד עכשיו:
   1. עבור על דרך ניצחון של אלכסונים, ועשה זאת עבור אלכסונים מימין ואלכסונים משמאל:
      1. עבור על שני האלכסונים בכל צד:
         1. עבור על כל המסכות האפשריות לניצחון:
            1. אם קיימים 4 חיילים משל השחקן האנושי ואפס משל הממוחשב

עבור על כל אחד מהביטים במסכה ובדוק אם קיים חייל.

קבע בהתאם את האינדקס, מיני לוח וכיוון סיבוב.

החזר 100.

* 1. עבור על דרך ניצחון של אלכסונים, ועשה זאת עבור אלכסונים מימין ואלכסונים משמאל:
     1. עבור על שני האלכסונים בכל צד:
        1. עבור על כל המסכות האפשריות לניצחון:
           1. אם קיימים 3 חיילים משל השחקן האנושי ואפס משל הממוחשב:

עבור על כל אחד מהביטים במסכה ובדוק אם קיים חייל.

קבע בהתאם את האינדקס, מיני לוח וכיוון סיבוב.

1. אם לא היה צורך בהגנה עד עכשיו:
   1. עבור על דרך הניצחון של the triple power play:
      1. עבור על כל אחת מהמסכות האפשריות לניצחון:
         1. אם לא קיים חייל מהשחקן הממוחשב שמגן על מסיכה, נחפש הגנה
            1. עבור על המסכות האפשריות לניצחון(4):

בדוק אם קיימים שלושה חיילים ומעלה של השחקן האנושי ואפס של הממוחשב ובהתאם לכך תקבע אינדקסים. (תחילה בדיקה במיני הלוח הראשון) ולכן יתבצע פעמיים.

אם לא מצאת אינדקס מתאים, חפש במיני הלוח השני, ולכן יחזור פעם אחת.

אם לא מצאת אינדקס, חפש במיני הלוח השלישי, ולכן יחזור פעמיים.

1. אם היה צורך בהגנה:
   1. החזר 100
2. אחרת:
   1. החזר 0

**אסטרטגית התקפה- the triple power play**

הפונקציה מבצעת התקפה עבור אסטרטגיה זו.

שדות עיקריים:

* Mini\_board\_for\_twist\_triple, מיני הלוח עליו ממליצה הפונקציה עבור אסטרטגיה זו.
* Index\_triple, האינדקס עליו ממליצה הפונקציה עבור אסטרטגיה זו.
* Direction\_rotating\_triple, כיוון סיבוב מיני הלוח עליו ממליצה הפונקציה עבור אסטרטגיה זו.
* Player\_move, מספר מהלך שבו משחק השחקן באסטרטגיה זו.
* Current\_strategy, האסטרטגיה הנוכחית בה משתמש, כלומר אלכסון מצד ימין או שמאל. כאשר 1 משמע אלכסון שמאלי, ו 2 משמע אלכסון ימני.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| שם הפונקציה | מה הפונקציה מבצעת | מה הפונקציה מקבלת | מה הפונקציה מחזירה | סיבוכיות |
| triple\_power\_play | הפונקציה הראשית של האסטרטגיה, נועדה לסווג בין הפונקציות לפי ידיעת מספר המהלך (player\_move). |  | מחזירה את הציון עבור האסטרטגיה- 80, עבור מהלך מנצח, ו 0 אם כל ארבע האופציות נפסלו. | O(n^2) |
| player\_move\_1 | הפונקציה מציעה את האינדקס המתאים, מיני הלוח לסובב ואת כיוון הסיבוב המתאים, עבור השמה של החייל הראשון באסטרטגיה. | Playing\_strategy, אשר אחראי לסווג את סוג האלכסון עליו עובדים- ימני או שמאלי. |  | O(1) |
| player\_move\_2 | הפונקציה מציעה את האינדקס המתאים, מיני הלוח לסובב ואת כיוון הסיבוב המתאים, עבור השמה של החייל השני באסטרטגיה. |  |  | O(1) |
| player\_move\_3 | הפונקציה מציעה את האינדקס המתאים, מיני הלוח לסובב ואת כיוון הסיבוב המתאים, עבור השמה של החייל השלישי באסטרטגיה. |  |  | O(1) |
| player\_move\_above\_3 | הפונקציה מציעה את האינדקס המתאים, מיני הלוח לסובב ואת כיוון הסיבוב המתאים, עבור השמה של החייל הרביעי ואילך באסטרטגיה. |  |  | O(n^2) |
| check\_for\_lose | בודק אם ישנו הפסד באלכסון הספציפי, אם כן הוא עובד לאלכסון השני, ואם שתי האסטרטגיות סיימו, מעדכנת את הציוון להיות 0. בנוסף, בודק אם יש ניצחון מידי – ארבע ברצף, ומעדכן את הציון להיות 120.  ההפסד נקבע לפי זוג של כדורי היריב מקבילים אחד לשני. | Strategy, אשר אחראי לסווג את סוג האלכסון עליו עובדים- ימני או שמאלי. | מחזירה אמת אם ישנו הפסד כלשהו, אחרת שקר. | O(1) |
| check\_for\_lose\_sides | בודק אם ישנו הפסד באלכסון הספציפי, אם כן הוא עובד לאלכסון השני, ואם שתי האסטרטגיות סיימו, מעדכנת את הציוון להיות 0.  ההפסד נקבע לפי ארבע כדורים של היריב המונחים בלוח האמצעי של האסטרטגיה. | Strategy, אשר אחראי לסווג את סוג האלכסון עליו עובדים- ימני או שמאלי. | מחזירה אמת אם ישנו הפסד כלשהו, אחרת שקר. | O(1) |
| start\_next\_mini\_board | לאחר ששמנו שני חיילים על הלוח, נצטרך לחפש את מיני הלוח השני בו נשים את הכדורים. זוהי מטרתה של הפונקציה. | Playing\_strategy, אשר אחראי לסווג את סוג האלכסון עליו עובדים- ימני או שמאלי.  Start\_index, האינדקס הראשוני, ממנו נתחיל לחפש מיקום- בהתאם למיני הלוח.  End\_index, האינדקס הסופי, ממנו נסיים לחפש מיקום. |  | O(1) |
| find\_index | פונקציה זו מחפשת את האינדקס הבא שיוביל לניצחון. | Mini\_board, מיני לוח שייתן לנו אינדיקציה עם איזו מסיכה לעבוד. | מחזירה את האינדקס המומלץ. | O(n^2) |
| find\_rotate | פונקציה זו מחפשת את מיני הלוח הבא שיוביל לניצחון. ואת כיוונו. | Mini\_board, מיני לוח שייתן לנו אינדיקציה עם איזו מסיכה לעבוד. |  | O(1) |
| my\_color\_ball | מחפש לנו את המהלך הבא לביצוע, לאחר ידיעה שיש זוג במיני הלוח הנוכחי שעומד במסיכת הניצחון. |  |  | O(n^2) |

פסיאודו קוד עבור פונקציית check\_for\_lose:

1. בהתאם לאסטרטגיה שנבחרה(אלכסונים ימניים או שמאליים), עבור על שתי המסכות המתאימות לה:
   1. אם קיים מצב שבו לשחקן הממוחשב יש 4 חיילים על הלוח ולשחקן האנושי חייל 1:
      1. קבע שהציון הוא 120. (ניצחון בטוח)
2. אם קיימים שני חיילים של השחקן האנושי שחוסמים את שני מסכות הניצחון בכל דרך שהיא(שני חיילים מקבילים):
   1. קבע שהציון הוא 0 (האסטרטגיה נפסלה).
   2. החזר אמת.
3. אם האסטרטגיה היא האלכסון מימין:
   1. אם קיים בה הפסד:
      1. קבע שהאסטרטגיה הנוכחית היא כעת אלכסון משמאל.
      2. קבע שהשלב הנוכחי הוא כעת החייל הראשון ששים על הלוח.
      3. זמן את פעולת החייל הראשון ששמים על הלוח.
      4. החזר אמת.
4. אחרת אם האסטרטגיה היא האלכסון משמאל:
   1. אם קיים בה הפסד:
      1. קבע שהאסטרטגיה הנוכחית היא כעת אלכסון מימין.
      2. קבע שהשלב הנוכחי הוא כעת החייל הראשון ששים על הלוח.
      3. זמן את פעולת החייל הראשון ששמים על הלוח.
      4. החזר אמת.
5. החזר שקר.

פסיאודו קוד עבור פונקציית check\_for\_lose\_sides:

1. אם קיימים ארבעה חיילים של השחקן האנושי שחוסמים את שני מסכות הניצחון בכל דרך שהיא(ארבעה חיילים המתאימים למיני הלוח השני עם חייל אחד בלבד):
   1. קבע שהציון הוא 0 (האסטרטגיה נפסלה).
   2. החזר אמת.
2. אם האסטרטגיה היא האלכסון מימין:
   1. אם קיים בה הפסד:
      1. קבע שהאסטרטגיה הנוכחית היא כעת אלכסון משמאל.
      2. קבע שהשלב הנוכחי הוא כעת החייל הראשון ששים על הלוח.
      3. זמן את פעולת החייל הראשון ששמים על הלוח.
      4. החזר אמת.
3. אחרת אם האסטרטגיה היא האלכסון משמאל:
   1. אם קיים בה הפסד:
      1. קבע שהאסטרטגיה הנוכחית היא כעת אלכסון מימין.
      2. קבע שהשלב הנוכחי הוא כעת החייל הראשון ששים על הלוח.
      3. זמן את פעולת החייל הראשון ששמים על הלוח.
      4. החזר אמת.
4. החזר שקר.

פסיאודו קוד עבור פונקציית player\_move\_1:

1. אם לא חל הפסד באחת מהאסטרטגיות:
   1. המשך כל עוד לא מצאת מקום ריק- עבור על כל האינדקסים האפשריים למהלך הראשון(אחד משמונה המקומות למיני לוחות עם שני חיילים עליהם- הוסבר בתיאור האלגוריתם):
      1. בחר אינדקס רנדומלי
      2. אם עובדים על אלכסונים מימין:

1.1.2.1. אם המקום לא תפוס על ידי אחד מהשחקנים:

1.1.2.1.1 קבע אינדקס מתאים, סיבוב מתאים ומיני לוח לסובב.

1.1.2.1.2 הגדר כי מצאנו מקום פנוי.

1.1.2.2. אם המקום כן תפוס:

1.1.2.2.1 הגדר כי כעת עובדים על אופציה אחת פחות.

* + 1. אם עובדים על אלכסונים משמאל:

1.1.3.1. אם המקום לא תפוס על ידי אחד מהשחקנים:

1.1.3.1.1 קבע אינדקס מתאים, סיבוב מתאים ומיני לוח לסובב.

1.1.3.1.2 הגדר כי מצאנו מקום פנוי.

1.1.3.2. אם המקום כן תפוס:

1.1.3.2.1 הגדר כי כעת עובדים על אופציה אחת פחות.

פסיאודו קוד עבור פונקציית player\_move\_2:

1. אם לא חל הפסד באחת האסטרטגיות:
   1. עבור על כל האינדקסים האפשריים למהלך הראשון(אחד משמונה המקומות למיני לוחות עם שני חיילים עליהם- הוסבר בתיאור האלגוריתם):
      1. אם המקום אינו תפוס באופציה הטובה יותר למיקום(מימינו או משמאלו):
         1. קבע בהתאם אינדקס מתאים, מיני לוח מתאים וכיוון סיבוב.
      2. אחרת אם המקום אינו תפוס באופציה הפחות טובה למיקום(ממולו):
         1. קבע בהתאם אינדקס מתאים, מיני לוח מתאים וכיוון סיבוב.
      3. עבור לאינדקס הבא.
   2. עבור על כל האינדקסים האפשריים למהלך השני(אחד משמונה המקומות למיני לוחות עם שני חיילים עליהם- הוסבר בתיאור האלגוריתם):
      1. אם המקום אינו תפוס באופציה הטובה יותר למיקום(מימינו או משמאלו):
         1. קבע בהתאם אינדקס מתאים, מיני לוח מתאים וכיוון סיבוב.
      2. אחרת אם המקום אינו תפוס באופציה הפחות טובה למיקום(ממולו):
         1. קבע בהתאם אינדקס מתאים, מיני לוח מתאים וכיוון סיבוב.
      3. עבור לאינדקס הבא.

פסיאודו קוד עבור פונקציית player\_move\_3:

1. אם לא חל הפסד באחת האסטרטגיות:
   1. עבור על כל מיקומי מיני הלוח הראשון- האפשריים למיקום חייל ראשון ושני- במטרה להבין באיזה מיני לוח מתאים לשים את האינדקסים:
      1. אם קיים חייל ממוחשב:
         1. סמן כי קיים חייל.
      2. עבור לחייל הבא.
   2. בדוק אם היה קיים חייל במיני הלוח הראשון:
      1. קרא לפונקציה start\_next\_mini\_board, יחד עם מיני הלוח הבא שיש לשים בו חיילים, אשר תמצא את האינדקס הנכון הבא, יחד עם מיני הלוח לסיבוב וכיוון הסיבוב.
   3. עבור על כל מיקומי מיני הלוח השני- האפשריים למיקום חייל ראשון ושני- במטרה להבין באיזה מיני לוח מתאים לשים את האינדקסים:
      1. אם קיים חייל ממוחשב:
         1. סמן כי קיים חייל.
      2. עבור לחייל הבא.
   4. בדוק אם היה קיים חייל במיני הלוח הראשון:
      1. קרא לפונקציה start\_next\_mini\_board, יחד עם מיני הלוח הבא שישי לשים בו חיילים, אשר תמצא את האינדקס הנכון הבא, יחד עם מיני הלוח לסיבוב וכיוון הסיבוב.

פסיאודו קוד עבור פונקציית player\_move\_above\_3:

1. אם לא חל הפסד באחת האסטרטגיות:
   1. אם מדובר באסטרטגיה של אלכסון מימין(עשה אותו דבר אחר כך עבור אלכסון משמאל):
      1. עבור על כל המיקומים האפשריים למקם במיני הלוח השני:
         1. אם קיים חייל ממוחשב:
            1. אם קיים במיקום הבא הטוב שרצית לשים חייל אנושי:

אם ריק, במיקום הפחות טוב הבא שרצית לשים:

קבע במקום זה את האינדקס, את מיני הלוח המתאים ואת סיבובו.

אחרת אם קיים חייל ממוחשב במיקום הפחות טוב הבא שרצית לשים:

קרא לפונקציה my\_color\_ball(), כלומר כדור שהוא בצבע שלי- היא תמליץ להתחיל לשים במיני בלוח השלישי, ותקבע בהתאם את מיני הלוח ואת כיוונו ואת האינדקס המתאים.

* + - * 1. אחרת אם קיים במיקום הבא הטוב שרצית לשים חייל ממוחשב:

קרא לפונקציה my\_color\_ball(), כלומר כדור שהוא בצבע שלי- היא תמליץ להתחיל לשים במיני בלוח השלישי, ותקבע בהתאם את מיני הלוח ואת כיוונו ואת האינדקס המתאים.

* + - * 1. אחרת=ריק:

אם במיקום הבא הפחות טוב שרצית לשים, קיים חייל אנושי או ריק:

קבע במקום זה את האינדקס, את מיני הלוח המתאים ואת סיבובו.

אחרת=חייל ממוחשב:

קרא לפונקציה my\_color\_ball(), כלומר כדור שהוא בצבע שלי- היא תמליץ להתחיל לשים במיני בלוח השלישי, ותקבע בהתאם את מיני הלוח ואת כיוונו ואת האינדקס המתאים.

פסיאודו קוד עבור פונקציית triple\_power\_play:

1. אם התור הנוכחי הוא 1:
   1. בחר ברנדומליות אחת מהאסטרטגיות- אלכסון מימין או משמאל.
   2. זמן את פונקציה player\_move\_1()
2. אחרת אם התור הנוכחי הוא 2:
   1. זמן את פונקציה player\_move\_2()
3. אחרת אם התור הנוכחי האי 3:
   1. זמן את פונקציה player\_move\_3()
4. אחרת אם התור הנוכחי מעל 3:
   1. זמן את פונקציה player\_move\_above\_3()
5. החזר את הציון.

**אסטרטגית ההתקפה- rows and columns**

הפונקציה מבצעת התקפה עבור אסטרטגיה זו.

שדות עיקריים:

* mini\_board\_for\_twist\_rows\_columns, מיני הלוח עליו ממליצה הפונקציה עבור אסטרטגיה זו.
* Index\_rows\_columns, האינדקס עליו ממליצה הפונקציה עבור אסטרטגיה זו.
* Direction\_rotating\_rows\_columns, כיוון סיבוב מיני הלוח עליו ממליצה הפונקציה עבור אסטרטגיה זו.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| שם הפונקציה | מה הפונקציה מבצעת | מה הפונקציה מקבלת | מה הפונקציה מחזירה | סיבוכיות |
| repeat | הפונקציה מחפשת את האינדקס הבא המתאים, את מיני הלוח ואת סיבובו, נועדה לצמצום קוד. | Check, בודק אם כבר מצאנו אינדקס או לא.  I, איזה מסיכה מתאימה עובדים.  Keep\_ai, לוח לפי שחקן ממוחשב.  Keep\_player1, לוח לפי שחקן 1  X, נותן אינדיקציה למיני הלוח איתו עובדים.  B, מופע של המחלקה Board.  R, עבור איזה שורות או עמודות עובדים. | מחזירה אמת אם מצאה מיקום, אחרת שקר. | O(1) |
| Rows\_columns | הפונקציה הראשית של האסטרטגיה, נועדה להחליט את האינדקס הכי מתאים לצעד הבא, את מיני הלוח הכי מתאים ואת סיבובו. |  | מחזירה את הציון עבור האסטרטגיה במידה ולא עובר במרכז- 60, ובמידה וכן עובר במרכז- 40, עבור מהלך מנצח. | O(n) |

פסיאודו קוד עבור פונקציית rows\_columns:

1. עבור על שלושת דרכי הניצחון של האסטרטגיה rows and columns:
   1. עבור כל דרך ניצחון, עבור על המסכות לניצחון- 4 בכל אחד:
      1. סובב את הלוחות בהתאם כך שיהיו כמו לוחות 1,2
      2. עבור כל מסכת ניצחון, יש 10 דרכים לנצחה כולל סיבובים:
         1. אם קיים 4 חיילים משל השחקן האנושי ואפס חיילים משל הממוחשב
            1. עבור על כל אחד מהביטים במסכה בדוק אם קיים של חייל.
            2. קבע בהתאם את האינדקס, מיני הלוח וכיוון הסיבוב.
            3. החזר 120(ניצחון בטוח).
2. במידה אין ארבעה חיילים, עבור על שלושת דרכי הניצחון של האסטרטגיה rows and columns:
   1. עבור כל דרך ניצחון, עבור על המסכות לניצחון- 4 בכל אחד:
      1. סובב את הלוחות בהתאם כך שיהיו כמו לוחות 1,2
      2. עבור כל מסכת ניצחון, יש 10 דרכים לנצחה כולל סיבובים:
         1. אם קיים 3 חיילים משל השחקן האנושי ופחות משני חיילים משל הממוחשב
            1. עבור על כל אחד מהביטים במסכה בדוק אם קיים של חייל.
            2. קבע בהתאם את האינדקס, מיני הלוח וכיוון הסיבוב.
            3. אם מדובר באסטרטגיה שעובר במרכזים:

החזר 40

* + - * 1. אחרת:

החזר 60

1. במידה אין ארבעה או שלושה חיילים, עבור על שלושת דרכי הניצחון של האסטרטגיה rows and columns:
   1. עבור כל דרך ניצחון, עבור על המסכות לניצחון- 4 בכל אחד:
      1. סובב את הלוחות בהתאם כך שיהיו כמו לוחות 1,2
      2. עבור כל מסכת ניצחון, יש 10 דרכים לנצחה כולל סיבובים:
         1. אם קיים 2 חיילים משל השחקן האנושי ופחות משני חיילים משל הממוחשב
            1. עבור על כל אחד מהביטים במסכה בדוק אם קיים של חייל.
            2. קבע בהתאם את האינדקס, מיני הלוח וכיוון הסיבוב.
            3. אם מדובר באסטרטגיה שעובר במרכזים:

החזר 40

* + - * 1. אחרת:

החזר 60

1. במידה אין ארבעה או שלושה או שניים חיילים, עבור על שלושת דרכי הניצחון של האסטרטגיה rows and columns:
   1. עבור כל דרך ניצחון, עבור על המסכות לניצחון- 4 בכל אחד:
      1. סובב את הלוחות בהתאם כך שיהיו כמו לוחות 1,2
      2. עבור כל מסכת ניצחון, יש 10 דרכים לנצחה כולל סיבובים:
         1. אם קיים 1 חיילים משל השחקן האנושי ופחות משני חיילים משל הממוחשב
            1. עבור על כל אחד מהביטים במסכה בדוק אם קיים של חייל.
            2. קבע בהתאם את האינדקס, מיני הלוח וכיוון הסיבוב.
            3. אם מדובר באסטרטגיה שעובר במרכזים:

החזר 40

* + - * 1. אחרת:

החזר 60

1. במידה ואין כלל חיילים, עבור על שלושת דרכי הניצחון של האסטרטגיה rows and columns:
   1. עבור כל דרך ניצחון, עבור על המסכות לניצחון- 4 בכל אחד:
      1. סובב את הלוחות בהתאם כך שיהיו כמו לוחות 1,2
      2. עבור כל מסכת ניצחון, יש 10 דרכים לנצחה כולל סיבובים:
         1. אם קיים 0 חיילים משל השחקן האנושי ופחות משני חיילים משל הממוחשב
            1. עבור על כל אחד מהביטים במסכה בדוק אם קיים של חייל.
            2. קבע בהתאם את האינדקס(יפעל למרכזים בכל אחד), מיני הלוח וכיוון הסיבוב.
            3. אם מדובר באסטרטגיה שעובר במרכזים:

החזר 40

* + - * 1. אחרת:

החזר 60

1. החזר ציון סופי.

מחלקת TheBoard

מחלקה זו אחראית ליצירת הוויזואליות של לוח המשחק, וניהול כל ה events במשחק.

שדות עיקריים:

* controller, משתנה המאפשר גישה אל ה controller- כמו חזרה לתפריט הראשי או יציאה מהמסך.
* rotateOnce, משתנה המאפשר לכל שחקן לסובב מיני לוח רק פעם אחת.
* forceRotating, משתנה המחייב סיבוב של מיני לוח לפני העברה לשחקן הבא.
* Error, משתנה המאפשר ביצוע של מהלכים חוקיים בלבד.
* Buttons, מערך כפתורים המחזיקים את המקומות האפשריים לשים את החיילים בלוח – 36 כפתורים.
* Buttons\_rotate, מערך כפתורים המחזיקים את הסיבוב ימינה ושמאלה לכל מיני לוח – 2x4 = 8 כפתורים.
* Mini\_board, מיני הלוח עליו המליץ ה AI לסובב
* Index, האינדקס עליו המליץ ה AI להניח את החייל
* Rotating, כיוון סיבוב מיני הלוח עליו המליץ ה AI
* La, תווית אשר מציגה תור שחקן נוכחי, כולל הצגת ניצחון או תיקו.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| שם הפונקציה | מה הפונקציה מבצעת | מה הפונקציה מקבלת | מה הפונקציה מחזירה | סיבוכיות |
| TheBoard | פעולה בונה |  |  | O(1) |
| endGame | אחראית על הצגת השחקן שניצח ואחראית על ביטול הכפתורים למשחק.  אם לא ניצח, מעבירה לפונקציה אחרת אם בוצע מהלך לא חוקי. | S, מחרוזת המציינת ניצחון או אם בוצעה טעות.  B, כפתורים של הלוח, על מנת לבטלם.  B2, כפתורים של הסיבוב על מנת לבטלם. | מחזירה אמת אם נעשה מהלך לא חוקי, אחרת מחזירה שקר. | O(n^2) |
| initialize\_the\_board | הפונקציה מאתחלת את הלוח- מבחינה ויזואלית. | Stage, המרכזי איתו נפעיל את הלוח. |  | O(n^2) |
| start\_rotate\_buttons | הפונקציה אחראית על אתחול כפתורי הסיבוב ימינה ושמאלה לכל מיני לוח. | Stage, המרכזי איתו נפעיל את הכפתורים.  boarderPane, המרכזי אליו נשייך את הכפתורים. |  | O(1) |
| Ai\_play | הפונקציה מנהלת את המשחק מפניו של ה AI, כלומר היא מזמנת את הפונקציות לבדיקת אסטרטגיה, ובוחרת את הציון הכי גבוה, ולפי ההמלצה שמה חייל ומסובבת מיני לוח. | A, מופע של החלקה AIPlayer אשר אחראית על ביצוע פעולות האסטרטגיה והחזרת המלצה. |  | O(n^2) |
| start | הפונקציה מנהלת את כל המשחק, אחראית להחליף בין תורות, לזמן את ה AI וגם לנהל משחק של שני שחקנים אנושיים. הוא אחראי לשינויי הלוח בהתאם לבחירת המשתמש. | primaryStage, המרכזי איתו נפעיל את המשחק. |  | O(n^2) |

פסיאודו קוד עבור פונקציית ai\_play:

1. במידה ותורו של השחקן הממוחשב:
   1. קבע את השחקנים במחלקת AiPlayer להיות מסונכרנים עם לוחות השחקנים.
   2. הגדל את מספר התור שתבצע- רלוונטי עבור פעולת triple\_power\_play.
   3. קרא לפונקציה triple\_power\_play שבמחלקת ה Ai, ושמור במשתנה את הציון שהיא נתנה.
   4. קרא לפונקציה defense שבמחלקת ה Ai, ושמור במשתנה את הציון שהיא נתנה.
   5. קרא לפונקציה rows\_columns שבמחלקת ה Ai, ושמור במשתנה את הציון שהיא נתנה.
   6. בדוק מי מהפונקציות נתנה ציון הכי גבוהה, ובהתאם לכך קבע את האינדקס הנוכחי, מיני הלוח לסיבוב וכיוונו, בהתאם להמלצתה.
   7. הוסף את הכלי ללוח, בהתאם להמלצה.
   8. אם אינו חל ניצחון:
      1. אם חל מהלך חוקי:
         1. שנה את תור השחקן הנוכחי לשחקן האנושי.
   9. בהתאם למיני הלוח שקיבלת לסובב ולכיוונו, חפש את הכפתור הנכון המתאים לבקשה.
   10. אם חל מהלך חוקי:
       1. סובב את מיני הלוח בהתאם להמלצה.

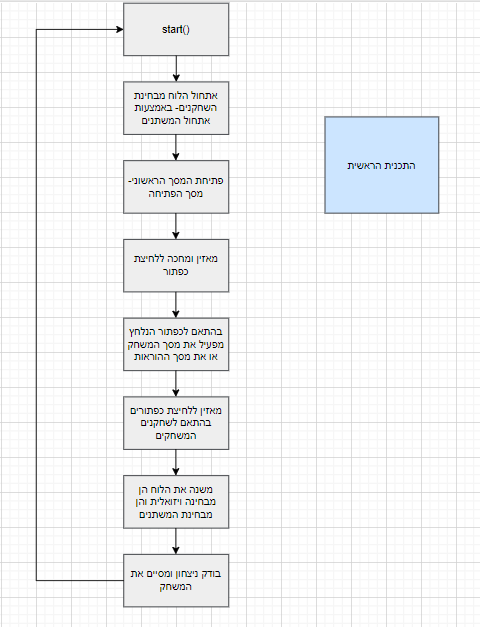
פסיאודו קוד עבור פונקציית start:

1. אתחל את כפתורי הלוח.
2. אם המשחק הוא שחקן אנושי מול אנושי:
   1. עבור על כל כפתורים החיילים:
      1. הגדר שבעת לחיצת כפתור, יתווסף החייל ללוח של השחקן הנוכחי.
      2. בנוסף הגדר שאם המהלך היה חוקי:
         1. שנה את תור חייל הנוכחי.
   2. עבור על כל כפתורי הסיבוב:
      1. הגדר שבעת לחיצת הפתור, מיני הלוח המתאים יסתובב, וישנה את לוחות שני השחקנים בהתאמה.
      2. בנוסף הגדר שאם המהלך היה חוקי:
         1. שנה את תווית השחקן הנוכחי בהתאם.
3. אם המשחק הוא שחקן אנושי מול שחקן ממוחשב:
   1. צור מופע של המחלקה AiPlayer.
   2. קרא לפונקציה ai\_play.
   3. קבע את התווית להיות תור השחקן האנושי.
   4. עבור על כל כפתורים החיילים:
      1. הגדר שבעת לחיצת כפתור, יתווסף החייל ללוח של השחקן הנוכחי.
      2. בנוסף הגדר שאם המהלך היה חוקי:
         1. שנה את תור חייל הנוכחי.
   5. עבור על כל כפתורי הסיבוב:
      1. הגדר שבעת לחיצת הפתור, מיני הלוח המתאים יסתובב, וישנה את לוחות שני השחקנים בהתאמה.
      2. שנה את תווית השחקן הנוכחי להיות ai.
      3. חכה חצי שניה וקרא לפונקציה ai\_play.
      4. חכה שניה ושנה את התווית להיות player1.

# **התכנית הראשית**

התכנית הראשית מתחילה מהרצת התכנית- מהמחלקה DemoDriver. מחלקה זו מריצה את הפונקציה start, אשר ממנה רצה כל התכנית. הפונקציה למעשה, מזמנת את הפעולה init(), אשר מאתחלת את הלוחות של השחקנים, ומאתחלת את מסך הפתיחה.

לאחר מכן, מסך הפתיחה עולה למשתמש, אשר יכול לבחור על איזה כפתור ללחוץ על מנת לשחק במשחק. ברגעים אלו, המחשב מאזין ומחכה ללחיצה של אחד הכפתורים על מנת שיוכל להגיב בהתאם. לאחר שמקבל לחיצה, מעלה את המסך המתאים בהתאם לכפתור שנלחץ, ומתחיל את המשחק. גם במשחק האנושי וגם בממוחשב, המחשבן מאזין ומחכה ללחיצה על כפתורים אשר יאפשרו הוספת חייל ללוח וסיבוב מיני לוח בהתאם. לאחר שמקבל לחיצה, משנה את הלוח בהתאם- את המשתנים של השחקנים וגם את הוויזואליות במשחק. בכל מהלך המשחק, בודק המחשב אם יש ניצחון של אחד הצדדים או תיקו, ובעת שהוא מוצא את אחד משני המצבים ההלו, הוא מסיים את המשחק. לאחר סיום המשחק, הוא מאפשר לשחקן לחזור למסך הפתיחה ולשחק מחדש.



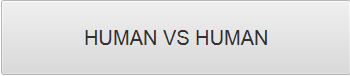
# **מדריך למשתמש**

בעת הרצת התכנית מופיע המסך הראשי:



במסך הראשי ישנן 3 אופציות בחירה:

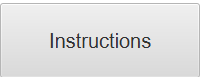
* משחק אנושי אל מול אנושי, כאשר כל שחקן משחק בתורו.



* משחק אנושי אל מול שחקן ממוחשב, כאשר כל שחקן משחק בתורו.



* דף הוראות המסביר איך משחקים במשחק.



בכל מסך מבין האופציות שציינתי ישנם שני כפתורים קבועים בהם ניתן ללחוץ בכל רגע:

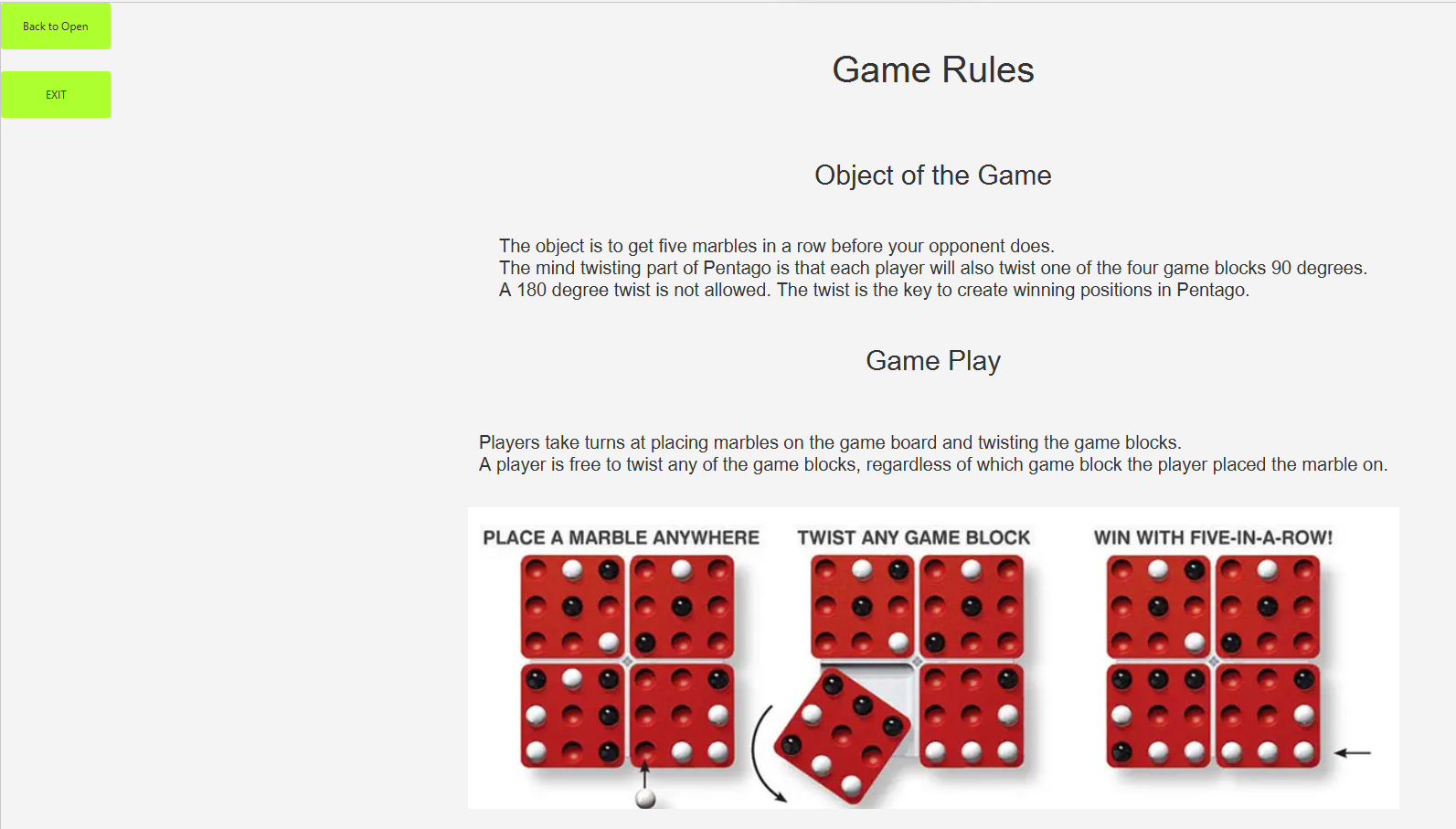
* כפתור יציאה מהמשחק.

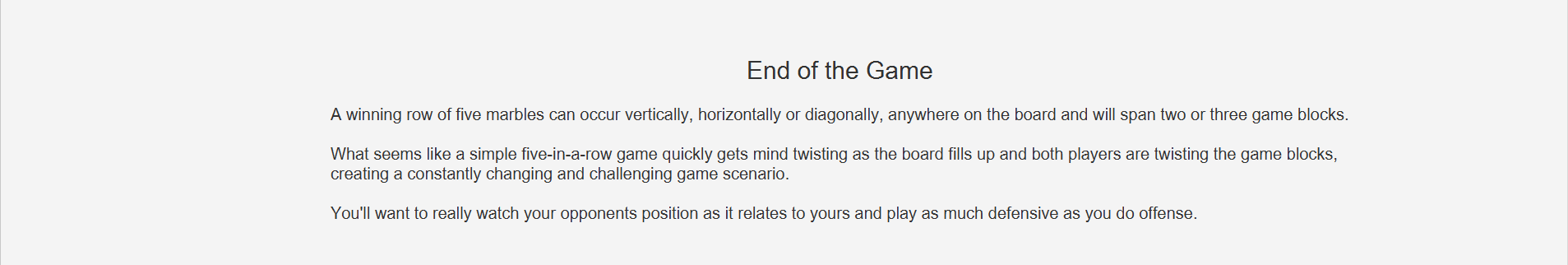


* כפתור חזרה למסך הפתיחה.

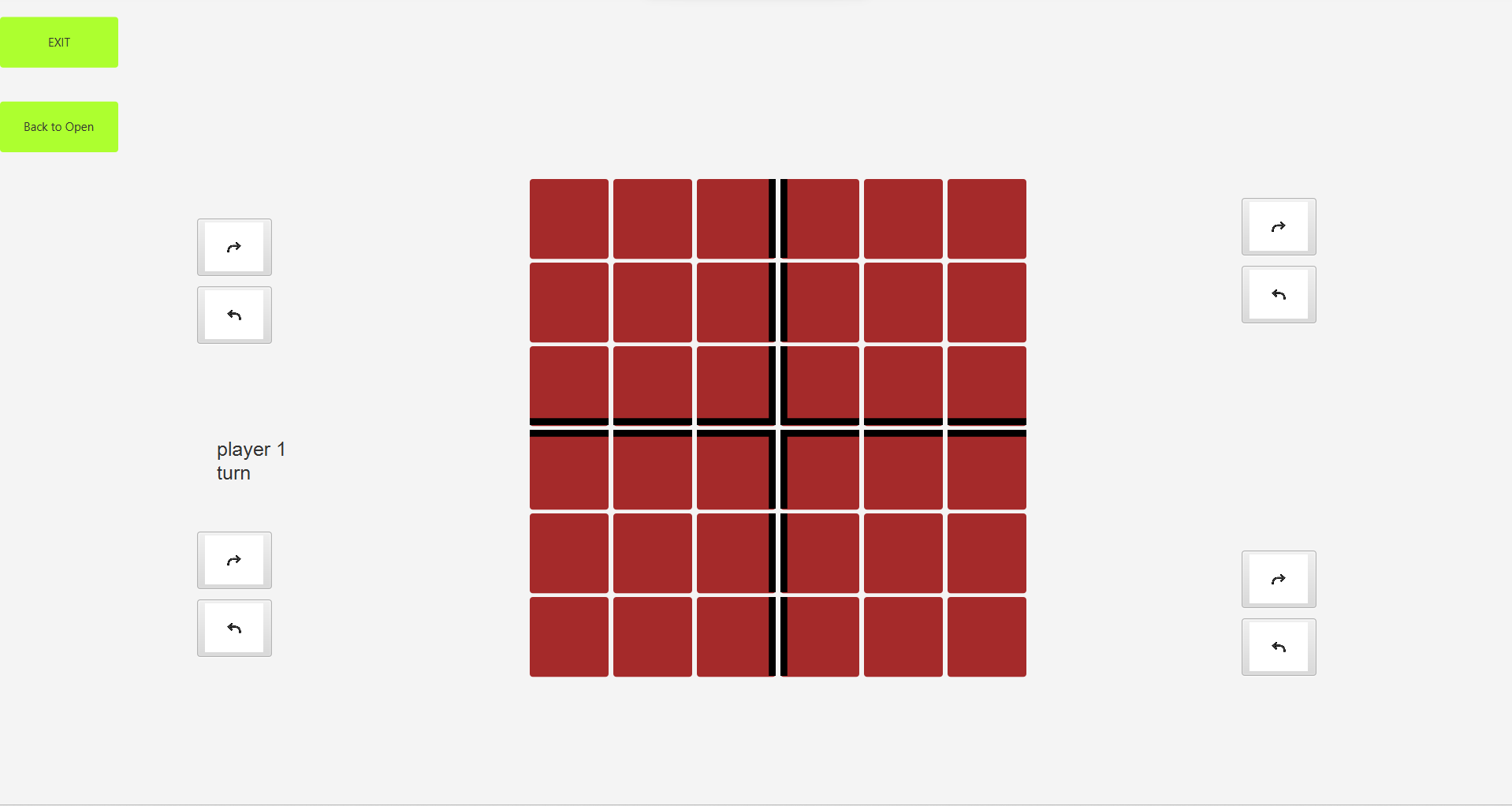


במסך ההוראות ניתן לעיין בחוקי המשחק, ומה מצופה מהשחקן לעשות:





לאחר בחירת המשתמש בסוג המשחק בו הוא מעוניין לשחק, מופיע מסך המשחק:



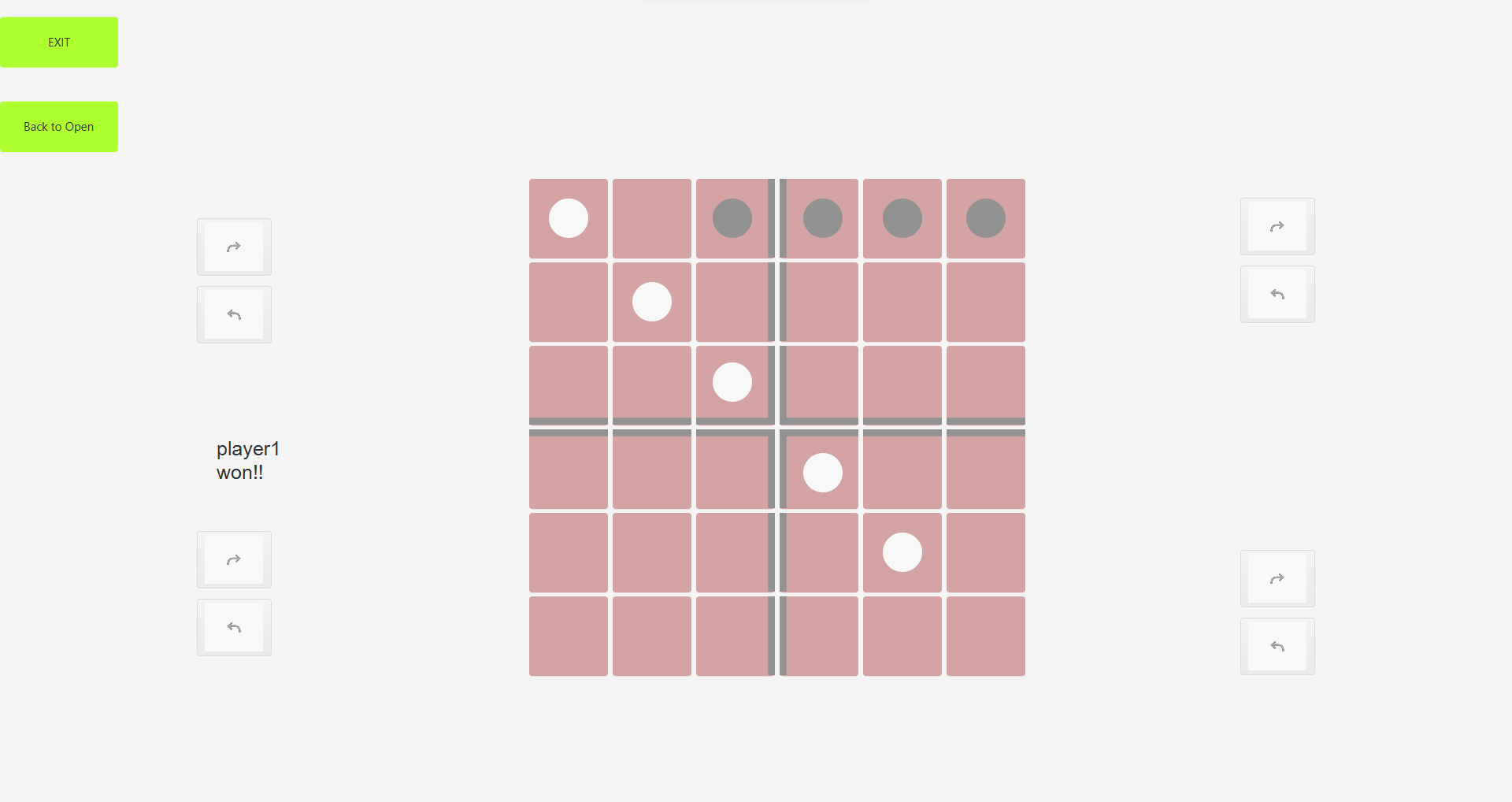
מיקומים ריקים לשים בהם חיילים

כפתורים לסיבוב מיני - לוח ימינה או שמאלה

תווית לציון תור נוכחי

כעת, על המשחקן לשים חייל על הלוח, לסובב את אחד ממיני הלוחות, ולחכות לתורו שיופיע בצד שמאל בתווית.

בעת ניצחון של אחד הצדדים, השחקן יוכל לראות מי השחקן שניצח, כאשר כל כפתורי המשחק נחסמים לשימוש מלבד כפתור היציאה או החזרה למסך הפתיחה.



תווית לציון השחקן המנצח

תווית לציון השחקן המנצח

# **סיכום אישי / רפלקציה**

העבודה על הפרויקט הייתה קשה ומאתגרת. תחילה, העבודה על הייצוג הוויזואלי הייתה מאתגרת, שכן העבודה על JAVAVFX כללה הרבה לימוד עצמי, כיוון שלפני כן לא הכרתי זאת. לאחר מכן, היה קושי בעבודה עם מבנה הנתונים שבחרתי, שאמנם מאוד יעיל אך מצריך דיוק רב ולשים לב לפרטים הקטנים. הייתי צריכה לחשוב את הדרך הכי יעילה לייצג את הלוח בביטים ולפיה לעבוד איתו.

בנוסף, העבודה על ה AI הייתה הכי מאתגרת, כיוון שהייתי צריכה לחשוב על דרך לפתור את האלגוריתם, כאשר נאסר עליי שימוש במונטה קרלו. הייתי צריכה לערוך עבודת מחקר ארוכה על מנת להבין את האסטרטגיות הקיימות, וכיצד לממש אותן בקוד. כל זאת בשילוב עם סיביות.

למרות הקשיים, אני מרגישה שלמדתי המון מהעבודה על הפרויקט, ואני שיצא לי לעבוד עליו, כיוון שלמדתי לא לוותר, ושכדי להצליח צריך לעשות עבודת מחקר גדולה. בנוסף, למדתי כיצד לעבוד עם JAVAFX, כיצד לעבוד עם סיביות ואיך לעבוד עם סביבת ה Intellij.

במידה והייתי עושה את התהליך מההתחלה היום, הייתי מתחילה קודם כל עם המחקר ורק לאחר שסיימתי אותו בצורה חלקה שאני מבינה איך הסיביות משתנות ואת האסטרטגיות להתחיל את הקוד. בנוסף, הייתי מתחילה את הפרויקט כמה שיותר מוקדם כי הייתה עליו עבודה רבה, והמון השקעה.

הרבה פעמים מצאתי את עצמי מיואשת, ומתקשה להבין איך להתקדם כל פעם מחדש. למעשה, זה היה האתגר הכי גדול עבורי, להמשיך לעבוד על הפרויקט למרות שאני מיואשת ולא תמיד מצליחה בקלות. אני מרגישה שבעקבות העבודה על הפרויקט, התחשלתי והבנתי שזה בסדר לא להצליח מיד, כי בסופו של דבר מצליחים, גם אם זה לוקח זמן.

אני שמחה שבחרתי את המשחק פנטגו, כי למרות שהצריך ממני עבודה קשה, הכרתי משחק חדש ומעניין, אשר מצריך חשיבה ברמה גבוהה. כך שגם אני אותגרתי לשחק מול המשחק הקיים.

# **ביבליוגרפיה**

* בינה מלאכותית – ויקיפדיה (2023)

[בינה מלאכותית](https://he.wikipedia.org/wiki/%D7%91%D7%99%D7%A0%D7%94_%D7%9E%D7%9C%D7%90%D7%9B%D7%95%D7%AA%D7%99%D7%AA)

* המשחק פנטגו – ויקיפדיה (2022)

[פנטגו](https://he.wikipedia.org/wiki/%D7%A4%D7%A0%D7%98%D7%92%D7%95)

* Ultra BoardGames (2020). Pentago Game Rules.

[חוקי המשחק](https://www.ultraboardgames.com/pentago/game-rules.php)

* MINDTWISTER USA (2005). BASIC PLAY, RULES AND STRATEGY GUIDE.

[אסטרטגיות פנטגו](https://webdav.info.ucl.ac.be/webdav/ingi2261/ProblemSet3/PentagoRulesStrategy.pdf)

# **נספח א – קוד הפרויקט**

## מחלקת DemoDriver:

package com.example.pentagogame;  
import com.example.pentagogame.Controller.ControllerClass;  
import javafx.application.Application;  
import javafx.stage.Stage;  
  
  
public class DemoDriver extends Application {  
 public static void main(String[] args) {  
 *launch*(args);  
 }  
  
 */\*\*  
 \* initialize the controller  
 \*  
 \* @param primaryStage- the main stage to start the game  
 \*/* @Override  
 public void start(Stage primaryStage) {  
 ControllerClass c = new ControllerClass();  
 c.init(); *//initialize the model and the view* c.start\_Game(primaryStage); *//starts the opening screen* }  
}

## מחלקת Instructions:

package com.example.pentagogame.View;  
import com.example.pentagogame.Controller.ControllerClass;  
import javafx.application.Application;  
import javafx.geometry.Pos;  
import javafx.scene.Scene;  
import javafx.scene.control.Button;  
import javafx.scene.control.Label;  
import javafx.scene.control.ScrollPane;  
import javafx.scene.image.Image;  
import javafx.scene.image.ImageView;  
import javafx.scene.layout.\*;  
import javafx.scene.text.Font;  
import javafx.stage.Screen;  
import javafx.stage.Stage;  
import java.io.File;  
  
public class Instructions extends Application {  
 private ControllerClass controller = new ControllerClass();*//connection to the controller  
  
  
 /\*\*  
 \* initialize the instructions  
 \* @param primaryStage- the stage to start the screen  
 \*/* @Override  
 public void start(Stage primaryStage) {  
 BorderPane rulesPane = new BorderPane();*//the main board* Screen screen = Screen.*getPrimary*();  
 double x = screen.getBounds().getWidth();*//the screen size* double y = screen.getBounds().getHeight();*//the screen size* VBox v = new VBox();  
 Label titleLabel = new Label("\nGame Rules");*//title* titleLabel.setFont(new Font("Arial", 40));  
 Label top1 = new Label("\n\n" + "Object of the Game\n");*//mini title* top1.setFont(new Font("Arial", 30));  
 Label rulesLabel = new Label(  
 "The object is to get five marbles in a row before your opponent does.\n" +

"The mind twisting part of Pentago is that each player will also twist one of the four game blocks 90 degrees" +  
 ".\n" + "A 180 degree twist is not allowed. The twist is the key to create winning positions in Pentago." + "\n\n\n\n\n");*//content* rulesLabel.setFont(new Font("Arial", 20));  
  
  
  
  
 Label top2 = new Label("Game Play\n");*//mini title* top2.setFont(new Font("Arial", 30));  
 Label rulesLabel2 = new Label("\nPlayers take turns at placing marbles on the game board and twisting the game blocks.\n" +  
 "A player is free to twist any of the game blocks, regardless of which game block the player placed the marble on.\n" +  
 "\n\n" +  
 "\n");*//content* rulesLabel2.setFont(new Font("Arial", 20));  
  
  
  
 File file = new File("src/main/photob/צילום מסך 2023-03-06 210855.png");  
 String filePath = file.getAbsolutePath();  
 Image i = new Image(filePath);*//adding a photo* ImageView imageView = new ImageView(i);  
  
  
  
 Label top3 = new Label("\n\n" + "End of the Game\n");*//mini title* top3.setFont(new Font("Arial", 30));  
 Label rulesLabel3 = new Label("\nA winning row of five marbles can occur vertically, horizontally or diagonally, anywhere on the board and will span two or three game blocks.\n" +  
 "\n" +  
 "What seems like a simple five-in-a-row game quickly gets mind twisting as the board fills up and both players are twisting the game blocks, " +  
 "\ncreating a constantly changing and challenging game scenario.\n" +  
 "\n" +

"You'll want to really watch your opponents position as it relates to yours and play as much defensive as you do offense.\n\n" +  
 " \n");*//content* rulesLabel3.setFont(new Font("Arial", 20));  
  
  
  
 Button backButton = new Button("Back to Open");*//back button* backButton.setOnAction(event -> {  
 controller.transferToOpeningScreen(primaryStage);*//transfers to the opening screen* });  
 backButton.setPrefSize(120, 50);  
 backButton.setStyle("-fx-background-color: greenyellow;");  
  
  
  
  
 Button exit\_button = new Button("EXIT");*//exit button* exit\_button.setPrefSize(120, 50);  
 exit\_button.setStyle("-fx-background-color: greenyellow;");  
 exit\_button.setTranslateY(25);  
 exit\_button.setOnAction(event -> {  
 controller.setExitButton();*//exits the game* });  
  
  
  
 VBox v2 = new VBox();  
 v2.getChildren().addAll(backButton, exit\_button);  
 v2.setAlignment(Pos.*TOP\_LEFT*);  
  
 v.getChildren().addAll(titleLabel, top1, rulesLabel,  
 top2, rulesLabel2, imageView, top3, rulesLabel3);  
 v.setAlignment(Pos.*TOP\_CENTER*);  
  
  
 rulesPane.setCenter(v);  
 rulesPane.setLeft(v2);

ScrollPane sp = new ScrollPane();*//scrolling thr screen* sp.setPrefSize(x, y);  
 rulesPane.setPrefSize(x, 1.5 \* y);  
 sp.setContent(rulesPane);  
  
  
 Scene scene = new Scene(sp);  
 primaryStage.setScene(scene);  
 primaryStage.show();  
  
  
 }  
}

## מחלקת OpeningScreen:

package com.example.pentagogame.View;  
import com.example.pentagogame.Controller.ControllerClass;  
import javafx.application.Application;  
import javafx.geometry.Pos;  
import javafx.scene.Scene;  
import javafx.scene.control.Button;  
import javafx.scene.control.Label;  
import javafx.scene.image.Image;  
import javafx.scene.layout.\*;  
import javafx.scene.text.Font;  
import javafx.stage.Screen;  
import javafx.stage.Stage;  
  
import java.io.File;  
  
public class OpeningScreen extends Application {  
  
 private ControllerClass controller = new ControllerClass(); *//connection to the controller* public static int *num*=0; *//0- hVSh, 1- aiVSh* public static void main(String[] args) {  
 *launch*(args);  
 }  
  
  
 */\*\*  
 \* //initializes the opening screen  
 \* @param primaryStage- the stage to start the screen  
 \*/* @Override  
 public void start(Stage primaryStage) {*//initializes the opening screen* BorderPane board = new BorderPane();*//the main board* Screen screen = Screen.*getPrimary*();  
 double x = screen.getBounds().getWidth();*//the screen size* double y = screen.getBounds().getHeight();*//the screen size* Button hVSh = new Button("HUMAN VS HUMAN");  
 hVSh.setFont(new Font("Arial", 20));  
 hVSh.setPrefSize(350, 75);  
 hVSh.setTranslateX(-200);

hVSh.setTranslateY(0);  
 hVSh.setOnAction(event -> {  
 controller.transferToBoard(primaryStage);*//transfers to the board screen* });  
  
  
 Button aiVSh = new Button("HUMAN VS AI");  
 aiVSh.setFont(new Font("Arial", 20));  
 aiVSh.setPrefSize(350, 75);  
 aiVSh.setTranslateX(-200);  
 aiVSh.setTranslateY(0);  
  
  
 aiVSh.setOnAction(event -> {  
 controller.transferToAI(primaryStage);*//transfers to the AI board screen* });  
  
  
 Button instructions = new Button("Instructions");  
 instructions.setFont(new Font("Arial", 20));  
 instructions.setPrefSize(200, 75);  
 instructions.setTranslateX(-280);  
 instructions.setTranslateY(0);  
  
  
 instructions.setOnAction(event -> {  
 controller.transferToistructions(primaryStage);*//transfers to the instructions screen* });  
  
  
 VBox v = new VBox(10); *// 10 pixels of spacing between the buttons* Label l = new Label("Welcome to PENTAGO game :)" +  
 "\nchoose one of the above options!" +  
 "\nenjoy the game!!!\n\n\n\n");  
 l.setTranslateX(-200);*//250* l.setTranslateY(0);*//450* l.setFont(new Font("Arial", 25));  
  
  
 v.getChildren().add(l);  
 v.getChildren().add(hVSh);  
 v.getChildren().add(aiVSh);  
 v.getChildren().add(instructions);  
 v.setAlignment(Pos.*TOP\_RIGHT*);  
  
 VBox v2 = new VBox();  
 Button exit\_button = new Button("EXIT");  
 exit\_button.setPrefSize(120, 50);  
 exit\_button.setStyle("-fx-background-color: greenyellow;");  
  
  
 exit\_button.setOnAction(event -> {  
 controller.setExitButton();*//exit the game* });  
  
 v2.getChildren().add(exit\_button);  
 v2.setAlignment(Pos.*TOP\_LEFT*);  
  
 board.setCenter(v);  
 board.setLeft(v2);  
  
  
 File file = new File("src/main/ph/pentago1.jpg");  
 String filePath = file.getAbsolutePath();  
 Image image = new Image(filePath);*//adding background image  
  
  
 // Set the background image* BackgroundImage backgroundImage = new BackgroundImage(image, BackgroundRepeat.*NO\_REPEAT*, BackgroundRepeat.*NO\_REPEAT*,  
 BackgroundPosition.*CENTER*, new BackgroundSize(100, 100,  
 true, true, true, false));  
  
 board.setBackground(new Background(backgroundImage));  
  
  
 Scene scene = new Scene(board, x, y);  
 primaryStage.setScene(scene);  
 primaryStage.show();  
  
  
 }  
  
}

## מחלקת TheBoard:

package com.example.pentagogame.View;  
import com.example.pentagogame.AiPlayer;  
import com.example.pentagogame.Controller.ControllerClass;  
import javafx.application.Application;  
import javafx.application.Platform;  
import javafx.geometry.Insets;  
import javafx.geometry.Pos;  
import javafx.scene.Scene;  
import javafx.scene.control.Button;  
import javafx.scene.control.Label;  
import javafx.scene.layout.\*;  
import javafx.scene.paint.Color;  
import javafx.scene.shape.\*;  
import javafx.scene.text.Font;  
import javafx.stage.Screen;  
import javafx.stage.Stage;  
import javafx.scene.image.Image;  
import javafx.scene.image.ImageView;  
import java.io.File;  
import java.util.\*;  
  
  
public class TheBoard extends Application {  
 private final int BOARD\_SIZE = 6; *//6x6* private final int MINI\_BOARD\_SIZE = 3; *//mini board size* private final int ROTATE\_BUTTONS= 8; *//number of buttons for rotate* private int turn; *//the current turn* private boolean rotateOnce; *//rotating only once* private boolean forceRotating; *//forcing to rotate* private boolean gameOver; *//checks for victory ot tie* private boolean error; *//checks for invalid moves* private Label la; *//label for current turn and winner or tie* private ControllerClass controller = new ControllerClass();*//connection to the controller* private Button[] buttons = new Button[BOARD\_SIZE\*BOARD\_SIZE]; *// create an array to store the buttons- of the board itself* private Button[] buttons\_rotate = new Button[ROTATE\_BUTTONS]; *//create an array to store the buttons- of the rotate* Integer[] rightBold = new Integer[]{3,6,24,27};*//contains buttons to bold in right side* Integer[] leftBold = new Integer[]{10,13,31,34};*//contains buttons to bold in left side* Integer[] upBold = new Integer[]{29,30,19,20};*//contains buttons to bold in up* Integer[] downBold = new Integer[]{7,8,17,18};*//contains buttons to bold in down* List<Integer> right = new ArrayList<>(Arrays.*asList*(rightBold));  
 List<Integer> left = new ArrayList<>(Arrays.*asList*(leftBold));  
 List<Integer> up = new ArrayList<>(Arrays.*asList*(upBold));  
 List<Integer> down = new ArrayList<>(Arrays.*asList*(downBold));  
  
 private final int RIGHT\_DOWN\_BOLD=9;  
 private final int RIGHT\_UP\_BOLD=21;  
 private final int LEFT\_DOWN\_BOLD=16;  
 private final int LEFT\_UP\_BOLD=28;  
  
  
 private boolean winning=false;  
 private int mini\_board;  
 private int index;  
 private int rotating;  
  
 */\*\*  
 \* initialize the board  
 \*/* public TheBoard() {  
 this.turn = 0;  
 this.rotateOnce = false;  
 this.forceRotating = true;  
 this.gameOver = false;  
 this.error = true;  
 this.la = new Label();*//the starting player* la.setPrefSize(80, 80);  
 la.setFont(new Font("Arial", 20));  
 la.setTranslateX(220);  
 la.setTranslateY(150);  
 }  
  
 */\*\*  
 \* sets the rotate once  
 \* @param rotateOnce- rotating only once  
 \*/* public void setRotateOnce(boolean rotateOnce) {  
 this.rotateOnce = rotateOnce;  
 }  
  
 */\*\*  
 \* sets the turn  
 \* @param turn- current turn to change  
 \*/* public void setTurn(int turn) {  
 this.turn = turn;  
 }  
  
 */\*\*  
 \* gets the current turn  
 \* @return thr current turn  
 \*/* public int getTurn() {  
 return turn;  
 }  
  
 public boolean getWinning() {  
 return this.winning;  
 }  
  
  
 */\*\*  
 \* sets the error  
 \* @param b- invalid move  
 \*/* public void setError(boolean b) {  
 this.error = b;  
 }  
  
 */\*\*  
 \* sets the force rotating  
 \* @param bool  
 \*/* public void setForceRotating(boolean bool) {  
 this.forceRotating = bool;  
 }  
  
 */\*\*  
 \* sets the label- the current turn or the winner or tie  
 \* @param s  
 \*/* public void setLa(String s) {  
 this.la.setText(s);  
 }  
  
  
  
 */\*\*  
 \* checks if end of game, if yes than returns the winner or tie, checks if there was an invalid move  
 \* @param s- the string that indicates if there was a win, a tie, an error or the rotating worked well  
 \* @param b- the buttons of the board  
 \* @param b2- the buttons of the rotate  
 \* @return true if there was an error or false otherwise  
 \*/* public boolean endGame(String s, Button[] b, Button[] b2) {  
 System.*out*.println(s);  
 if (s != "") {  
 if (s == "error") {  
 return true;  
 } else {  
 this.winning=true;  
 setLa(s);  
 this.la.setTextFill(Color.*web*("#0076a3"));  
 la.setPrefSize(120, 120);  
 la.setFont(new Font("Arial", 30));  
 for (int i = 0; i < b.length; i++) {  
 b[i].setDisable(true);*//finish game* }  
 for (int i = 0; i < b2.length; i++) {  
 b2[i].setDisable(true);*//finish game* }  
 return false;  
 }  
 }  
 return false;  
 }  
  
 */\*\*  
 \* initializes the board itself  
 \* @param primaryStage- the stage to work with  
 \*/* public void initialize\_the\_board(Stage primaryStage)  
 {  
 GridPane grid = new GridPane();*//the buttons of the board itself* Screen screen = Screen.*getPrimary*();  
 grid.setPadding(new Insets(10));  
 double x = screen.getBounds().getWidth();*//the screen size* double y = screen.getBounds().getHeight();*//the screen size* grid.setHgap(5);  
 grid.setVgap(5);  
  
 *//initialies the board itself* int num = 0;*//the id for each button* for (int row = 0; row < BOARD\_SIZE; row++) {  
 for (int col = 0; col < BOARD\_SIZE; col++) {  
  
 Button cell = new Button();  
 cell.setPrefSize(80, 80);  
 Circle circle = new Circle(20);  
 circle.setFill(Color.*TRANSPARENT*);*//sets the initialization color to TRANSPARENT* cell.setGraphic(circle);

if (row == 0 & col == 0) {  
 num = 0;  
 } else if (col % BOARD\_SIZE == 0) {  
 num = num - MINI\_BOARD\_SIZE - BOARD\_SIZE;  
 }  
 if ((col % BOARD\_SIZE) == MINI\_BOARD\_SIZE) { *//half board* num = num + BOARD\_SIZE;  
 }  
 if (row == MINI\_BOARD\_SIZE & col == 0) {  
 num = MINI\_BOARD\_SIZE \* BOARD\_SIZE;  
 }  
 num++;  
 cell.setId(Integer.*toString*(num));*//sets the id* cell.setPadding(new Insets(1));  
 if(right.contains(Integer.*parseInt*(cell.getId()))|left.contains(Integer.*parseInt*(cell.getId()))|  
 up.contains(Integer.*parseInt*(cell.getId()))|down.contains(Integer.*parseInt*(cell.getId()))){  
 if(right.contains(Integer.*parseInt*(cell.getId())))*//right bold* {  
 cell.setStyle("-fx-background-color: brown; -fx-border-color: black;" +  
 " -fx-border-width: 0 7px 0 0;");  
 }  
 else if(left.contains(Integer.*parseInt*(cell.getId())))*//left bold* {  
 cell.setStyle("-fx-background-color: brown; -fx-border-color: black;" +  
 " -fx-border-width: 0 0 0 7px;");  
 }  
 else if(up.contains(Integer.*parseInt*(cell.getId())))*//up bold* {  
 cell.setStyle("-fx-background-color: brown; -fx-border-color: black;" +  
 " -fx-border-width: 7px 0 0 0;");  
 }  
 else if(down.contains(Integer.*parseInt*(cell.getId())))*//down bold* {  
 cell.setStyle("-fx-background-color: brown; -fx-border-color: black;" +  
 " -fx-border-width: 0 0 7px 0;");  
 }  
  
 }  
 else if(Integer.*parseInt*(cell.getId())==RIGHT\_DOWN\_BOLD)  
 {  
 cell.setStyle("-fx-background-color: brown; -fx-border-color: black;" +  
 " -fx-border-width: 0 7px 7px 0;");  
 }  
 else if(Integer.*parseInt*(cell.getId())==LEFT\_DOWN\_BOLD)  
 {  
 cell.setStyle("-fx-background-color: brown; -fx-border-color: black;" +  
 " -fx-border-width: 0 0 7px 7px;");  
 }  
 else if(Integer.*parseInt*(cell.getId())==RIGHT\_UP\_BOLD)  
 {  
 cell.setStyle("-fx-background-color: brown; -fx-border-color: black;" +  
 " -fx-border-width: 7px 7px 0 0;");  
 } else if (Integer.*parseInt*(cell.getId())==LEFT\_UP\_BOLD) {  
 cell.setStyle("-fx-background-color: brown; -fx-border-color: black;" +  
 " -fx-border-width: 7px 0 0 7px;");  
 }  
 else {  
 cell.setStyle("-fx-background-color: brown;");*//regular buttons* }  
 grid.add(cell, col, row);*//adding to the grid* buttons[num - 1] = cell;*//adding to the array of buttons* }  
 }  
  
 *// Create the border pane and add the grid to the center* grid.setAlignment(Pos.*CENTER*);  
 BorderPane borderPane = new BorderPane();*//the main board* borderPane.setMaxSize(800, 600);  
*// Create a new scene and show the stage* borderPane.setCenter(grid);  
 start\_rotate\_buttons(primaryStage, borderPane); *//sets the rotation buttons* }  
  
 */\*\*  
 \* starts the rotation buttons  
 \* @param primaryStage- the stage to work with  
 \* @param borderPane- the same borderPane as the board itself  
 \*/* public void start\_rotate\_buttons(Stage primaryStage, BorderPane borderPane)  
 {  
 Screen screen = Screen.*getPrimary*();  
 double x = screen.getBounds().getWidth();*//the screen size* double y = screen.getBounds().getHeight();*//the screen size* String[] paths = new String[2];  
  
 File file = new File("src/main/photo1/photoLeft.png");  
 String filePath = file.getAbsolutePath();  
  
 File file2 = new File("src/main/photo2/photoRight.png");  
 String filePath2 = file2.getAbsolutePath();  
  
 paths[1] = filePath; *//rotate left photo* paths[0] = filePath2; *//rotate right photo* VBox vBox = new VBox(10); *// 10 pixels of spacing between the buttons* for (int i = 0; i < 2; i++) { *//top left button for miniboard number 1* Image img = new Image(paths[i]);  
 ImageView view = new ImageView(img);  
 view.setFitHeight(50);  
 view.setPreserveRatio(true);  
 Button button = new Button();  
 button.setTranslateX(200);  
 button.setTranslateY(220);  
 button.setPrefSize(50, 50);  
 button.setGraphic(view);  
 vBox.getChildren().add(button);  
 button.setId(Integer.*toString*(i));  
 buttons\_rotate[i] = button; *//adding to the array of buttons* }  
  
 for (int i = 0; i < 2; i++) { *//buttom left button for miniboard number 3* Image img = new Image(paths[i]);  
 ImageView view = new ImageView(img);  
 view.setFitHeight(50);  
 view.setPreserveRatio(true);  
 Button button = new Button();  
 button.setTranslateX(200);  
 button.setTranslateY(400);  
 button.setPrefSize(50, 50);  
 button.setGraphic(view);  
 vBox.getChildren().add(button);  
 button.setId(Integer.*toString*(i + 2));  
 buttons\_rotate[i + 2] = button;*//adding to the array of buttons* }  
 vBox.getChildren().add(this.la);  
 vBox.setAlignment(Pos.*BOTTOM\_LEFT*);  
 vBox.setAlignment(Pos.*TOP\_LEFT*);  
  
  
  
 Button exit\_button = new Button("EXIT");*//exit button* exit\_button.setPrefSize(120, 50);  
 exit\_button.setTranslateX(0);  
 exit\_button.setStyle("-fx-background-color: greenyellow;");  
 exit\_button.setTranslateY(-350);  
 exit\_button.setOnAction(event -> {  
 controller.setExitButton();*//exits the game* });  
 vBox.getChildren().add(exit\_button);  
  
  
  
 Button backButton = new Button("Back to Open");*///back button* backButton.setPrefSize(120, 50);  
 backButton.setTranslateX(0);  
 backButton.setStyle("-fx-background-color: greenyellow;");  
 backButton.setTranslateY(-325);  
 backButton.setOnAction(event -> {  
 controller.transferToOpeningScreen(primaryStage); *//transfers to the opening screen* });  
 vBox.getChildren().add(backButton);  
 vBox.setAlignment(Pos.*TOP\_LEFT*);  
 borderPane.setLeft(vBox);  
  
  
  
 VBox vBox3 = new VBox(10); *// 10 pixels of spacing between the buttons* for (int i = 0; i < 2; i++) {*//top right buttons- for miniboard number 2* Image img = new Image(paths[i]);  
 ImageView view = new ImageView(img);  
 view.setFitHeight(50);  
 view.setPreserveRatio(true);  
 Button button = new Button();  
 button.setTranslateX(-200);  
 button.setTranslateY(-400);  
 button.setPrefSize(50, 50);  
 button.setGraphic(view);  
 vBox3.getChildren().add(button);  
 button.setId(Integer.*toString*(i + 4));  
 buttons\_rotate[i + 4] = button; *//adding to array buttons* }  
  
 for (int i = 0; i < 2; i++) {*//buttom right buttons- for miniboard number 4* Image img = new Image(paths[i]);  
 ImageView view = new ImageView(img);  
 view.setFitHeight(50);  
 view.setPreserveRatio(true);  
 Button button = new Button();  
 button.setTranslateX(-200);  
 button.setTranslateY(-180);  
 button.setPrefSize(50, 50);  
 button.setGraphic(view);  
 vBox3.getChildren().add(button);  
 button.setId(Integer.*toString*(i + 6));  
 buttons\_rotate[i + 6] = button; *// adding to the array button* }  
 vBox3.setAlignment(Pos.*TOP\_RIGHT*);  
 vBox3.setAlignment(Pos.*BOTTOM\_RIGHT*);  
 borderPane.setRight(vBox3);  
  
  
 *// Create the scene and set it to the stage* Scene scene = new Scene(borderPane, x, y);  
 primaryStage.setScene(scene);  
 primaryStage.show();  
 }  
  
 */\*\*  
 \* manages the AI player  
 \* @param a- the AIPlayer play  
 \*/* public void ai\_play(AiPlayer a)  
 {  
 Button b;  
 if (this.turn == 1) { *//ai turn* a.setPlayers();  
 AiPlayer.*player\_move*++; *//adding the strategy turn of the triple power play* int grade1= a.triple\_power\_play(); *//the grade that the triple power play returns* int grade2= a.defense(); *//the grade that the defence returns* int grade3= a.rows\_columns(); *//the grade that the rows amd columns returns* if(grade1>grade2 & grade1>grade3) *//the triple power play chosen* {  
 mini\_board = AiPlayer.*mini\_board\_for\_twist\_triple*; *//the mini board to twist* index = AiPlayer.*index\_triple*; *//the index to add the trophy* rotating = AiPlayer.*direction\_rotating\_triple*; *//the direction to rotate* }  
 else if(grade2>grade1 & grade2>grade3) *//the defence chosen* {  
 mini\_board = AiPlayer.*mini\_board\_for\_twist\_defense*; *//the mini board to twist* index = AiPlayer.*index\_defense*; *//the index to add the trophy* rotating = AiPlayer.*direction\_rotating\_defense*; *//the direction to rotate* AiPlayer.*player\_move*--; *//the triple power play wasn't chosen* } else if (grade3>grade2 & grade3>grade1) { *//the rows\_columns chosen* mini\_board = AiPlayer.*mini\_board\_for\_twist\_rows\_columns*; *//the mini board to twist* index = AiPlayer.*index\_rows\_columns*; *//the index to add the trophy* rotating = AiPlayer.*direction\_rotating\_rows\_columns*; *//the direction to rotate* AiPlayer.*player\_move*--; *//the triple power play wasn't chosen* }  
 b = buttons[index - 1];  
 setError(endGame((controller.addToolAI(getTurn(), b, this.forceRotating)), buttons, buttons\_rotate));*//adding a tool* if(this.winning==false) {  
 a.setPlayers();  
 if (this.error == false) {*//valid move* if (this.turn == 0)  
 setTurn(1);*//change turn* else  
 setTurn(0);  
 setRotateOnce(true);  
 setForceRotating(false);  
 }  
 Button b2;  
 *//check which button to push for the right twist* if (mini\_board == 1) {  
 if (rotating == 1)  
 b2 = buttons\_rotate[1];  
 else  
 b2 = buttons\_rotate[0];  
 } else if (mini\_board == 2) {  
 if (rotating == 1)  
 b2 = buttons\_rotate[5];  
 else  
 b2 = buttons\_rotate[4];  
 } else if (mini\_board == 3) {  
 if (rotating == 1)  
 b2 = buttons\_rotate[3];  
 else  
 b2 = buttons\_rotate[2];  
 } else {  
 if (rotating == 1)  
 b2 = buttons\_rotate[7];  
 else  
 b2 = buttons\_rotate[6];  
 }  
  
 if (this.error == false) {  
 if (getTurn() == 0) *//still not human* {  
 setRotateOnce(controller.rotateBoardAi( mini\_board,  
 rotating, this.rotateOnce, buttons));*//rotating the miniboard* }  
 setForceRotating(true);  
 }  
 }  
  
 }  
  
  
 }  
  
 */\*\*  
 \* starts the game  
 \* @param primaryStage- the main stage  
 \*/* public void start(Stage primaryStage) {  
 initialize\_the\_board(primaryStage); *//initializes the board* setLa("player 1\nturn");  
 if(OpeningScreen.*num*==0) { *//hVSh* for (int i = 0; i < BOARD\_SIZE \* BOARD\_SIZE; i++) {  
 Button b = buttons[i];  
 b.setOnAction(e -> {  
 setError(endGame((controller.addTool(getTurn(), b, this.forceRotating)), buttons, buttons\_rotate));*//adding a tool* if (this.error == false) {*//valid move* if (this.turn == 0)  
 setTurn(1);*//change turn* else  
 setTurn(0);  
 setRotateOnce(true);  
 setForceRotating(false);  
 }  
 });  
  
 for (int j = 0; j < ROTATE\_BUTTONS; j++) {  
 Button b2 = buttons\_rotate[j];  
 b2.setOnAction(e2 -> {  
 if (this.error == false) {  
 setRotateOnce(controller.rotateBoard(b2, this.rotateOnce, buttons));*//rotating the miniboard* if (getTurn() == 1)  
 setLa("player 2\nturn");*//sets player turn* else  
 setLa("player 1\nturn");  
 setForceRotating(true);  
 }  
 });  
 }  
 }  
 }  
  
  
  
 else if(OpeningScreen.*num*==1){ *//aiVSh* AiPlayer a= new AiPlayer();  
 this.turn=1; *//the first player is ai* ai\_play(a); *//sets the AI* setLa("player1\nturn");  
 Timer timer = new Timer();  
  
  
 for (int i = 0; i < BOARD\_SIZE \* BOARD\_SIZE; i++) {  
  
 Button b;  
 b = buttons[i];  
 b.setOnAction(e -> {  
 setError(endGame((controller.addToolAI(getTurn(), b, this.forceRotating)), buttons, buttons\_rotate));*//adding a tool* if (this.error == false) {*//valid move* if (this.turn == 0)  
 setTurn(1);*//change turn* else  
 setTurn(0);  
 setRotateOnce(true);  
 setForceRotating(false);  
 }  
 });  
  
 for (int j = 0; j < ROTATE\_BUTTONS; j++) {  
 Button b2 = buttons\_rotate[j];  
 b2.setOnAction(e2 -> {  
 setRotateOnce(controller.rotateBoard(b2, this.rotateOnce, buttons));*//rotating the miniboard* setForceRotating(true);  
 setLa("ai\nturn");  
 TimerTask task2 = new TimerTask() {  
 @Override  
 public void run() {  
 Platform.*runLater*(() -> ai\_play(a));  
 }  
 };  
 timer.schedule(task2, 500);  
  
 TimerTask task = new TimerTask() {

@Override

public void run() {  
 Platform.*runLater*(() -> {  
 if (getWinning() == false) {  
 setLa("player1\nturn");  
 }  
 });  
 }  
 };  
 if(this.winning==false){  
 timer.schedule(task, 1000);  
 }  
 });  
 }  
 }  
 }  
 }  
}

## מחלקת MiniBoard:

package com.example.pentagogame.Model;  
public class MiniBoard {  
 private short board\_mini;  
  
 */\*\*  
 \* initialize mini board  
 \*  
 \* @param board\_mini  
 \*/* public MiniBoard(short board\_mini) {  
 this.board\_mini = board\_mini;  
 }  
  
 */\*\*  
 \* rotates the board left (the variable)  
 \*  
 \* @return- returns the board after rotating left  
 \*/* public short rotate\_left() {  
  
 final int START\_1 = 6;  
 final int END\_1 = -3;  
 final int GAP\_123 = 4;  
 final int START\_2 = 4;  
 final int END\_2 = -5;  
 final int START\_3 = -2;  
 final int END\_3 = 7;  
 final short MISUH\_1 = 0b100000000000000;  
 final short MISUH\_3 = 0b000000100000000;  
 final short MISUH\_2 = 0b000100000000000;  
 final int STRAT\_INDEX1 = 8;  
 final int START\_INDEX2 = 7;  
 final int START\_INDEX3 = 6;  
 final int LEN\_ROW = 3;  
  
  
 short x = this.board\_mini;  
 short save = x;  
 short newx = x;  
 short num = MISUH\_1;  
 short index = 0;  
 for (int i = START\_1; i > END\_1; i = i - GAP\_123) {  
 newx = (short) (x & num);  
 if (i == -2) {  
 newx = (short) (newx << 2);  
 } else  
 newx = (short) (newx >> i);  
 if (newx == 0) {  
 save = (short) (save & ~(1 << STRAT\_INDEX1 + index \* LEN\_ROW));  
 } else {  
 save = (short) (save | (newx | save));  
 }  
 num = (short) (num >>> 1);  
 index++;  
 }  
 short level1 = save; *//rotates the first row* x = this.board\_mini;  
 save = x;  
 newx = x;  
 num = MISUH\_2;  
 index = 0;  
 for (int i = START\_2; i > END\_2; i = i - GAP\_123) {  
 newx = (short) (x & num);  
 if (i == -4) {  
 newx = (short) (newx << 4);  
 } else  
 newx = (short) (newx >>> i);  
 if (newx == 0) {  
 save = (short) (save & ~(1 << START\_INDEX2 + index \* LEN\_ROW));  
 } else {  
 save = (short) (save | (newx | save));  
 }  
 num = (short) (num >>> 1);  
 index++;  
  
 }  
  
 short level2 = save;*//rotates the second row* x = this.board\_mini;  
 save = x;  
 newx = x;  
 num = MISUH\_3;  
 index = 0;

for (int i = START\_3; i < END\_3; i = i + GAP\_123) {  
 newx = (short) (x & num);  
 if (i == -2) {  
 newx = (short) (newx >>> 2);  
 } else {  
 newx = (short) (newx << i);  
 }  
 if (newx == 0) {  
 save = (short) (save & ~(1 << START\_INDEX3 + index \* LEN\_ROW));  
 } else {  
 save = (short) (save | (newx | save));  
 }  
 num = (short) (num >>> 1);  
 index++;  
  
 }  
 short level3 = save;*//rotates the third row* short last = 0;  
 short keep = 0;  
 short move = MISUH\_1;  
  
 for (int i = 0; i < LEN\_ROW; i++) {  
 keep = (short) (level1 & move);  
 last = (short) (keep | last);  
 move = (short) (move >>> 1);  
  
 keep = (short) (level2 & move);  
 last = (short) (keep | last);  
 move = (short) (move >>> 1);  
  
 keep = (short) (level3 & move);  
 last = (short) (keep | last);  
 move = (short) (move >>> 1);  
 }  
  
  
 return last; *//matching the 3 rows* }

*/\*\*  
 \* rotates the board right (the variable)  
 \*  
 \* @return- returns the board after rotating right  
 \*/* public short rotate\_right() {  
 final int SHORT\_SIZE = 16;  
 final int LEN\_ROW = 3;  
 final int INDEX\_START\_ROW1 = 2;  
 final int INDEX\_END\_ROW1 = 6;  
 final int DIFFERENCE\_ROWS = 2;  
 final int START\_INDEX2 = 7;  
 final int STRAT\_INDEX3 = 8;  
 final int START\_ROW2 = -2;  
 final short MISUH\_1 = 0b100000000000000;  
 final short MISUH\_3 = 0b000000100000000;  
 final short MISUH\_2 = 0b000100000000000;  
  
  
 short x = board\_mini;  
 short save = x;  
 short newx = x;  
 short save2;  
 short num = MISUH\_1;  
 short index = 2;  
 for (int i = INDEX\_START\_ROW1; i < INDEX\_END\_ROW1 + 1; i = i + DIFFERENCE\_ROWS) {  
 newx = (short) (x & num);  
 newx = (short) (newx >>> i);  
 if (newx == 0) {  
 save = (short) (save & ~(1 << SHORT\_SIZE - i - index));  
 } else {  
 save = (short) (save | (newx | save));  
 }  
 num = (short) (num >>> 1);  
 index++;  
 }  
  
 short level1 = save;*//rotates the first row* x = board\_mini;  
 save = x;  
 newx = x;  
 num = MISUH\_2;  
 index = 2;  
 for (int i = START\_ROW2; i < DIFFERENCE\_ROWS + 1; i = i + DIFFERENCE\_ROWS) {  
 newx = (short) (x & num);  
  
 if (i == -2) {  
 newx = (short) (newx << 2);  
 } else {  
 newx = (short) (newx >>> i);  
 }  
 if (newx == 0) {  
 save = (short) (save & ~(1 << START\_INDEX2 + index \* LEN\_ROW));  
 } else {  
 save = (short) (save | (newx | save));  
 }  
 num = (short) (num >>> 1);  
 index--;  
  
 }  
  
 short level2 = save;*//rotates the second row* x = board\_mini;  
 save = x;  
 newx = x;  
 num = MISUH\_3;  
 index = 2;  
 for (int i = INDEX\_END\_ROW1; i > 1; i = i - DIFFERENCE\_ROWS) {  
 newx = (short) (x & num);  
 newx = (short) (newx << i);  
 if (newx == 0) {  
 save = (short) (save & ~(1 << STRAT\_INDEX3 + index \* LEN\_ROW));  
 } else {  
 save = (short) (save | (newx | save));  
 }  
 num = (short) (num >>> 1);  
 index--;  
 }  
 short level3 = save;*//rotates the third row* short last = 0;  
 short keep = 0;  
 short move = 0b100000000000000;  
  
 for (int i = 0; i < LEN\_ROW; i++) {  
 keep = (short) (level3 & move);  
 last = (short) (keep | last);  
 move = (short) (move >>> 1);  
  
 keep = (short) (level2 & move);  
 last = (short) (keep | last);  
 move = (short) (move >>> 1);  
  
 keep = (short) (level1 & move);  
 last = (short) (keep | last);  
 move = (short) (move >>> 1);  
 }  
 return last; *//matching the 3 rows* }  
  
}

## מחלקת Board:

package com.example.pentagogame.Model;  
  
public class Board {  
 private long player1; *//the board from player1 point of view* private long player2; *//the board from player2 point of view / ai player* private int player\_turn; *//the current turn* private long[] masks = new long[]{0b111111111000000000000000000000000000000000000000000000000000000L,  
 0b000000000111111111000000000000000000000000000000000000000000000L,  
 0b000000000000000000111111111000000000000000000000000000000000000L,  
 0b000000000000000000000000000111111111000000000000000000000000000L}; *//masks for each mini board* private final int LEFT = 1; *//rotating left* private final int RIGHT = 2; *//rotating right  
  
 /\*\*  
 \* initialize the board  
 \*/* public Board() {  
 this.player1 = 0;  
 this.player2 = 0;  
 this.player\_turn = 1;  
 }  
  
 */\*\*  
 \* getter  
 \*  
 \* @return- returns player1 board  
 \*/* public long getPlayer1() {  
 return this.player1;  
 }  
  
 */\*\*  
 \* getter  
 \*  
 \* @return- returns player2 board  
 \*/* public long getPlayer2() {  
 return this.player2;  
 }  
  
  
 */\*\*  
 \* setter  
 \*  
 \* @param player1- setting player1 board  
 \*/* public void setPlayer1(long player1) {  
 this.player1 = player1;  
 }  
  
 */\*\*  
 \* setter  
 \*  
 \* @param player2- setting player1 board  
 \*/* public void setPlayer2(long player2) {  
 this.player2 = player2;  
 }  
  
 */\*\*  
 \* setter  
 \*  
 \* @param pt- setting players turn  
 \*/* public void setPlayer\_turn(int pt) {  
 this.player\_turn = pt;  
 }  
  
  
 */\*\*  
 \* creates an array of masks and casts x from long to short  
 \* @param num- num of mini board  
 \* @param board- the board  
 \* @return - the board in short  
 \*/* public short mask(int num, long board) {  
 num = num - 1;  
 board = board & masks[num];  
 board = board >>> (21 + (3 - num) \* 9); *//moving the number to be in the right* short y = (short) board;  
 return y;  
  
 }  
  
  
 */\*\*  
 \* changes the board after the twist  
 \*  
 \* @param num- the mini board we want to twist  
 \* @param board- the board  
 \* @param last- the mini board after twist  
 \* @return the board after change  
 \*/* public long opmask(int num, long board, short last) {  
 num = num - 1;  
 board &= ~masks[num];  
 long last2 = last & (short) 0b111111111000000;  
 last2 = (long) last;  
 board |= (last2 << (21 + (3 - num) \* 9)) & masks[num];  
 return board;  
 }  
  
  
 */\*\*  
 \* twisting the board  
 \*  
 \* @param num- the board to twist 12  
 \* 34  
 \* @param rotate- 1 to rotate left and 2 to rotate right  
 \*/* public void create\_mini\_board\_twist(int num, int rotate) {  
 short miniboard1 = 0;  
 short miniboard2 = 0;  
 miniboard1 = mask(num, this.player1); *//casting to short* miniboard2 = mask(num, this.player2); *//casting to short* MiniBoard mini1 = new MiniBoard(miniboard1);  
 MiniBoard mini2 = new MiniBoard(miniboard2);  
 if (rotate == LEFT) {  
 miniboard1 = mini1.rotate\_left();  
 miniboard2 = mini2.rotate\_left();  
 } else {  
 miniboard1 = mini1.rotate\_right();  
 miniboard2 = mini2.rotate\_right();  
 }  
 setPlayer1(opmask(num, this.player1, miniboard1)); *//casting back to long with the mini board after change* setPlayer2(opmask(num, this.player2, miniboard2)); *//casting back to long with the mini board after change* }  
  
  
 */\*\*  
 \* changes the board after adding a trophy  
 \*  
 \* @param id\_cell- the place the trophy added to  
 \*/* public boolean change\_board(int id\_cell) {  
 long mask = 0b100000000000000000000000000000000000000000000000000000000000000L;  
 long new\_mask = (mask >>> (id\_cell - 1));  
  
 if ((new\_mask & this.player1) != 0 | (new\_mask & this.player2) != 0) {  
 return false; *//already taken, invalid step* }  
 if (this.player\_turn == 1) {  
 setPlayer1(getPlayer1() | new\_mask); *//adding the trophy to the board* } else {  
 setPlayer2(getPlayer2() | new\_mask); *//adding the trophy to the board* }  
 return true; *//valid move* }  
  
  
 */\*\*  
 \* returns true if current player won the game  
 \*  
 \* @return  
 \*/* public boolean checkForWin() {  
 long board;  
 if (this.player\_turn == 1) {  
 board = this.player1;  
 } else  
 board = this.player2;  
  
 *// Check for horizontal wins - אופקי* long mask1 = 0b111000000110000000000000000000000000000000000000000000000000000L; *//mask for the horizontal win* long mask2 = 0b011000000111000000000000000000000000000000000000000000000000000L; *//mask for the horizontal win* for (int row = 0; row < 6; row++) { *//6 masks* long rowBits1=0;  
 long rowBits2=0;  
 if(row<3){  
 rowBits1 = (mask1 >> (row \* 3));  
 rowBits2 = (mask2 >> (row \* 3));}  
 else {  
 rowBits1 = (mask1 >> (9+row \* 3));  
 rowBits2 = (mask2 >> (9+row \* 3));  
 }  
  
 if (((board & rowBits1) == rowBits1) || ((board & rowBits2) == rowBits2)) {  
 return true;  
 }  
 }  
  
  
 *// Check for vertical wins - אנכי* long colMask1 = 0b100100100000000000100100000000000000000000000000000000000000000L;*//mask for the vertical win* long colMask2 = 0b000100100000000000100100100000000000000000000000000000000000000L;*//mask for the vertical win* long colMask3 = 0b000000000100100100000000000100100000000000000000000000000000000L;*//mask for the vertical win* long colMask4 = 0b000000000000100100000000000100100100000000000000000000000000000L;*//mask for the vertical win* for (int col = 0; col < 3; col++) {  
 long colBits1 = (colMask1 >> (col));  
 long colBits2 = (colMask2 >> (col));  
 long colBits3 = (colMask3 >> (col));  
 long colBits4 = (colMask4 >> (col));  
 if (((board & colBits1) == colBits1) || ((board & colBits2) == colBits2) ||  
 ((board & colBits3) == colBits3) || ((board & colBits4) == colBits4)) {  
 return true;  
 }  
 }  
  
 *// Check for diagonal wins* long diag1Mask = 0b100010001000000000000000000100010000000000000000000000000000000L; *//monica's five winning* long diag2Mask = 0b000010001000000000000000000100010001000000000000000000000000000L; *//monica's five winning* if (((board & diag1Mask) == diag1Mask))  
 return true;  
 if (((board & diag2Mask) == diag2Mask))  
 return true;  
  
 long diag3Mask = 0b000000000001010100001010000000000000000000000000000000000000000L; *//monica's five winning* long diag4Mask = 0b000000000000010100001010100000000000000000000000000000000000000L; *//monica's five winning* if (((board & diag3Mask) == diag3Mask))  
 return true;  
 if (((board & diag4Mask) == diag4Mask))  
 return true;  
  
  
 long theTripleWin = 0b010001000000000100000000000010001000000000000000000000000000000L; *//the triple power play winning* long theTripleWin2 = 0b000100010000000000001000000000100010000000000000000000000000000L; *//the triple power play winning* long theTripleWin3 = 0b000000001010100000010100000000000000000000000000000000000000000L; *//the triple power play winning* long theTripleWin4 = 0b000000000000001010000001010100000000000000000000000000000000000L; *//the triple power play winning* if (((board & theTripleWin) == theTripleWin))  
 return true;  
 if (((board & theTripleWin2) == theTripleWin2))  
 return true;  
 if (((board & theTripleWin3) == theTripleWin3))  
 return true;  
 if (((board & theTripleWin4) == theTripleWin4))  
 return true;  
  
 return false; *//no winning* }  
  
  
 */\*\*  
 \* returns true if the game ends with tie  
 \*  
 \* @return  
 \*/* public boolean checkForTie() {  
 long mask = 0b111111111111111111111111111111111111000000000000000000000000000L;  
 long boards = this.player1 | this.player2;  
 if ((mask & boards) == mask) {  
 return true;  
 }  
 int current\_player= this.player\_turn;  
 this.player\_turn=1;  
 if(checkForWin()==true)  
 {  
 this.player\_turn=0;  
 if(checkForWin()==true) {  
 this.player\_turn=current\_player;  
 return true;  
 }  
 }  
 this.player\_turn=current\_player;  
 return false;  
 }  
  
  
 */\*\*  
 \* rotates the whole board - all the mini boards to right or left  
 \* @param mini\_board1- the first mini board to rotate  
 \* @param mini\_board2- the second mini board to trotate  
 \* @return the players after the change  
 \*/* public long [] rotate\_whole(int mini\_board1, int mini\_board2)  
 {  
 long [] players= new long[2]; *//keeps the players* if(mini\_board1==2 & mini\_board2==4) *//mini board 2 to be mini board 1, and mini board 4 to be mini board 2* {  
 create\_mini\_board\_twist(2, 1); *//rotate left* create\_mini\_board\_twist(4, 1); *//rotate left  
  
 //player2* long masking\_36= 0b111111111111111111111111111111111111000000000000000000000000000L; *//mask of the entire board - 36 bits from the left* long mini\_board\_second= getPlayer2()<<9; *//deletes the first mini board* mini\_board\_second=mini\_board\_second>>>27; *//move it to be mini board 4* mini\_board\_second=mini\_board\_second&masking\_36; *//deletes all the mini boards besides mini board 2* mini\_board\_second=mini\_board\_second<<27; *//moves it to be in mini board 1* long mini\_board\_fourth = getPlayer2()<<27; *//deletes all mini board besides mini board 4* mini\_board\_fourth=mini\_board\_fourth>>>9; *//moves it to mini board 2* setPlayer2(mini\_board\_second | mini\_board\_fourth); *//connects the 2 mini board with the 1  
  
  
 //player1* mini\_board\_second= getPlayer1()<<9;*//deletes the first mini board* mini\_board\_second=mini\_board\_second>>>27;*//move it to be mini board 4* mini\_board\_second=mini\_board\_second&masking\_36; *//deletes all the mini boards besides mini board 2* mini\_board\_second=mini\_board\_second<<27;*//moves it to be in mini board 1* mini\_board\_fourth = getPlayer1()<<27;*//deletes all mini board besides mini board 4* mini\_board\_fourth=mini\_board\_fourth>>>9;*//moves it to mini board* setPlayer1(mini\_board\_second | mini\_board\_fourth); *//connects the 2 mini board with the 1* }  
 else if(mini\_board1==1 & mini\_board2==3)*//mini board 3 to be mini board 1, and mini board 1 to be mini board 2* {  
 create\_mini\_board\_twist(1, 2); *//rotate right* create\_mini\_board\_twist(3, 2); *//rotate right  
  
  
 //player2* long masking\_36= 0b111111111111111111111111111111111111000000000000000000000000000L;*//mask of the entire board - 36 bits from the left* long mini\_board\_first= getPlayer2()>>>27; *//moving mini board 1 to be 4* mini\_board\_first=mini\_board\_first&masking\_36; *//deletes all mini board besides mini board 1* mini\_board\_first=mini\_board\_first<<18; *//moves it to mini board 2* long mini\_board\_third = getPlayer2()<<18; *//moving mini board 3 to be 1* mini\_board\_third=mini\_board\_third>>>27; *//moving mini board 3 to be 4* mini\_board\_third=mini\_board\_third&masking\_36;*//deletes all mini board besides mini board 3* mini\_board\_third=mini\_board\_third<<27; *//moves it to mini board 1* setPlayer2(mini\_board\_first | mini\_board\_third);*//connects the 2 mini board with the 1  
  
  
 //player1* mini\_board\_first= getPlayer1()>>>27;*//moving mini board 1 to be 4* mini\_board\_first=mini\_board\_first&masking\_36;*//deletes all mini board besides mini board 1* mini\_board\_first=mini\_board\_first<<18;*//moves it to mini board 2* mini\_board\_third = getPlayer1()<<18;*//moving mini board 3 to be 1* mini\_board\_third=mini\_board\_third>>>27; *//moving mini board 3 to be 4* mini\_board\_third=mini\_board\_third&masking\_36;*//deletes all mini board besides mini board 3* mini\_board\_third=mini\_board\_third<<27;*//moves it to mini board 1* setPlayer1(mini\_board\_first | mini\_board\_third);*//connects the 2 mini board with the 1* }  
 else if(mini\_board1==3 & mini\_board2==4)*//mini board 3 to be mini board 1, and mini board 4 to be mini board 2* {  
  
 setPlayer1(getPlayer1()<<18); *//moving mini boards to be 1 and 2* setPlayer2(getPlayer2()<<18); *//moving mini boards to be 1 and 2* mirroring(); *//mirroring the mini board* }  
 players[0]=getPlayer1();  
 players[1]=getPlayer2();  
 return players;  
 }  
  
  
 */\*\*  
 \* returns the index matching to the right mini board by doing the rotate back  
 \* @param index - the index to put the trophy according to mini board 1, 2  
 \* @param mini\_board1- the first mini board to rotate  
 \* @param mini\_board2- the second mini board to rotate  
 \* @return  
 \*/* public int rotate\_whole\_opp(int index, int mini\_board1, int mini\_board2)  
 {  
 long mask = 0b100000000000000000000000000000000000000000000000000000000000000L;  
 Board b;  
 if(mini\_board1==2 & mini\_board2==4)  
 {  
 long id\_mask = (mask >>> (index - 1)); *//the place to put the trophy* id\_mask = id\_mask >>>9; *//moving mini board 1 to be 2 - the original* long new\_mask = (long) (id\_mask << 18); *//mini board 2 to be in 1* new\_mask = (long) (new\_mask >>>27);*//moves it to mini board 4* if(new\_mask ==0){*//mini board 4 is empty* b= new Board();  
 b.setPlayer1(0);  
 b.setPlayer2(id\_mask);  
 }  
 else {b= new Board();  
 b.setPlayer1(0);  
 b.setPlayer2(new\_mask);  
 }  
  
 b.create\_mini\_board\_twist(2, 2); *//rotate back to the right* b.create\_mini\_board\_twist(4, 2); *//rotate back to the right* int count=1;  
 while((mask & b.getPlayer2()) ==0) *//count the new index according to mini board 2, 4* {  
 count++;  
 mask=mask>>>1;  
 }  
 return count;  
 }  
  
  
  
  
  
 else if(mini\_board1==1 & mini\_board2==3)  
 {  
 mask = 0b100000000000000000000000000000000000000000000000000000000000000L;  
 long id\_mask = (mask >>> (index - 1));*//the place to put the trophy* long new\_mask = (long) id\_mask >>>27;*//mini board 1 to 4* new\_mask = new\_mask <<9; *//mini board 1 to be 3* id\_mask = id\_mask <<9; *//mini board 2 to be 1* long mask\_total= id\_mask | new\_mask;*//matching them* if(mask\_total==0){*//* b= new Board();  
 b.setPlayer1(0);  
 b.setPlayer2(id\_mask);  
 }  
 else {  
 b= new Board();  
 b.setPlayer1(0);  
 b.setPlayer2(mask\_total);}  
 b.create\_mini\_board\_twist(1, 1);*//rotate back to the left* b.create\_mini\_board\_twist(3, 1);*//rotate back to the left* int count=1;  
 while((mask & b.getPlayer2()) ==0)*//count the new index according to mini board 1, 3* {  
 count++;  
 mask=mask>>>1;  
 }  
 return count;  
 }  
  
  
 else if(mini\_board1==3 & mini\_board2==4)  
 {  
 mask = 0b100000000000000000000000000000000000000000000000000000000000000L;  
 long id\_mask = (mask >>> (index - 1));*//the place to put the trophy* long save\_player2=getPlayer2();  
 long save\_player1=getPlayer1();  
 setPlayer2(id\_mask);  
 mirroring(); *//mirroring back* long get= getPlayer2()>>>18; *//move it back* setPlayer2(save\_player2);  
 setPlayer1(save\_player1);  
 int count=1;  
 while((mask & get) ==0)*//count the new index according to mini board 3, 4* {  
 count++;  
 mask=mask>>>1;  
 }  
 return count;  
  
 }  
 return 1;  
 }  
  
  
 */\*\*  
 \* mirroring mini board 1,2 back to 3, 4 / mini board 3, 4 to 1, 2  
 \*/* public void mirroring ()  
 {  
 long keep\_it=getPlayer2();  
 long [] players= new long[2];  
 long mask= 0b100000000000000000000000000000000000000000000000000000000000000L;  
  
 for(int i=0; i<2; i++) *//both mini boards* {  
 for(int j=0; j<9; j++) *//checking all the bits of each mini board* {  
 if(j<3)  
 {  
 if((mask & keep\_it)!=0) *//place is taken* {  
 if((getPlayer2() & mask >>>(6))==0) { *//place of the mirrored not taken* setPlayer2(getPlayer2() | mask >>> (6)); *//moving it to mirrored* setPlayer2(getPlayer2() ^ mask);  
 }  
  
 }  
 } else if (j>5)  
 {  
  
 if((mask & keep\_it)!=0) { *//place is taken* if((getPlayer2() & mask <<(6))==0) {*//place of the mirrored not taken* setPlayer2(getPlayer2() | mask << (6)); *//moving it to mirrored* setPlayer2(getPlayer2() ^ mask);  
 }  
 }  
  
 }  
 mask=mask>>>1;  
 }  
 }  
  
  
 mask= 0b100000000000000000000000000000000000000000000000000000000000000L;  
 keep\_it=getPlayer1();  
  
 for(int i=0; i<2; i++)*//both mini boards* {  
 for(int j=0; j<9; j++)*//checking all the bits of each mini board* {  
 if(j<3)  
 {  
 if((mask & keep\_it)!=0)*//place is taken* {  
 setPlayer1(getPlayer1() | mask >>>(6));*//place of the mirrored not taken* setPlayer1(getPlayer1() ^ mask); *//moving it to mirrored* }  
 } else if (j>5)  
 {  
 if((mask & keep\_it)!=0) {*//place is taken* setPlayer1(getPlayer1() | mask << (6));*//place of the mirrored not taken* setPlayer1(getPlayer1() ^ mask); *//moving it to mirrored* }  
 }  
 mask=mask>>>1;  
 }  
 }  
 }  
  
}

## מחלקת ControllerClass:

package com.example.pentagogame.Controller;  
import com.example.pentagogame.DemoDriver;  
import com.example.pentagogame.Model.Board;  
import com.example.pentagogame.View.Instructions;  
import com.example.pentagogame.View.OpeningScreen;  
import com.example.pentagogame.View.TheBoard;  
import javafx.application.Platform;  
import javafx.scene.control.Button;  
import javafx.scene.paint.Color;  
import javafx.scene.shape.Circle;  
import javafx.stage.Stage;  
import java.util.Arrays;  
import java.util.Comparator;  
import static javafx.application.Application.*launch*;  
  
public class ControllerClass {  
 public static Board *b*; *//the board of the model* private static OpeningScreen *pt*; *//the opening screen of the view* private final int NUM\_BUTTONS\_MINI\_BOARD=9; *//num of buttons in each mini board  
  
 /\*\*  
 \* initializes the class  
 \*/* public void init()  
 {  
 this.*b*= new Board();  
 this.*pt*=new OpeningScreen();  
 }  
  
  
 */\*\*  
 \* starts the game  
 \* @param stage- the main stage  
 \*/* public void start\_Game(Stage stage) {  
 this.*pt*.start(stage);  
 }  
  
 */\*\*  
 \* exits the game  
 \*/* public void setExitButton()  
 {  
 Platform.*exit*();  
 }  
  
 */\*\*  
 \* transfering to board screen  
 \* @param stage- the main stage  
 \*/* public void transferToBoard(Stage stage)  
 {  
 DemoDriver d= new DemoDriver();  
 Stage stage1=new Stage();  
 d.start(stage1);  
 TheBoard h= new TheBoard();  
 OpeningScreen.*num*=0; *//hVSh* h.start(stage);  
 }  
  
  
 */\*\*  
 \* transfering to AI board screen  
 \* @param stage- the main stage  
 \*/* public void transferToAI(Stage stage)  
 {  
 DemoDriver d= new DemoDriver();  
 Stage stage1=new Stage();  
 d.start(stage1);  
 TheBoard h= new TheBoard();  
 OpeningScreen.*num*=1; *//aiVSh* h.start(stage);  
 }  
  
 */\*\*  
 \* transfer to opening screen  
 \* @param stage- the main stage  
 \*/* public void transferToOpeningScreen(Stage stage)  
 {  
 OpeningScreen o= new OpeningScreen();  
 o.start(stage);  
 }  
  
 */\*\*  
 \* transfer to instructions screen  
 \* @param stage- the main stage  
 \*/* public void transferToistructions(Stage stage)  
 {  
 Instructions h= new Instructions();  
 h.start(stage);  
 }  
  
 */\*\*  
 \* adds a tool to the board(the model), and checks for win or tie.  
 \* changes the current turn and checks if the pushed button is legal  
 \* @param turn- gets the current turn  
 \* @param b- the pushed button  
 \* @param check- only if the older player twisted the board  
 \* @return- string that indicates if there has benn an error or tie or which player won  
 \*/* public String addTool(int turn, Button b, boolean check) {  
 if(check==true){  
  
 if(this.*b*.checkForTie()==true)  
 return "Tie!!";  
 else if (this.*b*.checkForWin()==true) {  
 if(turn==0){  
 return "player2 \nwon!!";}  
 else{  
 return "player1 \nwon!!";}  
 }  
  
 Circle c = (Circle) b.getGraphic();  
 if (c.getFill() == Color.*TRANSPARENT*) {  
 if (turn == 0) {  
 c.setFill(Color.*WHITE*);  
 b.setGraphic(c);  
 turn=1;  
 this.*b*.setPlayer\_turn(1);*//switches turn* this.*b*.change\_board(Integer.*parseInt*(b.getId()));*//adding the tool* }  
 else {  
 c.setFill(Color.*BLACK*);  
 b.setGraphic(c);  
 turn=0;  
 this.*b*.setPlayer\_turn(0);*//switches turn* this.*b*.change\_board(Integer.*parseInt*(b.getId()));*//adding the tool* }  
 }  
 else  
 return "error"; *//trying to add a button in a taken place- invalid move* }  
 return ""; *//continue* }  
  
 */\*\*  
 \* rotates the board  
 \* @param b- the rotate button that pushed  
 \* @param bool- true if didn't twist yet  
 \* @param buttons- all buttons  
 \* @return- returns false in order to allow twisting only once  
 \*/* public boolean rotateBoard(Button b, boolean bool, Button [] buttons)  
 {  
 short player1;  
 short player2;  
 short mask= 0b100000000000000;  
 if(bool==true) {  
 *//Integer.parseInt(b.getId())%2+1* int numBoard = 0;*//num of the board to rotate* int boardRotate = 0;*//rotating right or left* if (Integer.*parseInt*(b.getId()) < 2)  
 numBoard = 1;  
 else if (Integer.*parseInt*(b.getId()) > 5)  
 numBoard = 4;  
 else if (Integer.*parseInt*(b.getId()) > 1 && Integer.*parseInt*(b.getId()) < 4)  
 numBoard = 3;  
 else  
 numBoard = 2;  
 if (Integer.*parseInt*(b.getId()) % 2 == 0)  
 boardRotate = 2;  
 else  
 boardRotate = 1;  
 this.*b*.create\_mini\_board\_twist(numBoard, boardRotate);*//twistingin model* player1= this.*b*.mask(numBoard, this.*b*.getPlayer1());  
 player2= this.*b*.mask(numBoard, this.*b*.getPlayer2());  
  
 Arrays.*sort*(buttons, Comparator.*comparingInt*(button -> Integer.*parseInt*(button.getId()))); *//sorts the array of buttons by id* for(int i=1+NUM\_BUTTONS\_MINI\_BOARD\*(numBoard-1); i<NUM\_BUTTONS\_MINI\_BOARD+1+NUM\_BUTTONS\_MINI\_BOARD\*(numBoard-1); i++)  
 {  
 Circle c= new Circle(20);  
 if((player1&mask)!=0)  
 {  
 c.setFill(Color.*WHITE*); *//change to white* buttons[i-1].setGraphic(c);  
 }  
 else if((player2&mask)!=0)  
 {  
 c.setFill(Color.*BLACK*); *//change to black* buttons[i-1].setGraphic(c);  
 }  
 else if((player1&mask)==0 && ((player2&mask)==0))  
 {  
 c.setFill(Color.*TRANSPARENT*); *//change to empty* buttons[i-1].setGraphic(c);  
 }  
  
 mask= (short) (mask>>1);  
 }  
 }  
 return false;  
  
 }  
  
  
 */\*\*  
 \* adds a tool to the board(the model), and checks for win or tie. add tool in the AI mode  
 \* changes the current turn and checks if the pushed button is legal  
 \* @param turn- gets the current turn  
 \* @param b- the pushed button  
 \* @param check- only if the older player twisted the board  
 \* @return- string that indicates if there has benn an error or tie or which player won  
 \*/* public String addToolAI(int turn, Button b, boolean check) {  
 if(check==true){  
 if(this.*b*.checkForTie()==true)  
 return "Tie!!";  
 else if (this.*b*.checkForWin()==true) {  
 if(turn==0){  
 return "AI \nwon!!";} *//ai is 1* else{  
 return "player1 \nwon!!";} *//player1 is 0* }  
  
 Circle c = (Circle) b.getGraphic();  
 if (c.getFill() == Color.*TRANSPARENT*) {  
 if (turn == 0) { *//player1* c.setFill(Color.*WHITE*);  
 b.setGraphic(c);  
 turn=1;  
 this.*b*.setPlayer\_turn(1);*//switches turn* this.*b*.change\_board(Integer.*parseInt*(b.getId()));*//adding the tool* }  
 else { *//ai player* c.setFill(Color.*BLACK*);  
 b.setGraphic(c);  
 turn=0;  
 this.*b*.setPlayer\_turn(0);*//switches turn* this.*b*.change\_board(Integer.*parseInt*(b.getId()));*//adding the tool* }  
 }  
 else  
 return "error"; *//trying to add a button in a taken place- invalid move* }  
 return ""; *//continue* }  
  
 */\*\*  
 \* rotates the board  
 \* @param numBoard- num board to twist  
 \* @param boardRotate- direction of rotating  
 \* @param bool- true if didn't twist yet  
 \* @param buttons - all buttons  
 \* @return  
 \*/* public boolean rotateBoardAi(int numBoard, int boardRotate, boolean bool, Button [] buttons)  
 {  
 short player1;  
 short player2;  
 short mask= 0b100000000000000;  
 if(bool==true) {  
 this.*b*.create\_mini\_board\_twist(numBoard, boardRotate);*//twisting in model* player1= this.*b*.mask(numBoard, this.*b*.getPlayer1());  
 player2= this.*b*.mask(numBoard, this.*b*.getPlayer2());  
  
 Arrays.*sort*(buttons, Comparator.*comparingInt*(button -> Integer.*parseInt*(button.getId()))); *//sorts the array of buttons by id* for(int i=1+NUM\_BUTTONS\_MINI\_BOARD\*(numBoard-1); i<NUM\_BUTTONS\_MINI\_BOARD+1+NUM\_BUTTONS\_MINI\_BOARD\*(numBoard-1); i++)  
 {  
 Circle c= new Circle(20);  
 if((player1&mask)!=0)  
 {  
 c.setFill(Color.*WHITE*); *//change to white* buttons[i-1].setGraphic(c);  
 }  
 else if((player2&mask)!=0)  
 {  
 c.setFill(Color.*BLACK*); *//change to black* buttons[i-1].setGraphic(c);  
 }  
 else if((player1&mask)==0 && ((player2&mask)==0))  
 {  
 c.setFill(Color.*TRANSPARENT*); *//change to empty* buttons[i-1].setGraphic(c);  
 }  
  
 mask= (short) (mask>>1);  
 }  
 }  
 return false;  
  
 }  
  
  
 public static void main(String[] args) {  
 *launch*(args);  
 }  
}

## מחלקת AiPlayer:

package com.example.pentagogame;  
import com.example.pentagogame.Controller.ControllerClass;  
import com.example.pentagogame.Model.Board;  
import java.util.Random;  
  
public class AiPlayer {  
 public static void main(String[] args) {  
 }  
  
 private final int START\_MINI\_2 = 10;  
 private final int START\_MINI\_1 = 0;  
 private final int END\_MINI\_1 = 9;  
 private final int START\_MINI\_3 = 19;  
 private final int END\_MINI\_2 = 18;  
 private final int START\_MINI\_4 = 28;  
 private final int MINI\_BOARD\_SIZE = 9;  
 private final int ROW\_SIZE = 6;  
 private Long player1;  
 private Long ai;  
  
 Random rand = new Random();  
  
  
 private long[] masks\_boards\_12 = {0b011000000111000000000000000000000000000000000000000000000000000L,  
 0b011000000001001001000000000000000000000000000000000000000000000L,  
 0b011000000100100100000000000000000000000000000000000000000000000L,  
 0b100100000111000000000000000000000000000000000000000000000000000L,  
 0b000001001111000000000000000000000000000000000000000000000000000L,  
  
 0b111000000110000000000000000000000000000000000000000000000000000L,  
 0b111000000001001000000000000000000000000000000000000000000000000L,  
 0b111000000000100100000000000000000000000000000000000000000000000L,  
 0b100100100110000000000000000000000000000000000000000000000000000L,  
 0b001001001110000000000000000000000000000000000000000000000000000L};  
  
 private long[] masks\_boards\_12\_2 = {0b000111000000110000000000000000000000000000000000000000000000000L,  
 0b000111000010010000000000000000000000000000000000000000000000000L,  
 0b000111000000010010000000000000000000000000000000000000000000000L,  
 0b010010010000110000000000000000000000000000000000000000000000000L,  
 0b010010010000110000000000000000000000000000000000000000000000000L,  
  
 0b000011000000111000000000000000000000000000000000000000000000000L,  
 0b000011000010010010000000000000000000000000000000000000000000000L,  
 0b000011000010010010000000000000000000000000000000000000000000000L,  
 0b010010000000111000000000000000000000000000000000000000000000000L,  
 0b000010010000111000000000000000000000000000000000000000000000000L};  
  
  
 private long[] masks\_boards\_12\_3 = {0b000000111000000110000000000000000000000000000000000000000000000L,  
 0b000000111100100000000000000000000000000000000000000000000000000L,  
 0b000000111000001001000000000000000000000000000000000000000000000L,  
 0b001001001000000110000000000000000000000000000000000000000000000L,  
 0b100100100000000110000000000000000000000000000000000000000000000L,  
  
 0b000000011000000111000000000000000000000000000000000000000000000L,  
 0b000000011100100100000000000000000000000000000000000000000000000L,  
 0b000000011001001001000000000000000000000000000000000000000000000L,  
 0b001001000000000111000000000000000000000000000000000000000000000L,  
 0b000100100000000111000000000000000000000000000000000000000000000L};  
  
 private int[] mini\_board\_12 = {rand.nextInt(2) + 3, 2, 2, 1, 1};  
 private int[] mini\_board\_24 = {rand.nextInt(2) \* 2 + 1, 4, 4, 2, 2};  
 private int[] mini\_board\_34 = {rand.nextInt(2) + 1, 4, 4, 3, 3};  
 private int[] mini\_board\_13 = {rand.nextInt(2) \* 2 + 2, 1, 1, 3, 3};  
  
 private int[][] mini\_board = {mini\_board\_12, mini\_board\_24, mini\_board\_34, mini\_board\_13};  
  
 private int[][] mini\_boards\_to\_twist\_whole = {{2, 4}, {3, 4}, {1, 3}};  
  
  
 private int[][] indexes = {{2, 11, 3, 10, 12}, {2, 15, 3, 12, 18}, {2, 13, 3, 10, 16}, {4, 11, 1, 10, 12}, {6, 11, 9, 10, 12},  
 {2, 11, 3, 10, 1}, {2, 15, 3, 12, 1}, {2, 13, 3, 16, 1}, {4, 11, 1, 10, 7}, {6, 11, 9, 10, 3}};  
  
 private int[][] indexes\_2 = {{5, 14, 4, 6, 13}, {5, 14, 4, 6, 11}, {5, 14, 4, 6, 17}, {5, 14, 2, 8, 17}, {5, 14, 2, 8, 17},  
 {5, 14, 6, 13, 15}, {5, 14, 6, 11, 17}, {5, 14, 6, 11, 17}, {5, 14, 2, 13, 15}, {5, 14, 7, 13, 15}};  
  
 private int[][] indexes\_3 = {{8, 17, 7, 9, 16}, {8, 13, 7, 9, 10}, {8, 15, 7, 9, 18}, {6, 17, 3, 9, 16}, {4, 17, 1, 7, 16},  
 {8, 17, 9, 16, 18}, {8, 13, 9, 10, 16}, {8, 15, 9, 12, 18}, {6, 17, 3, 16, 18}, {4, 17, 7, 16, 18}};  
 private int[] twist\_mini\_boards\_12 = {rand.nextInt(2) + 1, 1, 2, 2, 1};  
 private int[] twist\_mini\_boards\_34 = {rand.nextInt(2) + 1, 2, 1, 1, 2};  
  
 private boolean check\_4\_finish = false;  
  
 private boolean check\_3\_finish = false;  
  
 private boolean check\_2\_finish = false;  
  
 private boolean check\_1\_finish = false;  
  
 private boolean check\_0\_finish = false;  
  
  
 Random random = new Random(); *//randomize* private int[] possible\_moves\_12 = {2, 4, 6, 8, 29, 31, 33, 35}; *//possible indexes to win the first strategy* private int[] mini\_board\_12\_possible1 = {random.nextInt(2) + 2, random.nextInt(2) + 2, 1, 1, 4, 4,  
 random.nextInt(2) + 2, random.nextInt(2) + 2}; *//the mini boards to twist for option 1(the better one)* private int[] twist\_board\_12\_possible1 = {random.nextInt(2) + 1, random.nextInt(2) + 1, 1, 2, 2, 1,  
 random.nextInt(2) + 1, random.nextInt(2) + 1}; *//the twists of the mini boards for option 1* private int[] indexes\_move2\_12\_possible1 = {4, 2, 2, 4, 33, 35, 35, 33}; *//the indexes for stage 2- the next ball* private int[] indexes\_move2\_12\_possible2 = {6, 8, 8, 6, 31, 29, 29, 31}; *//the indexes for stage 2- the next ball(the less good)* private int[] mini\_board\_12\_possible2 = {1, 1, 1, 1, 4, 4, 4, 4}; *//the mini boards to twist for option 2* private int[] twist\_board\_12\_possible2 = {1, 2, 2, 2, 1, 1, 2, 1}; *//the twists of the mini boards for option 2* private int[] possible\_moves\_34 = {11, 13, 15, 17, 20, 22, 24, 26}; *//possible indexes to win the second strategy* private int[] mini\_board\_34\_possible1 = {random.nextInt(2) \* 3 + 1, 2, random.nextInt(2) \* 3 + 1,  
 2, 3, random.nextInt(2) \* 3 + 1, 3, random.nextInt(2) \* 3 + 1}; *//the mini boards to twist for option 1(the better one)* private int[] twist\_board\_34\_possible1 = {random.nextInt(2) + 1, 2, random.nextInt(2) + 1,  
 1, 1, random.nextInt(2) + 1, 2, random.nextInt(2) + 1}; *//the twists of the mini boards for option 1* private int[] indexes\_move2\_34\_possible1 = {15, 11, 11, 15, 22, 26, 26, 22}; *//the indexes for stage 2- the next ball* private int[] indexes\_move2\_34\_possible2 = {13, 17, 17, 13, 24, 20, 20, 24}; *//the indexes for stage 2- the next ball(the less good)* private int[] mini\_board\_34\_possible2 = {2, 2, 2, 2, 3, 3, 3, 3}; *//the mini boards to twist for option 2* private int[] twist\_board\_34\_possible2 = {2, 1, 1, 1, 2, 1, 2, 2}; *//the twists of the mini boards for option 2* public static int *player\_move* = 0; *//number of stage to put the ball - the first ball to put is on move 1;* private int last\_turn\_mini\_board; *//last mini that took* private int last\_turn\_playing\_strategy; *//last strategy that took* public static int *mini\_board\_for\_twist\_defense*; *//mini board for twisting* public static int *index\_defense*; *//index of putting the ball* public static int *direction\_rotating\_defense*; *//the side to rotate* public static int *mini\_board\_for\_twist\_rows\_columns*; *//mini board for twisting* public static int *index\_rows\_columns*; *//index of putting the ball* public static int *direction\_rotating\_rows\_columns*; *//the side to rotate* public static int *mini\_board\_for\_twist\_triple*; *//mini board for twisting* public static int *index\_triple*; *//index of putting the ball* public static int *direction\_rotating\_triple*; *//the side to rotate* private int grade = 80;  
 private int grade\_rows\_columns = 0;  
  
 private int current\_strategy; *//the current strategy in the triple power play  
  
 /\*\*  
 \* set the initialite players  
 \*/* public AiPlayer() {  
 setBoard\_player1(ControllerClass.*b*.getPlayer1());  
 setBoard\_ai(ControllerClass.*b*.getPlayer2());  
 *player\_move*=0;  
 }  
  
 */\*\*  
 \* sets the players  
 \*/* public void setPlayers() {  
 setBoard\_player1(ControllerClass.*b*.getPlayer1());  
 setBoard\_ai(ControllerClass.*b*.getPlayer2());  
 }  
  
 */\*\*  
 \* set ai board  
 \*  
 \* @param board\_ai- the ai board to set  
 \*/* public void setBoard\_ai(Long board\_ai) {  
 this.ai = board\_ai;  
 }  
  
 */\*\*  
 \* set player 1 board  
 \*  
 \* @param board\_player1- the player1 board to set  
 \*/* public void setBoard\_player1(Long board\_player1) {  
 this.player1 = board\_player1;  
 }  
  
 */\*\*  
 \* counts how many 1 are in the string  
 \*  
 \* @param s- the board  
 \* @return  
 \*/* public int count\_1(String s) {  
 int count = 0;  
 for (int i = 0; i < s.length(); i++) {  
 if (s.charAt(i) == '1')  
 count++;  
 }  
 return count;  
 }  
  
  
 */\*\*  
 \* calculates the mini board to twist according to the index(the mini board to twist will be the opposite)  
 \*  
 \* @param index- index of the button to put  
 \* @return- the mini board  
 \*/* public int calc\_mini\_board\_diagonal\_triple(int index, int strategy\_mask) {  
 Random rand = new Random();  
  
 if (strategy\_mask == 0) *//mini boards 1, 2, 4* {  
 if (index < START\_MINI\_2 & index > START\_MINI\_1) {  
 return rand.nextInt(2) \* 2 + 2;*// 2 or 4* } else if (index > END\_MINI\_1 & index < START\_MINI\_3) {  
 return rand.nextInt(2) \* 3 + 1;*// 1 or 4* } else  
 return rand.nextInt(2) + 1;*// 1 or 2* } else if (strategy\_mask == 1) *//mini boards 1, 3, 4* {  
 if (index < START\_MINI\_2 & index > START\_MINI\_1) {  
 return rand.nextInt(2) + 3;*// 3 or 4* } else if (index > END\_MINI\_2 & index < START\_MINI\_4) {  
 return rand.nextInt(2) \* 3 + 1;*// 1 or 4* } else  
 return rand.nextInt(2) \* 2 + 1;*// 1 or 3* } else if (strategy\_mask == 2) *//mini boards 1, 2, 3* {  
 if (index < START\_MINI\_2 & index > START\_MINI\_1) {  
 return rand.nextInt(2) + 2;*// 2 or 3* } else if (index > END\_MINI\_2 & index < START\_MINI\_4) {  
 return rand.nextInt(2) + 1;*// 1 or 2* } else  
 return rand.nextInt(2) \* 2 + 1;*// 1 or 3* } else *//mini boards 2, 3, 4* {  
 if (index > END\_MINI\_1 & index < START\_MINI\_3) {  
 return rand.nextInt(2) + 3;*// 3 or 4* } else if (index > END\_MINI\_2 & index < START\_MINI\_4) {  
 return rand.nextInt(2) \* 2 + 2;*// 2 or 4* } else  
 return rand.nextInt(2) + 2;*// 2 or 3* }  
 }  
  
  
 */\*\*  
 \* defensing the game from vertical, horizontal or diagonal win.  
 \* starts from the 3 ball.  
 \* return- the grade for the strategy  
 \*/* public int defense() {  
 long board = player1; *//the board to check* int grade = 0; *//the grade for thr defence* boolean need\_defence = false; *//true if needs defence  
  
  
 // Check for horizontal wins  
 index\_defense* = 0;  
 *mini\_board\_for\_twist\_defense* = 0;  
 check\_4\_finish = false; *//check if there is 4 in a row* check\_3\_finish = false; *//check if there is 3 in a row* int[][][] indexes\_total = {indexes, indexes\_2, indexes\_3};  
 long[][] masks\_total = {masks\_boards\_12, masks\_boards\_12\_2, masks\_boards\_12\_3};  
  
 for (int z = 0; z < 3; z++) {  
  
 for (int x = 0; x < 4; x++) {  
 long keep\_ai = ai;  
 long keep\_player1 = player1;  
 Board b = new Board();  
 b.setPlayer1(player1);  
 b.setPlayer2(ai);  
 if (x > 0) {  
 long[] players = b.rotate\_whole(mini\_boards\_to\_twist\_whole[x - 1][0], mini\_boards\_to\_twist\_whole[x - 1][1]);  
 keep\_ai = players[1];  
 keep\_player1 = players[0];  
 }  
 for (int i = 0; i < 10; i++) {  
 if (count\_1(Long.*toBinaryString*(keep\_player1 & masks\_total[z][i])) == 4 & count\_1(Long.*toBinaryString*(keep\_ai & masks\_total[z][i])) == 0) { *//not found* long mask = 0b100000000000000000000000000000000000000000000000000000000000000L;  
 for (int j = 0; j < 5; j++) {  
 if (check\_4\_finish == false & ((mask >>> (indexes\_total[z][i][j] - 1)) & keep\_player1) == 0) {  
 *index\_defense* = indexes\_total[z][i][j];  
 if (x > 0) {  
 *index\_defense* = b.rotate\_whole\_opp(*index\_defense*, mini\_boards\_to\_twist\_whole[x - 1][0], mini\_boards\_to\_twist\_whole[x - 1][1]);*//the right index* }  
 *mini\_board\_for\_twist\_defense* = this.mini\_board[x][(i % 5)]; *//the mini board to twist* if (x == 2) {  
 *direction\_rotating\_defense* = this.twist\_mini\_boards\_34[(i % 5)]; *//the direction to rotate* } else  
 *direction\_rotating\_defense* = this.twist\_mini\_boards\_12[(i % 5)]; *//the direction to rotate* check\_4\_finish = true;  
 return 100;  
 }  
 }  
 }  
 }  
 }  
 }  
  
  
 if (check\_4\_finish == false) {  
 for (int z = 0; z < 3; z++) {  
 for (int x = 0; x < 4; x++) {  
 long keep\_ai = ai;  
 long keep\_player1 = player1;  
 Board b = new Board();  
 b.setPlayer1(player1);  
 b.setPlayer2(ai);  
 if (x > 0) {  
 long[] players = b.rotate\_whole(mini\_boards\_to\_twist\_whole[x - 1][0], mini\_boards\_to\_twist\_whole[x - 1][1]);  
 keep\_ai = players[1];  
 keep\_player1 = players[0];  
 }  
 for (int i = 0; i < 10; i++) {  
 if (count\_1(Long.*toBinaryString*(keep\_player1 & masks\_total[z][i])) == 3 & count\_1(Long.*toBinaryString*(keep\_ai & masks\_total[z][i])) == 0) { *//0 opponents* long mask = 0b100000000000000000000000000000000000000000000000000000000000000L;  
 for (int j = 0; j < 5; j++) {  
 if (((mask >>> (indexes\_total[z][i][j] - 1)) & keep\_ai) == 0 & ((mask >>> (indexes\_total[z][i][j] - 1)) & keep\_player1) == 0) { *//not taken* need\_defence = true;  
 *index\_defense* = indexes\_total[z][i][j];  
 if (x > 0) {  
 *index\_defense* = b.rotate\_whole\_opp(*index\_defense*, mini\_boards\_to\_twist\_whole[x - 1][0], mini\_boards\_to\_twist\_whole[x - 1][1]); *//the right index* }  
 *mini\_board\_for\_twist\_defense* = this.mini\_board[x][(i % 5)]; *//the mini board to twist* if (x == 2) {  
 *direction\_rotating\_defense* = this.twist\_mini\_boards\_34[(i % 5)]; *//the direction to rotate* } else  
 *direction\_rotating\_defense* = this.twist\_mini\_boards\_12[(i % 5)]; *//the direction to rotate* check\_3\_finish = true; *//already found* }  
 }  
  
 }  
 }  
 }  
 }  
 }  
  
  
 if (need\_defence == false) {  
 *// Check for diagonal wins* int[] twist\_mini\_boards = {rand.nextInt(2) + 1, rand.nextInt(2) + 1, 2, 1};  
 int[] mini\_boards\_1 = {rand.nextInt(2) + 2, 1, 4, 4};  
 int[] mini\_boards\_2 = {rand.nextInt(2) + 2, 4, 1, 1};  
  
 long[][] masks\_diagonal1 = {{0b100010001000000000000000000100010000000000000000000000000000000L,  
 0b001010100000000000000000000100010000000000000000000000000000000L,  
 0b100010001000000000000000000000010100000000000000000000000000000L,  
 0b100010001000000000000000000001010000000000000000000000000000000L},  
  
 {0b000010001000000000000000000100010001000000000000000000000000000L,  
 0b000010001000000000000000000001010010000000000000000000000000000L,  
 0b001010000000000000000000000100010001000000000000000000000000000L,  
 0b000010100000000000000000000100010001000000000000000000000000000L}};  
  
 int[][] indexes\_1 = {{5, 32, 1, 9, 28}, {5, 32, 3, 7, 28}, {5, 32, 1, 9, 34}, {5, 32, 1, 9, 30}};  
 int[][] indexes2 = {{5, 32, 9, 28, 36}, {5, 32, 9, 30, 34}, {5, 32, 3, 28, 36}, {5, 32, 7, 28, 36}};  
  
 check\_4\_finish = false; *//check if there is 4 in a row* check\_3\_finish = false; *//check if there is 3 in a row* long keep\_ai = ai;  
 long keep\_player1 = player1;  
  
 int[][][] index\_total = {indexes\_1, indexes2};  
 int[][] mini\_boards\_total = {mini\_boards\_1, mini\_boards\_2};  
  
 for (int i = 0; i < 2; i++) {  
 for (int j = 0; j < 4; j++) {  
 if (count\_1(Long.*toBinaryString*(keep\_player1 & masks\_diagonal1[i][j])) == 4 & count\_1(Long.*toBinaryString*(keep\_ai & masks\_diagonal1[i][j])) == 0) { *//0 opponents* long mask = 0b100000000000000000000000000000000000000000000000000000000000000L;  
 for (int z = 0; z < 5; z++) {  
 if ((check\_4\_finish == false) & ((mask >>> (index\_total[i][j][z] - 1)) & keep\_ai) == 0 & ((mask >>> (index\_total[i][j][z] - 1)) & keep\_player1) == 0) { *//not taken* need\_defence = true;  
 *index\_defense* = index\_total[i][j][z];  
 *mini\_board\_for\_twist\_defense* = mini\_boards\_total[i][j]; *//the mini board to twist  
 direction\_rotating\_defense* = twist\_mini\_boards[j]; *//the direction to rotate* check\_4\_finish = true; *//already found* return 100;  
 }  
 }  
  
 }  
 }  
 }  
 for (int i = 0; i < 2; i++) {  
 for (int j = 0; j < 4; j++) {  
 if (count\_1(Long.*toBinaryString*(keep\_player1 & masks\_diagonal1[i][j])) == 3 & count\_1(Long.*toBinaryString*(keep\_ai & masks\_diagonal1[i][j])) == 0) { *//0 opponents* long mask = 0b100000000000000000000000000000000000000000000000000000000000000L;  
 for (int z = 0; z < 5; z++) {  
 if ((check\_3\_finish == false) & ((mask >>> (index\_total[i][j][z] - 1)) & keep\_ai) == 0 & ((mask >>> (index\_total[i][j][z] - 1)) & keep\_player1) == 0) { *//not taken* need\_defence = true;  
 *index\_defense* = index\_total[i][j][z];  
 *mini\_board\_for\_twist\_defense* = mini\_boards\_total[i][j]; *//the mini board to twist  
 direction\_rotating\_defense* = twist\_mini\_boards[j]; *//the direction to rotate* check\_3\_finish = true; *//already found* }  
 }  
  
 }  
 }  
 }  
  
  
 int[] mini\_boards\_3 = {rand.nextInt(2) \* 3 + 1, 1, 3, 3};  
 int[] mini\_boards\_4 = {rand.nextInt(2) \* 3 + 1, 3, 1, 1};  
  
 long[][] masks\_diagonal2 = {{0b000000000001010100001010000000000000000000000000000000000000000L,  
 0b000000000100010001001010000000000000000000000000000000000000000L,  
 0b000000000001010100100010000000000000000000000000000000000000000L,  
 0b000000000001010100000010001000000000000000000000000000000000000L},  
  
 {0b000000000000010100001010100000000000000000000000000000000000000L,  
 0b000000000000010100100010001000000000000000000000000000000000000L,  
 0b000000000000010001001010100000000000000000000000000000000000000L,  
 0b000000000100010000001010100000000000000000000000000000000000000L}};  
  
 int[][] indexes3 = {{14, 23, 12, 16, 21}, {14, 23, 10, 18, 21}, {14, 23, 12, 16, 19}, {14, 23, 12, 16, 27}};  
 int[][] indexes4 = {{14, 23, 16, 21, 25}, {14, 23, 16, 19, 27}, {14, 23, 18, 21, 25}, {14, 23, 10, 21, 25}};  
  
 check\_4\_finish = false; *//check if there is 4 in a row* check\_3\_finish = false; *//check if there is 3 in a row* int[][][] index\_total\_2 = {indexes3, indexes4};  
 int[][] mini\_boards\_total\_2 = {mini\_boards\_3, mini\_boards\_4};  
  
 for (int i = 0; i < 2; i++) {  
 for (int j = 0; j < 4; j++) {  
 if (count\_1(Long.*toBinaryString*(keep\_player1 & masks\_diagonal2[i][j])) == 4 & count\_1(Long.*toBinaryString*(keep\_ai & masks\_diagonal2[i][j])) == 0) { *//0 opponents* long mask = 0b100000000000000000000000000000000000000000000000000000000000000L;  
 for (int z = 0; z < 5; z++) {  
 if ((check\_4\_finish == false) & ((mask >>> (index\_total\_2[i][j][z] - 1)) & keep\_ai) == 0 & ((mask >>> (index\_total\_2[i][j][z] - 1)) & keep\_player1) == 0) { *//not taken* need\_defence = true;  
 *index\_defense* = index\_total\_2[i][j][z];  
 *mini\_board\_for\_twist\_defense* = mini\_boards\_total\_2[i][j]; *//the mini board to twist  
 direction\_rotating\_defense* = twist\_mini\_boards[j]; *//the direction to rotate* check\_4\_finish = true; *//already found* return 100;  
 }  
 }  
  
 }  
 }  
 }  
 for (int i = 0; i < 2; i++) {  
 for (int j = 0; j < 4; j++) {  
 if (count\_1(Long.*toBinaryString*(keep\_player1 & masks\_diagonal2[i][j])) == 3 & count\_1(Long.*toBinaryString*(keep\_ai & masks\_diagonal2[i][j])) == 0) { *//0 opponents* long mask = 0b100000000000000000000000000000000000000000000000000000000000000L;  
 for (int z = 0; z < 5; z++) {  
 if ((check\_3\_finish == false) & ((mask >>> (index\_total\_2[i][j][z] - 1)) & keep\_ai) == 0 & ((mask >>> (index\_total\_2[i][j][z] - 1)) & keep\_player1) == 0) { *//not taken* System.*out*.println("IM INSIDE");  
 need\_defence = true;  
 *index\_defense* = index\_total\_2[i][j][z];  
 *mini\_board\_for\_twist\_defense* = mini\_boards\_total\_2[i][j]; *//the mini board to twist  
 direction\_rotating\_defense* = twist\_mini\_boards[j]; *//the direction to rotate* check\_3\_finish = true; *//already found* }  
 }  
  
 }  
 }  
 }  
  
 }  
  
  
 if (need\_defence == false) {  
 *// Check for triple power play wins* long theTripleWin = 0b010001000000000100000000000010001000000000000000000000000000000L;  
 long theTripleWin2 = 0b000100010000000000001000000000100010000000000000000000000000000L;  
 long theTripleWin3 = 0b000000001010100000010100000000000000000000000000000000000000000L;  
 long theTripleWin4 = 0b000000000000001010000001010100000000000000000000000000000000000L;  
  
 long mask = 0;  
 *mini\_board\_for\_twist\_defense* = 0;  
 *index\_defense* = 0;  
  
 String new\_bits1 = Long.*toBinaryString*(board & theTripleWin);  
 String new\_bits2 = Long.*toBinaryString*(board & theTripleWin2);  
 String new\_bits3 = Long.*toBinaryString*(board & theTripleWin3);  
 String new\_bits4 = Long.*toBinaryString*(board & theTripleWin4);  
  
 long new\_bits\_ai1 = ai & theTripleWin;*//check if already defending* long new\_bits\_ai2 = ai & theTripleWin2;*//check if already defending* long new\_bits\_ai3 = ai & theTripleWin3;*//check if already defending* long new\_bits\_ai4 = ai & theTripleWin4;*//check if already defending* int count1 = count\_1(new\_bits1);*//count how many 1 are in the board for each way of winning - the first mask* int count2 = count\_1(new\_bits2);*//count how many 1 are in the board for each way of winning - the second mask* int count3 = count\_1(new\_bits3);*//count how many 1 are in the board for each way of winning - the third mask* int count4 = count\_1(new\_bits4);*//count how many 1 are in the board for each way of winning - the fourth mask* Long masks2[] = {0b010000000000000000000000000000000000000000000000000000000000000L,  
 0b000100000000000000000000000000000000000000000000000000000000000L,  
 0b000000001000000000000000000000000000000000000000000000000000000L,  
 0b000000000000001000000000000000000000000000000000000000000000000L};  
 int start\_indexex2[] = {2, 4, 9, 15}; *//index to start* int middle\_indexes2[] = {16, 21, 13, 24}; *//changed mini board index - to 2 board* int end\_indexes2[] = {29, 31, 20, 26}; *//changed mini board index - to 3 board* int counts2[] = {count1, count2, count3, count4}; *//the counts* Long boards2[] = {theTripleWin, theTripleWin2, theTripleWin3, theTripleWin4}; *//the mask* Long ai\_masks2[] = {new\_bits\_ai1, new\_bits\_ai2, new\_bits\_ai3, new\_bits\_ai4}; *//the ai mask* int indexes\_gap1[] = {4, 4, 2, 2}; *//the gap between the indexes* int indexes\_shift1[] = {6, 9, 0, 5}; *//the shift between the mini boards - 1 to 2* int indexes\_shift2[] = {13, 10, 7, 2}; *//the shift between the mini boards - 2 to 3* for (int i = 0; i < 4; i++) {  
 if ((counts2[i] > 2 && ai\_masks2[i] == 0)) { *//need defending - mask 1* need\_defence = true;  
 mask = masks2[i]; *//find the next index to put the ball  
 index\_defense* = start\_indexex2[i];  
 boolean found = false;  
 for (int j = 0; j < 2; j++) {  
 if (found == false) {  
 if ((mask & (board & boards2[i])) == 0) { *//found an index for defending* found = true;  
 *mini\_board\_for\_twist\_defense* = calc\_mini\_board\_diagonal\_triple(*index\_defense*, i); *//the mini board to twist* } else *index\_defense* = *index\_defense* + indexes\_gap1[i];  
 mask = mask >> indexes\_gap1[i];  
 }  
 }  
  
  
 if (found == false) {  
 mask = mask >> indexes\_shift1[i];  
 *index\_defense* = middle\_indexes2[i];  
 }  
 if (found == false) {  
 if ((mask & (board & boards2[i])) == 0) { *//found an index for defending* found = true;  
 *mini\_board\_for\_twist\_defense* = calc\_mini\_board\_diagonal\_triple(*index\_defense*, i);*//the mini board to twist* }  
 }  
  
  
 if (found == false) {  
 mask = mask >> indexes\_shift2[i];  
 *index\_defense* = end\_indexes2[i];  
 }  
 for (int j = 0; j < 2; j++) {  
 if (found == false) {  
 if ((mask & (board & boards2[i])) == 0) { *//found an index for defending* found = true;  
 *mini\_board\_for\_twist\_defense* = calc\_mini\_board\_diagonal\_triple(*index\_defense*, i);*//the mini board to twist* } else *index\_defense* = *index\_defense* + indexes\_gap1[i];  
 mask = mask >> indexes\_gap1[i];  
 }  
 }  
  
 }  
 }  
 }  
  
  
 if (need\_defence == true)  
 grade = 100; *//sets defence grade* else grade = 0;  
*// direction\_rotating\_defence=rand.nextInt(2)+1;* return grade;  
 }  
  
  
  
  
  
 */\*\*  
 \* move 1, putting the first ball  
 \*  
 \* @param playing\_strategy- the strategy 1 or 2  
 \*/* private void player\_move\_1(int playing\_strategy) {  
 if (check\_for\_lose\_sides(current\_strategy) == false & check\_for\_lose(current\_strategy) == false) {  
 boolean found\_empty\_place=false;  
 int index=0;  
 int[] possible\_moves\_12\_copy = {2, 4, 6, 8, 29, 31, 33, 35}; *//possible indexes to win the first strategy* int[] mini\_board\_12\_possible1\_copy= {random.nextInt(2) + 2, random.nextInt(2) + 2, 1, 1, 4, 4,  
 random.nextInt(2) + 2, random.nextInt(2) + 2}; *//the mini boards to twist for option 1(the better one)* int [] twist\_board\_12\_possible1\_copy= {random.nextInt(2) + 1, random.nextInt(2) + 1, 1, 2, 2, 1,  
 random.nextInt(2) + 1, random.nextInt(2) + 1}; *//the twists of the mini boards for option 1* int[] possible\_moves\_34\_copy = {11, 13, 15, 17, 20, 22, 24, 26}; *//possible indexes to win the second strategy* int[] mini\_board\_34\_possible1\_copy = {random.nextInt(2) \* 3 + 1, 2, random.nextInt(2) \* 3 + 1,  
 2, 3, random.nextInt(2) \* 3 + 1, 3, random.nextInt(2) \* 3 + 1}; *//the mini boards to twist for option 1(the better one)* int[] twist\_board\_34\_possible1\_copy = {random.nextInt(2) + 1, 2, random.nextInt(2) + 1,  
 1, 1, random.nextInt(2) + 1, 2, random.nextInt(2) + 1}; *//the twists of the mini boards for option 1* while(found\_empty\_place==false){  
 long mask\_possible = 0b100000000000000000000000000000000000000000000000000000000000000L; *//follow each bit* int i = rand.nextInt(8-index++);  
 if (playing\_strategy == 1) {  
 if (((mask\_possible >>> (possible\_moves\_12\_copy[i] - 1)) & player1) == 0 &  
 ((mask\_possible >>> (possible\_moves\_12\_copy[i] - 1)) & ai) == 0) *//place not taken* {  
 found\_empty\_place=true;  
 AiPlayer.*index\_triple* = possible\_moves\_12\_copy[i];  
 AiPlayer.*mini\_board\_for\_twist\_triple* = mini\_board\_12\_possible1\_copy[i];  
 AiPlayer.*direction\_rotating\_triple* =twist\_board\_12\_possible1\_copy[i];  
 }  
 else { *//not found* possible\_moves\_12\_copy[i]=possible\_moves\_12\_copy[8-index];  
 mini\_board\_12\_possible1\_copy[i]=mini\_board\_12\_possible1\_copy[8-index];  
 twist\_board\_12\_possible1\_copy[i]=twist\_board\_12\_possible1\_copy[8-index];  
 }  
 } else if (playing\_strategy == 2) {  
 if (((mask\_possible >>> (possible\_moves\_34\_copy[i] - 1)) & player1) == 0 &  
 ((mask\_possible >>> (possible\_moves\_34\_copy[i] - 1)) & ai) == 0) *//place not taken* {  
 found\_empty\_place=true;  
 AiPlayer.*index\_triple* = possible\_moves\_34\_copy[i];  
 AiPlayer.*mini\_board\_for\_twist\_triple* = mini\_board\_34\_possible1\_copy[i];  
 AiPlayer.*direction\_rotating\_triple* = twist\_board\_34\_possible1\_copy[i];  
 }  
 else {  
 possible\_moves\_34\_copy[i]=possible\_moves\_34\_copy[8-index];  
 mini\_board\_34\_possible1\_copy[i]=mini\_board\_34\_possible1\_copy[8-index];  
 twist\_board\_34\_possible1\_copy[i]=twist\_board\_34\_possible1\_copy[8-index];  
 }  
 }  
 }  
 }  
 }  
  
 */\*\*  
 \* move 2, putting the second ball  
 \*/* private void player\_move\_2() {  
 long mask12 = 0b010000000000000000000000000000000000000000000000000000000000000L; *//mask to follow the bits - first strategy* long mask34 = 0b000000000010000000000000000000000000000000000000000000000000000L; *//mask to follow the bits - second strategy* boolean check\_finish\_12 = false;  
 boolean check\_finish\_34 = false;  
 if (check\_for\_lose(current\_strategy) == false & check\_for\_lose\_sides(current\_strategy) == false) {  
 for (int i = 0; i < 8; i++) {  
 if (check\_finish\_12 == false) {  
 long mask\_possible = 0b100000000000000000000000000000000000000000000000000000000000000L; *//follow each bit* if ((mask12 & ai) != 0) {  
 check\_finish\_12 = true; *//not continue the for* if (((mask\_possible >>> (indexes\_move2\_12\_possible1[i] - 1)) & player1) == 0 &  
 ((mask\_possible >>> (indexes\_move2\_12\_possible1[i] - 1)) & ai) == 0) { *//check opponent* AiPlayer.*index\_triple* = indexes\_move2\_12\_possible1[i];  
 AiPlayer.*mini\_board\_for\_twist\_triple* = mini\_board\_12\_possible1[i];  
 AiPlayer.*direction\_rotating\_triple* = twist\_board\_12\_possible1[i];  
  
 } else if (((mask\_possible >>> (indexes\_move2\_12\_possible2[i] - 1)) & player1) == 0 &  
 ((mask\_possible >>> (indexes\_move2\_12\_possible2[i] - 1)) & ai) == 0) {  
 AiPlayer.*index\_triple* = indexes\_move2\_12\_possible2[i];  
 AiPlayer.*mini\_board\_for\_twist\_triple* = mini\_board\_12\_possible2[i];  
 AiPlayer.*direction\_rotating\_triple* = twist\_board\_12\_possible2[i];  
 }  
 }  
 if (i == 3) { *//changing mini board* mask12 = mask12 >>> 21;  
 } else mask12 = mask12 >>> 2;  
 }  
 }  
 if (check\_finish\_12 == false) {  
 for (int i = 0; i < 8; i++) {  
 if (check\_finish\_34 == false) {  
 long mask\_possible = 0b100000000000000000000000000000000000000000000000000000000000000L; *//follow each bit* if ((mask34 & ai) != 0) {  
 check\_finish\_34 = true;  
 System.*out*.println(Long.*toBinaryString*(mask\_possible >>> (indexes\_move2\_34\_possible1[i] - 1)));  
 if (((mask\_possible >>> (indexes\_move2\_34\_possible1[i] - 1)) & player1) == 0 &  
 ((mask\_possible >>> (indexes\_move2\_34\_possible1[i] - 1)) & ai) == 0) { *//check opponent* AiPlayer.*index\_triple* = indexes\_move2\_34\_possible1[i];  
 AiPlayer.*mini\_board\_for\_twist\_triple* = mini\_board\_34\_possible1[i];  
 AiPlayer.*direction\_rotating\_triple* = twist\_board\_34\_possible1[i];  
  
 } else if (((mask\_possible >>> (indexes\_move2\_34\_possible2[i] - 1)) & player1) == 0 &  
 ((mask\_possible >>> (indexes\_move2\_34\_possible2[i] - 1)) & ai) == 0) {  
 AiPlayer.*index\_triple* = indexes\_move2\_34\_possible2[i];  
 AiPlayer.*mini\_board\_for\_twist\_triple* = mini\_board\_34\_possible2[i];  
 AiPlayer.*direction\_rotating\_triple* = twist\_board\_34\_possible2[i];  
 }  
 }  
 if (i == 3) { *//changing mini board* mask34 = mask34 >>> 3;  
 } else mask34 = mask34 >>> 2;  
 }  
 }  
 }  
 }  
 }  
  
 */\*\*  
 \* check for losing - 2 bits of opponent across each other  
 \*  
 \* @return  
 \*/* private boolean check\_for\_lose(int strategy) {  
 long mask\_lose1 = 0b010000010000000000000000000000000000000000000000000000000000000L;*//>>27, >>9, >>18* long mask\_lose2 = 0b000101000000000000000000000000000000000000000000000000000000000L;*//>>27, >>18, >>9* long mask\_win\_1;  
 long mask\_win\_2;  
 if (strategy == 1) {  
 mask\_win\_1 = 0b010001000000000100000000000010001000000000000000000000000000000L; *//winning 1* mask\_win\_2 = 0b000100010000000000001000000000100010000000000000000000000000000L; *//winning 2* } else {  
 mask\_win\_1 = 0b000000001010100000010100000000000000000000000000000000000000000L;  
 mask\_win\_2 = 0b000000000000001010000001010100000000000000000000000000000000000L;  
 }  
 for (int i = 0; i < 2; i++) {  
 if ((count\_1(Long.*toBinaryString*(ai & mask\_win\_1)) == 4 & count\_1(Long.*toBinaryString*(player1 & mask\_win\_1)) == 0) ||  
 (count\_1(Long.*toBinaryString*(ai & mask\_win\_2)) == 4 & count\_1(Long.*toBinaryString*(player1 & mask\_win\_2)) == 0)) {  
 grade = 120; *//one turn before winning* }  
 }  
  
 if ((((mask\_lose1 & player1) == mask\_lose1) || (((mask\_lose1 >> 27) & player1) == (mask\_lose1 >> 27)) ||  
 ((mask\_lose2 & player1) == mask\_lose2) || (((mask\_lose2 >> 27) & player1) == (mask\_lose2 >> 27)))  
 & ((((mask\_lose1 >> 9) & player1) == (mask\_lose1 >> 9)) || (((mask\_lose1 >> 18) & player1) == (mask\_lose1 >> 18)) ||  
 (((mask\_lose2 >> 9) & player1) == (mask\_lose2 >> 9)) || (((mask\_lose2 >> 18) & player1) == (mask\_lose2 >> 18)))) { *//both strategies are off* grade = 0;  
 return true;  
 }  
  
 if (strategy == 1) {  
 if (((mask\_lose1 & player1) == mask\_lose1) || (((mask\_lose1 >> 27) & player1) == (mask\_lose1 >> 27)) ||  
 ((mask\_lose2 & player1) == mask\_lose2) || (((mask\_lose2 >> 27) & player1) == (mask\_lose2 >> 27))) {  
 AiPlayer.*player\_move* = 1; *//go for the second strategy* current\_strategy = 2;  
 player\_move\_1(2);  
 return true;  
 }  
 } else if ((((mask\_lose1 >> 9) & player1) == (mask\_lose1 >> 9)) || (((mask\_lose1 >> 18) & player1) == (mask\_lose1 >> 18)) ||  
 (((mask\_lose2 >> 9) & player1) == (mask\_lose2 >> 9)) || (((mask\_lose2 >> 18) & player1) == (mask\_lose2 >> 18))) {  
 AiPlayer.*player\_move* = 1; *//go for the first strategy* current\_strategy = 1;  
 player\_move\_1(1);  
 return true;  
 }  
 return false;  
 }  
  
 */\*\*  
 \* check for lose if all 4 corners  
 \*  
 \* @return  
 \*/* private boolean check\_for\_lose\_sides(int strategy) {  
 long mask\_lose = 0b101000101000000000000000000000000000000000000000000000000000000L;  
 if (((((mask\_lose >> 9) & player1) == (mask\_lose >> 9)) || (((mask\_lose >> 18) & player1) == (mask\_lose >> 18)))  
 & ((((mask\_lose) & player1) == (mask\_lose)) || (((mask\_lose >> 27) & player1) == (mask\_lose >> 27)))) { *//both strategies are off* grade = 0;  
 return true;  
 }  
 if (strategy == 1) {  
 if ((((mask\_lose >> 9) & player1) == (mask\_lose >> 9)) || (((mask\_lose >> 18) & player1) == (mask\_lose >> 18))) {  
 AiPlayer.*player\_move* = 1; *//go for the second strategy* current\_strategy = 2;  
 player\_move\_1(2);  
 return true;  
 }  
 } else if ((((mask\_lose) & player1) == (mask\_lose)) || (((mask\_lose >> 27) & player1) == (mask\_lose >> 27))) {  
 AiPlayer.*player\_move* = 1; *//go for the first strategy* current\_strategy = 1;  
 player\_move\_1(1);  
 return true;  
 }  
 return false;  
 }  
  
 */\*\*  
 \* sets the next place that is not taken by any of the players - moving from 1 mini board to another  
 \*  
 \* @param playing\_strategy - the current strategy  
 \* @param start\_index - the start index to look for - 0 to first mini board and 4 to second mini board  
 \* @param end\_index - the end index to look for - 3 to first mini board and 7 to second mini board  
 \*/* private void start\_next\_mini\_board(int playing\_strategy, int start\_index, int end\_index) {  
 boolean already\_taken = true; *//if the ball is taken* while (already\_taken == true) {  
 long mask12 = 0b010000000000000000000000000000000000000000000000000000000000000L; *//checks the first strategy* long mask34 = 0b000000000010000000000000000000000000000000000000000000000000000L; *//checks the second strategy* int index = random.nextInt(start\_index, end\_index + 1); *//random index* if (playing\_strategy == 1) {  
 AiPlayer.*index\_triple* = possible\_moves\_12[index];  
 if (index < 4 & index > -1) { *//first mini board* if ((player1 & (mask12 >>> (index \* 2))) == 0 & (ai & (mask12 >>> (index \* 2))) == 0) { *//this place is not taken* already\_taken = false;  
 AiPlayer.*mini\_board\_for\_twist\_triple* = this.mini\_board\_12\_possible1[index];  
 AiPlayer.*direction\_rotating\_triple* = this.twist\_board\_12\_possible1[index];  
 }  
 } else {  
 if ((player1 & (mask12 >>> (27 + ((index % 4) \* 2)))) == 0 & (ai & (mask12 >>> (27 + ((index % 4) \* 2)))) == 0) { *//this place in not taken* already\_taken = false;  
 AiPlayer.*mini\_board\_for\_twist\_triple* = this.mini\_board\_12\_possible1[index];  
 AiPlayer.*direction\_rotating\_triple* = this.twist\_board\_12\_possible1[index];  
 }  
 }  
  
 } else {  
 AiPlayer.*index\_triple* = possible\_moves\_34[index];  
 if (index < 4 & index > -1) { *//first mini board* if ((player1 & (mask34 >>> (index \* 2))) == 0 & (ai & (mask34 >>> (index \* 2))) == 0) { *//this place in not taken* already\_taken = false;  
 AiPlayer.*mini\_board\_for\_twist\_triple* = this.mini\_board\_34\_possible1[index];  
 AiPlayer.*direction\_rotating\_triple* = this.twist\_board\_34\_possible1[index];  
 }  
 } else {  
 if ((player1 & (mask34 >>> (9 + ((index % 4) \* 2)))) == 0 & (ai & (mask34 >>> (9 + ((index % 4) \* 2)))) == 0) { *//this place in not taken* already\_taken = false;  
 AiPlayer.*mini\_board\_for\_twist\_triple* = this.mini\_board\_34\_possible1[index];  
 AiPlayer.*direction\_rotating\_triple* = this.twist\_board\_34\_possible1[index];  
 }  
 }  
 }  
 }  
 }  
  
 */\*\*  
 \* move 3, putting the third ball  
 \*/* private void player\_move\_3() {  
 if (check\_for\_lose(current\_strategy) == false & check\_for\_lose\_sides(current\_strategy) == false) {  
 boolean play\_board\_12 = false;  
 boolean play\_board\_34 = false;  
 long mask12\_1 = 0b010000000000000000000000000000000000000000000000000000000000000L; *//mask for first mini board in  
 // strategy 1* long mask34\_1 = 0b000000000010000000000000000000000000000000000000000000000000000L; *//mask for first mini board in  
 // strategy 2* long mask12\_2 = mask12\_1 >>> 27;*//mask for second mini board in strategy 1* long mask34\_2 = mask34\_1 >>> 9;*//mask for second mini board in strategy 2* for (int i = 0; i < 4; i++) { *//first mini board* if (current\_strategy == 1) {  
 if (play\_board\_12 == false) {  
 if ((mask12\_1 & ai) != 0) { *//already has a ball in there, so needs to take the second one* play\_board\_12 = true;  
 }  
 }  
 } else {  
 if (play\_board\_34 == false) {  
 if ((mask34\_1 & ai) != 0) { *//already has a ball in there, so needs to take the second one* play\_board\_34 = true;  
 }  
 }  
 }  
  
 mask12\_1 = mask12\_1 >>> 2;  
 mask34\_1 = mask34\_1 >>> 2;  
 }  
 if (play\_board\_12 == true) {  
 start\_next\_mini\_board(1, 4, 7); *//put a ball* this.last\_turn\_mini\_board = 4;  
 this.last\_turn\_playing\_strategy = 1;  
 } else if (play\_board\_34 == true) {  
 start\_next\_mini\_board(2, 4, 7); *//put a ball* this.last\_turn\_playing\_strategy = 2;  
 this.last\_turn\_mini\_board = 3;  
 }  
 if (play\_board\_12 == false & play\_board\_34 == false) {  
 for (int i = 0; i < 4; i++) { *//first mini board* if (current\_strategy == 1) {  
 if (play\_board\_12 == false) {  
 if ((mask12\_2 & ai) != 0) { *//already has a ball in there, so needs to take the second one* play\_board\_12 = true;  
 }  
 }  
 } else {  
 if (play\_board\_34 == false) {  
 if ((mask34\_2 & ai) != 0) { *//already has a ball in there, so needs to take the second one* play\_board\_34 = true;  
 }  
 }  
 }  
  
 mask12\_2 = mask12\_2 >>> 2;  
 mask34\_2 = mask34\_2 >>> 2;  
 }  
  
 if (play\_board\_12 == true) {  
 start\_next\_mini\_board(1, 0, 3); *//put a ball* this.last\_turn\_playing\_strategy = 1;  
 this.last\_turn\_mini\_board = 1;  
 } else if (play\_board\_34 == true) {  
 start\_next\_mini\_board(2, 0, 3); *//put a ball* this.last\_turn\_playing\_strategy = 2;  
 this.last\_turn\_mini\_board = 2;  
 }  
 }  
 }  
 }  
  
  
 */\*\*  
 \* finds the next index to put that is on the winning strategy  
 \*  
 \* @param mini\_board - mini board that tells us the mask to work with  
 \* @return  
 \*/* private int find\_index(int mini\_board) {  
 if (this.last\_turn\_playing\_strategy == 1) {  
 int[] places\_win = {};  
 int[] places1 = {2, 6, 16, 29, 33}; *//indexes of the winning places - first mini board* int[] places2 = {4, 8, 21, 31, 35}; *//indexes of the winning places - second mini board* long mask = 0b100000000000000000000000000000000000000000000000000000000000000L; *//mask to check these indexes* if (mini\_board == 2) { *//first* places\_win = places1;  
 } else if (mini\_board == 3) {*//second* places\_win = places2;  
 }  
 for (int i = 0; i < 5; i++) {  
 if (((mask >>> (places\_win[i] - 1)) & ai) == 0 & ((mask >>> (places\_win[i] - 1)) & player1) == 0)*//empty* {  
 return places\_win[i];  
 }  
 }  
 for (int i = 1; i < 37; i++) { *//if all the places are taken, will choose another one* if (((mask >>> (i - 1)) & ai) == 0 & ((mask >>> (i - 1)) & player1) == 0)*//empty* {  
 return i;  
 }  
 }  
 return -1;  
 } else {  
 int[] places\_win = {};  
 int[] places1 = {9, 11, 13, 20, 22}; *//indexes of the winning places - first mini board* int[] places2 = {15, 17, 24, 26, 28}; *//indexes of the winning places - second mini board* long mask = 0b100000000000000000000000000000000000000000000000000000000000000L; *//mask to check these indexes* if (mini\_board == 1) { *//first* places\_win = places1;  
 } else if (mini\_board == 4) { *//second* places\_win = places2;  
 }  
 for (int i = 0; i < 5; i++) {  
 if (((mask >>> (places\_win[i] - 1)) & ai) == 0 & ((mask >>> (places\_win[i] - 1)) & player1) == 0)*//empty* {  
 return places\_win[i];  
 }  
 }  
 for (int i = 1; i < 37; i++) { *//if all the places are taken, will choose another one* if (((mask >>> (i - 1)) & ai) == 0 & ((mask >>> (i - 1)) & player1) == 0)*//empty* {  
 return i;  
 }  
 }  
 return -1;  
 }  
 }  
  
  
 */\*\*  
 \* finds the best rotation for next move  
 \*  
 \* @param mini\_board - mini board that tells us the mask to work with  
 \*/* private void find\_rotate(int mini\_board) {  
 if (this.last\_turn\_playing\_strategy == 1) {  
 int[][] places\_win1 = {}; *//first indexes* int[][] places\_win2 = {}; *//second indexes* int[] mini\_board\_to\_twist1 = {}; *//first mini boards* int[] mini\_board\_to\_twist2 = {}; *//second mini boards* int[] twist1 = {}; *//first rotation* int[] twist2 = {}; *//second rotation* int[] mini\_board11 = {1, 1, 1};  
 int[] mini\_board12 = {3, 4, 4, 4};  
 int[] twisting11 = {2, 1, random.nextInt(2) + 1};  
 int[] twisting12 = {random.nextInt(2) + 1, 2, 1, random.nextInt(2) + 1};  
 int[] mini\_board21 = {1, 1, 1};  
 int[] mini\_board22 = {2, 4, 4, 4};  
 int[] twisting21 = {1, 2, random.nextInt(2) + 1};  
 int[] twisting22 = {random.nextInt(2) + 1, 1, 2, random.nextInt(2) + 1};  
 int[][] places11 = {{2, 6}, {2, 4}, {6, 8}, {8, 4}}; *//the pairs to check* int[][] places12 = {{29, 33}, {29, 31}, {33, 35}, {31, 35}}; *//the pairs to check* int[][] places21 = {{4, 8}, {2, 4}, {6, 8}, {2, 6}}; *//the pairs to check* int[][] places22 = {{31, 35}, {29, 31}, {33, 35}, {29, 33}}; *//the pairs to check* long mask = 0b100000000000000000000000000000000000000000000000000000000000000L;  
 if (mini\_board == 2) { *//first* places\_win1 = places11;  
 places\_win2 = places12;  
 mini\_board\_to\_twist1 = mini\_board11;  
 mini\_board\_to\_twist2 = mini\_board12;  
 twist1 = twisting11;  
 twist2 = twisting12;  
 } else if (mini\_board == 3) { *//second* places\_win1 = places21;  
 places\_win2 = places22;  
 mini\_board\_to\_twist1 = mini\_board21;  
 mini\_board\_to\_twist2 = mini\_board22;  
 twist1 = twisting21;  
 twist2 = twisting22;  
 }  
 boolean check\_twice = false;  
 boolean check\_second\_mini\_board = false;  
 for (int i = 0; i < 4; i++) {  
 if (check\_twice == false) {  
 if ((((mask >>> (places\_win1[0][0] - 1)) & ai) != 0) & (((mask >>> (places\_win1[0][1] - 1)) & ai) != 0)) { *//the first of the fourth* check\_twice = true;  
 check\_second\_mini\_board = true;  
 } else if ((((mask >>> (places\_win1[i][0] - 1)) & ai) != 0) & (((mask >>> (places\_win1[i][1] - 1)) & ai) != 0)) {*//both exists* check\_twice = true;  
 AiPlayer.*mini\_board\_for\_twist\_triple* = mini\_board\_to\_twist1[i - 1];  
 AiPlayer.*direction\_rotating\_triple* = twist1[i - 1];  
 }  
 }  
 }  
 if (check\_second\_mini\_board == true) {  
 check\_twice = false;  
 for (int i = 0; i < 4; i++) {  
 if (check\_twice == false) {  
 if ((((mask >>> (places\_win2[i][0] - 1)) & ai) != 0) & (((mask >>> (places\_win2[i][1] - 1)) & ai) != 0)) {*//both exists* check\_twice = true;  
 AiPlayer.*mini\_board\_for\_twist\_triple* = mini\_board\_to\_twist2[i];  
 AiPlayer.*direction\_rotating\_triple* = twist2[i];  
 }  
 }  
 }  
 }  
 } else {  
 int[][] places\_win1 = {}; *//first indexes* int[][] places\_win2 = {}; *//second indexes* int[] mini\_board\_to\_twist1 = {}; *//first mini boards* int[] mini\_board\_to\_twist2 = {}; *//second mini boards* int[] twist1 = {}; *//first twists* int[] twist2 = {}; *//second twists* int[] mini\_board11 = {2, 2, 2};  
 int[] mini\_board12 = {4, 3, 3, 3};  
 int[] twisting11 = {2, 1, random.nextInt(2) + 1};  
 int[] twisting12 = {random.nextInt(2) + 1, 2, 1, random.nextInt(2) + 1};  
 int[] mini\_board21 = {2, 2, 2};  
 int[] mini\_board22 = {1, 3, 3, 3};  
 int[] twisting21 = {1, 2, random.nextInt(2) + 1};  
 int[] twisting22 = {random.nextInt(2) + 1, 1, 2, random.nextInt(2) + 1};  
 int[][] places11 = {{11, 13}, {13, 17}, {11, 15}, {15, 17}};  
 int[][] places12 = {{20, 22}, {22, 26}, {20, 24}, {24, 26}};  
 int[][] places21 = {{15, 17}, {13, 17}, {11, 15}, {11, 13}};  
 int[][] places22 = {{24, 26}, {22, 26}, {20, 24}, {20, 22}};  
  
 long mask = 0b100000000000000000000000000000000000000000000000000000000000000L;  
 if (mini\_board == 1) { *//first* places\_win1 = places11;  
 places\_win2 = places12;  
 mini\_board\_to\_twist1 = mini\_board11;  
 mini\_board\_to\_twist2 = mini\_board12;  
 twist1 = twisting11;  
 twist2 = twisting12;  
 } else if (mini\_board == 4) { *//second* places\_win1 = places21;  
 places\_win2 = places22;  
 mini\_board\_to\_twist1 = mini\_board21;  
 mini\_board\_to\_twist2 = mini\_board22;  
 twist1 = twisting21;  
 twist2 = twisting22;  
 }  
 boolean check\_twice = false;  
 boolean check\_second\_mini\_board = false;  
 for (int i = 0; i < 4; i++) {  
 if (check\_twice == false) {  
 if ((((mask >>> (places\_win1[0][0] - 1)) & ai) != 0) & (((mask >>> (places\_win1[0][1] - 1)) & ai) != 0)) {*//the first one* check\_twice = true;  
 check\_second\_mini\_board = true;  
 } else if ((((mask >>> (places\_win1[i][0] - 1)) & ai) != 0) & (((mask >>> (places\_win1[i][1] - 1)) & ai) != 0)) {*//both exists* check\_twice = true;  
 AiPlayer.*mini\_board\_for\_twist\_triple* = mini\_board\_to\_twist1[i - 1];  
 AiPlayer.*direction\_rotating\_triple* = twist1[i - 1];  
 }  
 }  
 }  
 if (check\_second\_mini\_board == true) {  
 check\_twice = false;  
 for (int i = 0; i < 4; i++) {  
 if (check\_twice == false) {  
 if ((((mask >>> (places\_win2[i][0] - 1)) & ai) != 0) & (((mask >>> (places\_win2[i][1] - 1)) & ai) != 0)) {*//both exists* check\_twice = true;  
 AiPlayer.*mini\_board\_for\_twist\_triple* = mini\_board\_to\_twist2[i];  
 AiPlayer.*direction\_rotating\_triple* = twist2[i];  
 }  
 }  
 }  
 }  
 }  
 }  
  
 */\*\*  
 \* checks for the next move after the knowledge of two whites  
 \*/* private void my\_color\_ball() {  
 if (this.last\_turn\_playing\_strategy == 1) {  
 long mask\_win\_1 = 0b010001000000000100000000000010001000000000000000000000000000000L; *//winning 1* long mask\_win\_2 = 0b000100010000000000001000000000100010000000000000000000000000000L; *//winning 2* int[] places1 = {16, 10, 18, 12}; *//indexes of possible to the third mini board* int[] twisting1 = {1, 1, 2, random.nextInt(2) + 1}; *//twisting for the third mini board* int[] places = {};  
 int[] twist = {};  
 int[] places2 = {21, 19, 27, 25}; *//indexes of possible to the third mini board* int[] twisting2 = {2, 2, 1, random.nextInt(2) + 1}; *//twisting for the third mini board* int mini\_board = 0;  
  
 if ((count\_1(Long.*toBinaryString*(mask\_win\_1 & ai)) > count\_1(Long.*toBinaryString*(mask\_win\_2 & ai)))  
 || ((count\_1(Long.*toBinaryString*(mask\_win\_1 & ai)) == count\_1(Long.*toBinaryString*(mask\_win\_2 & ai))) &  
 (((0b100000000000000000000000000000000000000000000000000000000000000L >>> 15) & ai) != 0))) {  
 *//first is better than the second, or first is better than the second* places = places1;  
 twist = twisting1;  
 mini\_board = 2;  
 } else if ((count\_1(Long.*toBinaryString*(mask\_win\_1 & ai)) < count\_1(Long.*toBinaryString*(mask\_win\_2 & ai)))  
 || ((count\_1(Long.*toBinaryString*(mask\_win\_1 & ai)) == count\_1(Long.*toBinaryString*(mask\_win\_2 & ai))) &  
 (((0b100000000000000000000000000000000000000000000000000000000000000L >>> 20) & ai) != 0))) {  
 *//second is better than the first, or second is better than the first* places = places2;  
 twist = twisting2;  
 mini\_board = 3;  
 } else {  
 places = places1;  
 twist = twisting1;  
 mini\_board = 2;  
 }  
  
 boolean check = false;  
 for (int i = 0; i < 4; i++) {  
 if (check == false) {  
 if (((0b100000000000000000000000000000000000000000000000000000000000000L >>> (places[i] - 1)) & ai) == 0  
 & ((0b100000000000000000000000000000000000000000000000000000000000000L >>> (places[i] - 1)) & player1) == 0)*//empty* {  
 check = true;  
 if (i > 0) {  
 AiPlayer.*index\_triple* = places[i];  
 AiPlayer.*mini\_board\_for\_twist\_triple* = mini\_board;  
 AiPlayer.*direction\_rotating\_triple* = twist[i];  
 } else if (i == 0) {  
 AiPlayer.*index\_triple* = places[i];  
 find\_rotate(mini\_board);  
 }  
 } else if (((0b100000000000000000000000000000000000000000000000000000000000000L >>> (places[i] - 1)) & ai) != 0)*//white* {  
 check = true;  
 if (i > 0) {  
 AiPlayer.*index\_triple* = find\_index(mini\_board);  
 AiPlayer.*mini\_board\_for\_twist\_triple* = mini\_board;  
 AiPlayer.*direction\_rotating\_triple* = twist[i];  
 } else if (i == 0) {  
 AiPlayer.*index\_triple* = find\_index(mini\_board);  
 find\_rotate(mini\_board);  
 }  
 }  
 }  
 }  
 } else { *//strategy 2* long mask\_win\_1 = 0b000000001010100000010100000000000000000000000000000000000000000L;  
 long mask\_win\_2 = 0b000000000000001010000001010100000000000000000000000000000000000L;  
 int[] places1 = {9, 3, 7, 1};  
 int[] twisting1 = {1, 2, 1, random.nextInt(2) + 1};  
 int[] places = {};  
 int[] twist = {};  
 int[] places2 = {28, 30, 34, 36};  
 int[] twisting2 = {1, 1, 2, random.nextInt(2) + 1};  
 int mini\_board = 0;  
  
  
 if ((count\_1(Long.*toBinaryString*(mask\_win\_1 & ai)) > count\_1(Long.*toBinaryString*(mask\_win\_2 & ai)))  
 || ((count\_1(Long.*toBinaryString*(mask\_win\_1 & ai)) == count\_1(Long.*toBinaryString*(mask\_win\_2 & ai))) &  
 (((0b100000000000000000000000000000000000000000000000000000000000000L >>> 8) & ai) != 0))) {  
 places = places1;  
 twist = twisting1;  
 mini\_board = 1;  
 } else if ((count\_1(Long.*toBinaryString*(mask\_win\_1 & ai)) < count\_1(Long.*toBinaryString*(mask\_win\_2 & ai)))  
 || ((count\_1(Long.*toBinaryString*(mask\_win\_1 & ai)) == count\_1(Long.*toBinaryString*(mask\_win\_2 & ai))) &  
 (((0b100000000000000000000000000000000000000000000000000000000000000L >>> 27) & ai) != 0))) {  
 places = places2;  
 twist = twisting2;  
 mini\_board = 4;  
 } else {  
 places = places1;  
 twist = twisting1;  
 mini\_board = 1;  
 }  
  
 boolean check = false;  
 for (int i = 0; i < 4; i++) {  
 if (check == false) {  
 if (((0b100000000000000000000000000000000000000000000000000000000000000L >>> (places[i] - 1)) & ai) == 0  
 & ((0b100000000000000000000000000000000000000000000000000000000000000L >>> (places[i] - 1)) & player1) == 0)*//empty* {  
 check = true;  
 if (i > 0) {  
 AiPlayer.*index\_triple* = places[i];  
 AiPlayer.*mini\_board\_for\_twist\_triple* = mini\_board;  
 AiPlayer.*direction\_rotating\_triple* = twist[i];  
 } else if (i == 0) {  
 AiPlayer.*index\_triple* = places[i];  
 find\_rotate(mini\_board);  
 }  
 } else if (((0b100000000000000000000000000000000000000000000000000000000000000L >>> (places[i] - 1)) & ai) != 0)*//white* {  
 check = true;  
 if (i > 0) {  
 AiPlayer.*index\_triple* = find\_index(mini\_board);  
 AiPlayer.*mini\_board\_for\_twist\_triple* = mini\_board;  
 AiPlayer.*direction\_rotating\_triple* = twist[i];  
 } else if (i == 0) {  
 AiPlayer.*index\_triple* = find\_index(mini\_board);  
 find\_rotate(mini\_board);  
 }  
 }  
 }  
 }  
 }  
  
 }  
  
 */\*\*  
 \* move above 3  
 \*/* private void player\_move\_above\_3() {  
 if (check\_for\_lose(current\_strategy) == false & check\_for\_lose\_sides(current\_strategy) == false) {  
 if (this.last\_turn\_playing\_strategy == 1) {  
 long mask = 0b010000000000000000000000000000000000000000000000000000000000000L;  
 boolean check = false;  
 for (int i = 0; i < 4; i++) {  
 if (check == false) {  
 long mask\_possible = 0b100000000000000000000000000000000000000000000000000000000000000L;  
 if ((mask & ai) != 0) {  
 check = true;  
 if (((mask\_possible >>> (indexes\_move2\_12\_possible1[i] - 1)) & player1) != 0) *//opponent* {  
 if (((mask\_possible >>> (indexes\_move2\_12\_possible2[i] - 1)) & ai) == 0) *//empty* {  
 AiPlayer.*index\_triple* = indexes\_move2\_12\_possible2[i];  
 AiPlayer.*mini\_board\_for\_twist\_triple* = mini\_board\_12\_possible2[i];  
 AiPlayer.*direction\_rotating\_triple* = twist\_board\_12\_possible2[i];  
 } else if (((mask\_possible >>> (indexes\_move2\_12\_possible2[i] - 1)) & ai) != 0) *//my ball* {  
 my\_color\_ball();  
 }  
  
 } else if (((mask\_possible >>> (indexes\_move2\_12\_possible1[i] - 1)) & ai) != 0) *//my ball* {  
 my\_color\_ball();  
 } else *//empty* {  
 if ((((mask\_possible >>> (indexes\_move2\_12\_possible2[i] - 1)) & player1) != 0)  
 || ((((mask\_possible >>> (indexes\_move2\_12\_possible2[i] - 1)) & ai) == 0))) *//opponent or empty* {  
 AiPlayer.*index\_triple* = indexes\_move2\_12\_possible1[i];  
 AiPlayer.*mini\_board\_for\_twist\_triple* = mini\_board\_12\_possible1[i];  
 AiPlayer.*direction\_rotating\_triple* = twist\_board\_12\_possible1[i];  
 } else *//white ball* {  
 my\_color\_ball();  
 }  
 }  
 }  
 }  
 mask = mask >>> 2;  
 }  
 } else {  
 long mask = 0b000000000010000000000000000000000000000000000000000000000000000L;  
 boolean check = false;  
 for (int i = 0; i < 4; i++) {  
 if (check == false) {  
 long mask\_possible = 0b100000000000000000000000000000000000000000000000000000000000000L;  
 if ((mask & ai) != 0) {  
 check = true;  
 if (((mask\_possible >>> (indexes\_move2\_34\_possible1[i] - 1)) & player1) != 0) *//opponent* {  
 if (((mask\_possible >>> (indexes\_move2\_34\_possible2[i] - 1)) & ai) == 0) *//empty* {  
 AiPlayer.*index\_triple* = indexes\_move2\_34\_possible2[i];  
 AiPlayer.*mini\_board\_for\_twist\_triple* = mini\_board\_34\_possible2[i];  
 AiPlayer.*direction\_rotating\_triple* = twist\_board\_34\_possible2[i];  
 } else if (((mask\_possible >>> (indexes\_move2\_34\_possible2[i] - 1)) & ai) != 0) *//my ball* {  
 my\_color\_ball();  
 }  
  
 } else if (((mask\_possible >>> (indexes\_move2\_34\_possible1[i] - 1)) & ai) != 0) *//my ball* {  
 my\_color\_ball();  
 } else *//empty* {  
 if ((((mask\_possible >>> (indexes\_move2\_34\_possible2[i] - 1)) & player1) != 0)  
 || ((((mask\_possible >>> (indexes\_move2\_34\_possible2[i] - 1)) & ai) == 0))) *//opponent or empty* {  
 AiPlayer.*index\_triple* = indexes\_move2\_34\_possible1[i];  
 AiPlayer.*mini\_board\_for\_twist\_triple* = mini\_board\_34\_possible1[i];  
 AiPlayer.*direction\_rotating\_triple* = twist\_board\_34\_possible1[i];  
 } else *//white ball* {  
 my\_color\_ball();  
 }  
 }  
 }  
 }  
 mask = mask >>> 2;  
 }  
 }  
  
 }  
 }  
  
 */\*\*  
 \* the strategy with all the steps  
 \* return- the grade for the strategy  
 \*/* public int triple\_power\_play() {  
 if (*player\_move* == 1) *//first ball to put on board* {  
 int playing\_strategy = random.nextInt(2) + 1; *// generates either 1 or 2, 1 for winning 12, and 2 for winning 34* current\_strategy = playing\_strategy;  
 player\_move\_1(playing\_strategy);  
 } else if (*player\_move* == 2) *//second ball to put on board* {  
 player\_move\_2();  
 } else if (*player\_move* == 3) { *//third ball to put on board* player\_move\_3();  
 } else if (*player\_move* > 3) {  
 player\_move\_above\_3(); *//above third ball to put on board* }  
 return grade;  
  
 }  
  
  
  
  
  
 */\*\*  
 \* repeating the checking after every strategy  
 \*  
 \* @param check- check if already finished  
 \* @param i- the i  
 \* @param keep\_ai- the ai board  
 \* @param keep\_player1- the player 1 board  
 \* @param x- the x  
 \* @param b- the boars  
 \* @param r- the right schema  
 \* @return  
 \*/* public boolean repeat(boolean check, int i, long keep\_ai, long keep\_player1, int x, Board b, int r) {  
 int[][][] indexes\_total = {indexes, indexes\_2, indexes\_3};  
 long mask = 0b100000000000000000000000000000000000000000000000000000000000000L;  
 for (int j = 0; j < 5; j++) {  
 if (check\_4\_finish == false & ((mask >>> (indexes\_total[r][i][j] - 1)) & keep\_ai) == 0 & ((mask >>> (indexes\_total[r][i][j] - 1)) & keep\_player1) == 0) { *//not taken  
 index\_rows\_columns* = indexes\_total[r][i][j];  
 if (x > 0) {  
 *index\_rows\_columns* = b.rotate\_whole\_opp(*index\_rows\_columns*, mini\_boards\_to\_twist\_whole[x - 1][0], mini\_boards\_to\_twist\_whole[x - 1][1]); *//the right index* }  
 *mini\_board\_for\_twist\_rows\_columns* = this.mini\_board[x][(i % 5)]; *//the mini board to twist* if (x == 2) {  
 *direction\_rotating\_rows\_columns* = this.twist\_mini\_boards\_34[(i % 5)]; *//the direction to rotate* } else *direction\_rotating\_rows\_columns* = this.twist\_mini\_boards\_12[(i % 5)]; *//the direction to rotate* check\_4\_finish = true; *//already found* if (r == 1)  
 grade\_rows\_columns = 40;  
 else grade\_rows\_columns = 60;  
 }  
 }  
 if (check == true)  
 return true;  
 return false;  
 }  
  
 */\*\*  
 \* find the rows\_and\_columns  
 \*  
 \* @return- the grade for the strategy  
 \*/* public int rows\_columns() {  
 check\_4\_finish = false; *//check if there is 4 in a row* check\_3\_finish = false; *//check if there is 3 in a row* check\_2\_finish = false; *//check if there is 2 in a row* check\_1\_finish = false; *//check if there is 1 in a row* check\_0\_finish = false; *//check if there is 0 in a row* int[][][] indexes\_total = {indexes, indexes\_2, indexes\_3};  
 long[][] masks\_total = {masks\_boards\_12, masks\_boards\_12\_2, masks\_boards\_12\_3};  
 for (int r = 0; r < 3; r++) {  
 for (int x = 0; x < 4; x++) {  
 long keep\_ai = ai;  
 long keep\_player1 = player1;  
 Board b = new Board();  
 b.setPlayer1(player1);  
 b.setPlayer2(ai);  
 if (x > 0) {  
 long[] players = b.rotate\_whole(mini\_boards\_to\_twist\_whole[x - 1][0], mini\_boards\_to\_twist\_whole[x - 1][1]);  
 keep\_ai = players[1];  
 keep\_player1 = players[0];  
 }  
 for (int i = 0; i < 10; i++) {  
 if (count\_1(Long.*toBinaryString*(keep\_ai & masks\_total[r][i])) == 4 & count\_1(Long.*toBinaryString*(keep\_player1 & masks\_total[r][i])) == 0) { *//not found* long mask = 0b100000000000000000000000000000000000000000000000000000000000000L;  
 for (int j = 0; j < 5; j++) {  
 if (check\_4\_finish == false & ((mask >>> (indexes\_total[r][i][j] - 1)) & keep\_ai) == 0) {  
 *index\_rows\_columns* = indexes\_total[r][i][j];  
 if (x > 0) {  
 *index\_rows\_columns* = b.rotate\_whole\_opp(*index\_rows\_columns*, mini\_boards\_to\_twist\_whole[x - 1][0], mini\_boards\_to\_twist\_whole[x - 1][1]);*//the right index* }  
 *mini\_board\_for\_twist\_rows\_columns* = this.mini\_board[x][(i % 5)]; *//the mini board to twist* if (x == 2) {  
 *direction\_rotating\_rows\_columns* = this.twist\_mini\_boards\_34[(i % 5)]; *//the direction to rotate* } else  
 *direction\_rotating\_rows\_columns* = this.twist\_mini\_boards\_12[(i % 5)]; *//the direction to rotate* check\_4\_finish = true;  
 return 120;  
 }  
 }  
 }  
 }  
 }  
 }  
  
  
 if (check\_4\_finish == false) {  
 for (int r = 0; r < 3; r++) {  
 for (int x = 0; x < 4; x++) {  
 long keep\_ai = ai;  
 long keep\_player1 = player1;  
 Board b = new Board();  
 b.setPlayer1(player1);  
 b.setPlayer2(ai);  
 if (x > 0) {  
 long[] players = b.rotate\_whole(mini\_boards\_to\_twist\_whole[x - 1][0], mini\_boards\_to\_twist\_whole[x - 1][1]);  
 keep\_ai = players[1];  
 keep\_player1 = players[0];  
 }  
 for (int i = 0; i < 10; i++) {  
 if (count\_1(Long.*toBinaryString*(keep\_ai & masks\_total[r][i])) == 3 & count\_1(Long.*toBinaryString*(keep\_player1 & masks\_total[r][i])) < 2) { *//less than 2 opponents* check\_3\_finish = repeat(check\_3\_finish, i, keep\_ai, keep\_player1, x, b, r);  
 }  
 }  
 }  
 }  
 }  
  
 if (check\_4\_finish == false & check\_3\_finish == false) {  
 for (int r = 0; r < 3; r++) {  
 for (int x = 0; x < 4; x++) {  
 long keep\_ai = ai;  
 long keep\_player1 = player1;  
 Board b = new Board();  
 b.setPlayer1(player1);  
 b.setPlayer2(ai);  
 if (x > 0) {  
 long[] players = b.rotate\_whole(mini\_boards\_to\_twist\_whole[x - 1][0], mini\_boards\_to\_twist\_whole[x - 1][1]);  
 keep\_ai = players[1];  
 keep\_player1 = players[0];  
 }  
 for (int i = 0; i < 10; i++) {  
 if (count\_1(Long.*toBinaryString*(keep\_ai & masks\_total[r][i])) == 2 & count\_1(Long.*toBinaryString*(keep\_player1 & masks\_total[r][i])) < 2) { *//less than 2 opponents* check\_2\_finish = repeat(check\_2\_finish, i, keep\_ai, keep\_player1, x, b, r);  
 }  
 }  
 }  
 }  
 }  
  
 if (check\_4\_finish == false & check\_3\_finish == false & check\_2\_finish == false) {  
 for (int r = 0; r < 3; r++) {  
 for (int x = 0; x < 4; x++) {  
 long keep\_ai = ai;  
 long keep\_player1 = player1;  
 Board b = new Board();  
 b.setPlayer1(player1);  
 b.setPlayer2(ai);  
 if (x > 0) {  
 long[] players = b.rotate\_whole(mini\_boards\_to\_twist\_whole[x - 1][0], mini\_boards\_to\_twist\_whole[x - 1][1]);  
 keep\_ai = players[1];  
 keep\_player1 = players[0];  
 }  
 for (int i = 0; i < 10; i++) {  
 if (count\_1(Long.*toBinaryString*(keep\_ai & masks\_total[r][i])) == 1 & count\_1(Long.*toBinaryString*(keep\_player1 & masks\_total[r][i])) < 2) { *//less than 2 opponents* check\_1\_finish = repeat(check\_1\_finish, i, keep\_ai, keep\_player1, x, b, r);  
 }  
 }  
 }  
 }  
 }  
  
  
 if (check\_4\_finish == false & check\_3\_finish == false & check\_2\_finish == false & check\_1\_finish == false) {  
 for (int r = 0; r < 3; r++) {  
 for (int x = 0; x < 4; x++) {  
 long keep\_ai = ai;  
 long keep\_player1 = player1;  
 Board b = new Board();  
 b.setPlayer1(player1);  
 b.setPlayer2(ai);  
 if (x > 0) {  
 long[] players = b.rotate\_whole(mini\_boards\_to\_twist\_whole[x - 1][0], mini\_boards\_to\_twist\_whole[x - 1][1]);  
 keep\_ai = players[1];  
 keep\_player1 = players[0];  
  
 }  
 for (int i = 0; i < 10; i++) {  
 if (count\_1(Long.*toBinaryString*(keep\_ai & masks\_total[r][i])) == 0 & count\_1(Long.*toBinaryString*(keep\_player1 & masks\_total[r][i])) < 2) { *//less than 2 opponents* long mask = 0b100000000000000000000000000000000000000000000000000000000000000L;  
 for (int j = 0; j < 5; j++) {  
 if (check\_0\_finish == false & ((mask >>> (indexes\_total[r][i][j] - 1)) & keep\_ai) == 0 & ((mask >>> (indexes\_total[r][i][j] - 1)) & keep\_player1) == 0) { *//not found  
 index\_rows\_columns* = indexes\_total[r][i][j];  
 *mini\_board\_for\_twist\_rows\_columns* = this.mini\_board\_12[(i % 5)]; *//the mini board to twist* if (x == 2) {  
 *direction\_rotating\_rows\_columns* = this.twist\_mini\_boards\_34[(i % 5)]; *//the direction to rotate* } else  
 *direction\_rotating\_rows\_columns* = this.twist\_mini\_boards\_12[(i % 5)]; *//the direction to rotate* check\_0\_finish = true;  
 if (r == 1)  
 grade\_rows\_columns = 40;  
 else grade\_rows\_columns = 60;  
 }  
 }  
 }  
 }  
 }  
 }  
 }  
 return grade\_rows\_columns;  
  
 }  
   
}