

מבוא

רקע על המשחק

אבלון (Abalone) הוא משחק אסטרטגיה עטור פרסים אשר הוצג לראשונה ב-1987.



המשחק מיועד לשני שחקנים שמתחרים זה נגד זה. המגרש בנוי מלוח משושה בעל 5 חורי משחק בכל צלע, מה שיוצר 61 חורי משחק אפשריים. כל שחקן מתחיל עם 14 כדורים כאשר מטרת המשחק היא "לאכול" ליריב 6 כדורים.

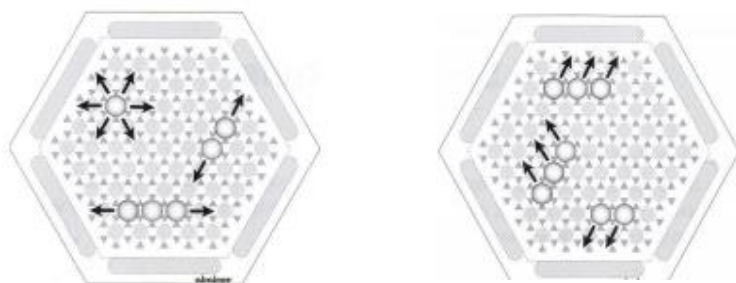
חוקי המשחק

בכל תור של שחקן מסוים, הוא יכול לבצע מהלך אחד. ישנם שני סוגי מהלכים: תזוזה ודחיפה.

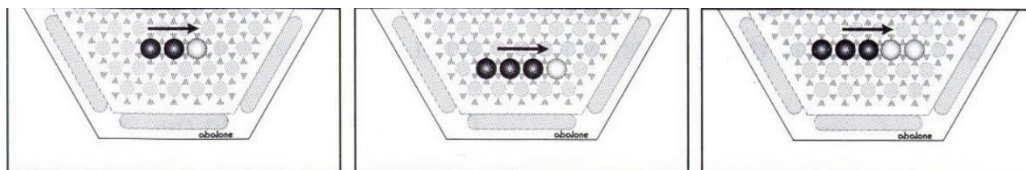
ניתן להזיז קבוצה של 1/2/3 כדורים בצבע של השחקן המשחק כאשר הכדורים מסודרים בשורה, לאחד מששת הכיוונים האפשריים בתנאי שאף אחד מהמקומות שאליהם מוזזים הכדורים אינו תפוס.

האפשרות השנייה היא להזיז את הכדורים בשורה, כך שניתן לדחוף כדורים של האויב (ובכך גם להשיג ניקוד) בהתאם לחוקים. שני כדורים של השחקן הדחוף יכולים לדחוף כדור אויב אחד בלבד (או לזוז מבלי לדחוף). שלושה כדורים של השחקן הדחוף יכולים לדחוף 0/1/2 שחקנים של האויב בלבד ובתנאי שאין כדור של השחקן הדחוף בצד הנדחף. (3 כדורים שלי לא יכולים לדחוף שורה שבה יש שלוש שלי, אחד אויב ואז עוד אחד שלי).

אפשרויות תזוזה:



אפשרויות דחיפה:



תיאור המערכת

המערכת שלי היא משחק אבולון אשר מובא למשתמש בגרסת הדפדפן.

ניתן להתחבר לשרת המשחק דרך אינטרנט מקומי (Lan) ולשחק בשתי גרסאות: שחקן נגד שחקן או שחקן נגד המחשב. הסבר להפעלת המערכת נמצא [במדריך למשתמש](#)

המשחק אבולון בנוי מלוח בעל 61 כדורים כאשר כל שחקן מתחיל עם 14 כדורים המסודרים במקביל ליריב. בכל תור השחקן המחשב יכול לבחור בין אחד לשלושה כדורים שלו וכיוון. במידה ובהלך בחירת המהלך השחקן בחר כדור שהמשך המהלך לא יהיה חוקי, המהלך מתבטל ובחירת המהלך מבוצעת מההתחלה.

במשחק קיימים שני אפשרויות- שחקן נגד שחקן או שחקן נגד המחשב. במידה ומשחקים בשחקן נגד שחקן כל שחקן בתורו יוכל לבחור רק את הכדורים שלו (ואז כיוון לתזוזה). לאחר בחירת מהלך חוקי, יעבור התור אל השחקן השני. במצב המשחק שחקן נגד המחשב לאחר בחירת המהלך של השחקן, האלגוריתם יבחר מהלך ואז יוצג ללקוח את המהלך שהאלגוריתם בחר ורק לאחר מכן התור יתממש במשחק. לאחר מכן התור יוחזר אל השחקן.

כאשר המשחק נגמר (אחד השחקנים הוציא 6 כדורים ליריב), בשני מצבי המשחק, מוצגת הודעה מתאימה והמשחק נגמר.

לפני כל משחק ניתן לבחור את המצב ההתחלתי של הלוח שלפיו יתחיל המשחק:

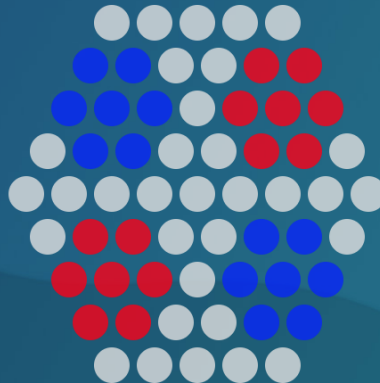


Welcome to Abalone!

•Classic •Pro •Snake •Wall

Start Game Player VS Player

Start Game Player VS Computer

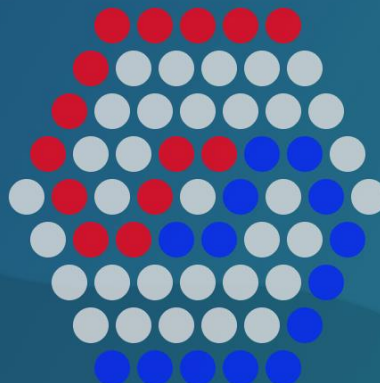


Welcome to Abalone!

•Classic •Pro •Snake •Wall

Start Game Player VS Player

Start Game Player VS Computer

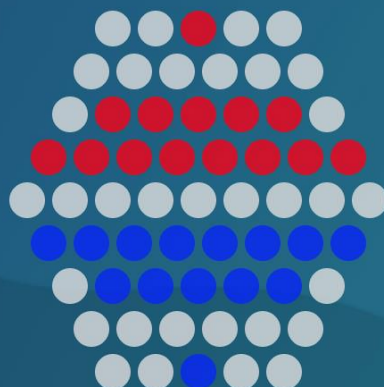


Welcome to Abalone!

•Classic •Pro •Snake •Wall

Start Game Player VS Player

Start Game Player VS Computer



סביבת עבודה

פיתוח המערכת נעשה בסביבת העבודה 12 – Eclipse 2019

שפת התכנות העיקרית היא java, כאשר נעשה שימוש ב html, jQuery ו- CSS בצד הלקוח (Web).

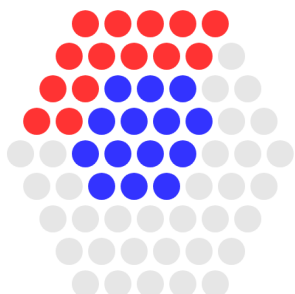
מחקר

לפני תחילת כתיבת הקוד, ביצעתי מחקר מקיף בנוגע למשחק והאסטרטגיות השונות בו.

מצאתי כי לכל מצב לוח אפשרי יש בממוצע כ 60 אפשרויות משחק שונות אך ראיתי גם שהמהלכים הריאליים שבאמת יתרמו לכל שחקן לרוב לא עולים על 10 מן האפשרויות הקיימות. עקב זאת, חקרתי כל אלגוריתם פתרון המחשב שיהיה כולל גזירה. ארחיב על כך [באלגוריתם המערכת](#)

במהלך המחקר מצאתי כי ישנם מספר עקרונות שחשוב לשמור עליהם בכדי לנצח. עקרון ראשון הוא שמירת כל הכדורים שלי בקבוצה אחת צפופה ככל הניתן. דבר זה מעלה את אופציות הדחיפה האפשריות (יותר שורות של 3 שחקנים שלי) ובכך שבירה של חומת היריב. העיקרון השני הוא שליטה על האמצע. שילוב של שני העקרונות יכול להעלות משמעותית את אחוזי הניצחון.

השאיפה של כל שחקן היא להגיע למצב בסגנון הבא, ובכך ניתן להבטיח בצורה כמעט וודאית את ניצחונו. (השחקן הכחול ינצח)



מפרטי תוכנה

ניסוח וניתוח הבעיות האלגוריתמיות.

בעיה אלגוריתמית: יצירת לוח יעיל

לוח המשחק בנוי מ-61 משבצות אשר מסודרות כמשושה, כך שלכל משבצת (מלבד המסגרת) ישנם 6 אפשרויות משחק (East, South East, South West, West, North West, North East) ולא 4 כמו בלוח מרובע רגיל. עקב כך, פתרון פשוט כמו מטריצה או מערך רגיל אשר כל מספר איברים קבוע מסמן שורה אינו פתרון ריאלי כאשר יש שוני במספר המשבצות בכל שורה בלוח.

על הפתרון להיות יעיל מאוד כיוון שבמהלך ריצת AI יהיו הרבה מאוד קריאת לפונקציות השונות שבודקות את הלוח.

בעיה אלגוריתמית: בדיקת תקינות מהלך (של שחקן)

השרת אמור לקבל מידע מן הלקוח בנוגע לאיזו משבצת הוא לחץ ובהתאם למיקום עליו לבדוק בלוח שלו האם המהלך חוקי. קיימים הרבה דברים שצריך לבדוק כמו האם הלחיצות נעשו אחת על יד השנייה, בשורה, לחיצה חוקית ולא לחיצה על היריב, ובדיקת תקינות מהלך (למשל שניים לא יכולים לדחוף שניים). דבר זה דורש לוגיקה ובדיקה יעילה.

בעיה אלגוריתמית: יצירת כל המהלכים האפשריים (של המערכת).

פעולת יצירת כל המהלכים האפשריים ללוח מסוים היא פעולה שתיקרא הרבה מאוד פעמים בכל תור של האלגוריתם. עקב כך, הפעולה צריכה להיות יעילה מאוד ומהירה מאוד ועל כך יש אתגר גדול ליצור את הפעולה שתהיה הכי יעילה.

בעיה אלגוריתמית: אלגוריתם המשחק של המחשב

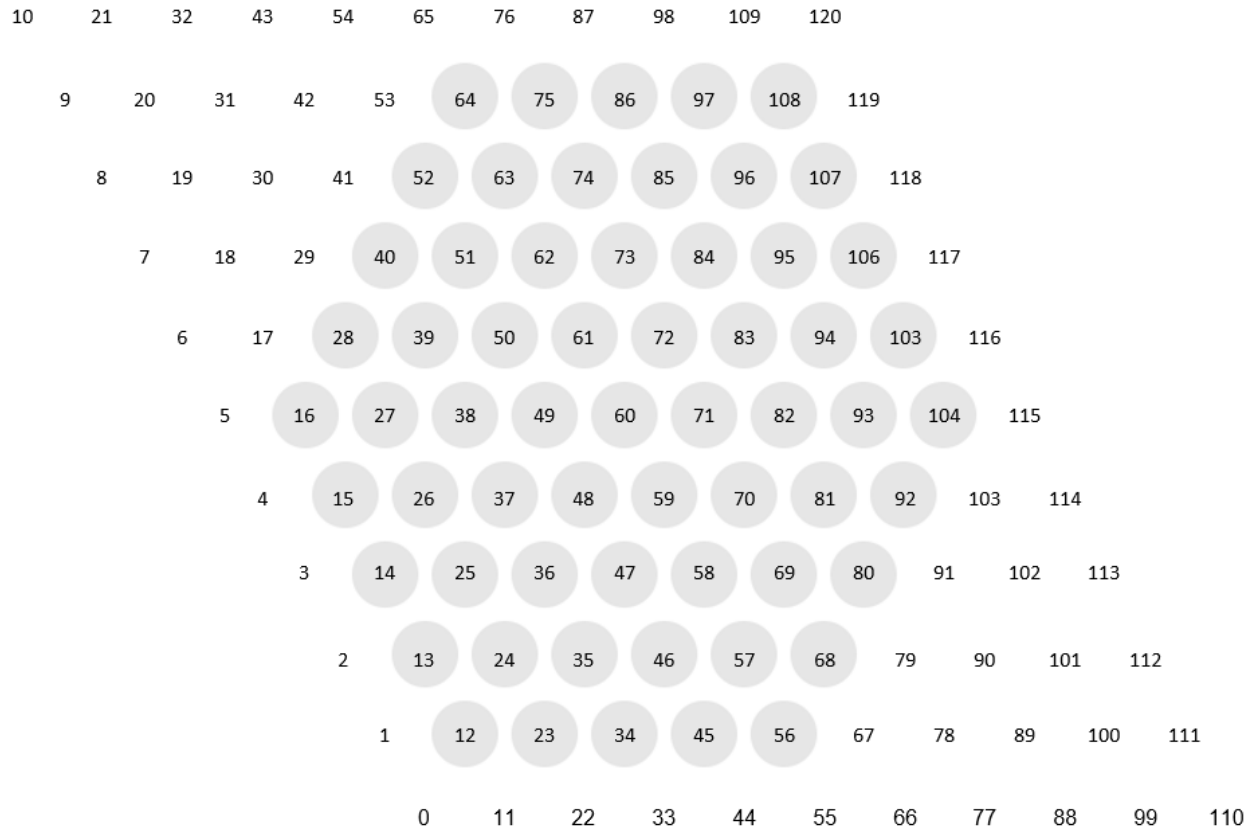
המשחק אבלון הוא משחק בעל מקדם גבוה מאוד - כ-60 מהלכים אפשריים לכל לוח בממוצע. עקב כך, פתרון פשוט כמו אלגוריתם MiniMax אינו ריאלי עבור משחק זה (מצורפת [טבלה](#) בפתרון). עלי ליצור פונקציית Evaluation טובה כך שאוכל להסתמך עליה יותר מאשר ירידה לעומק גדול יחסית שלא אפשרי במשחק זה מבחינת זמן הריצה.

פיתוח פתרונות ויישומם:

פיתוח פתרון יצירת לוח יעיל.

לאחר מחקר וניסיונות רבים, הפתרון שבחרתי הינו שימוש בשני BitSet שכל אחד מהם מייצג את מצב הכדורים של כל שחקן. בנוסף אני משתמש ב-BitSet קבוע כללי שפועל כקיר ביטחון. שלושת ה-BitSets הם בגודל 121 איברים (ביטים).

הלוח מיוצג כך:



החלוקה בין שלושת ה-BitSets היא:

- כל האינדקסים שבתוך הלוח (המשושה) דלוקים ב-BitSet הגבולות וכל מה שמחוץ ללוח לא דלוק.
- בהתאם למצב הלוח, ה-BitSet של כל שחקן מכון כך שהוא מחזיק (מסמן כדלוק) את כל המקומות שהכדורים שלו נמצאים.

בעזרת שקלול שלושת ה-BitSets ניתן לגלות בדיוק מהו מצב הלוח.

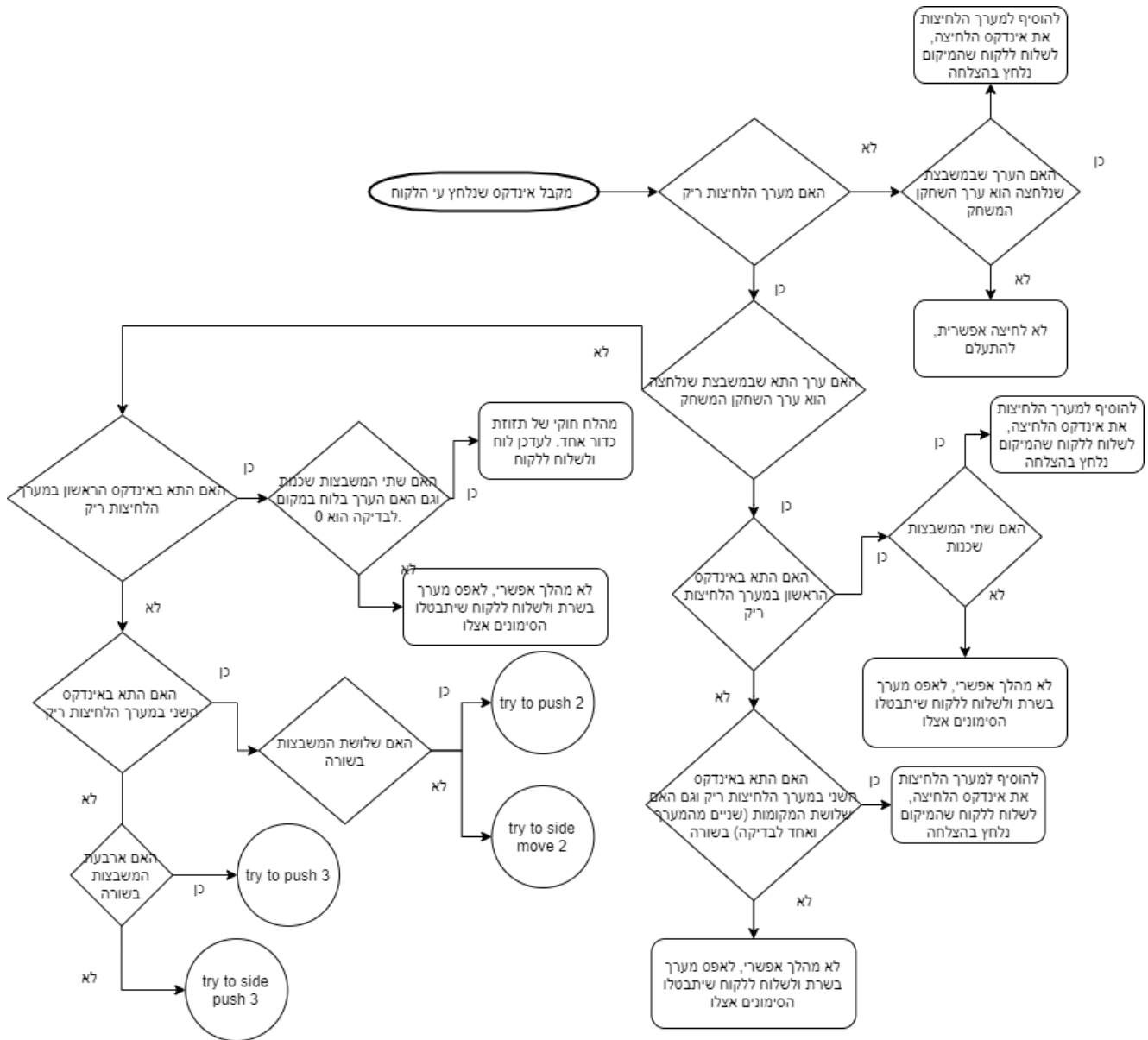
הערך שיש להוסיף	כיוון
+11	East
-1	South East
-12	South West
-11	West
+1	North West
+12	North East

פתרון זה של הלוח מאפשר הוספת ערך קבוע לכל תא בכדי לתת את השכנים שלו. הערכים של ההוספה מתוארים בטבלה:

בכדי למצוא את כל השכנים של מקום מסוים, יש לבדוק עבור כל אחד מן הערכים שבתבלה האם הערך המשוקלל נמצא (דלוק) בתוך ה-BitSet הגבולות.

כל הפעולות הקשורות ללוח: מציאת שכנים, מציאת ושינוי ערך של תא ובדיקת האם מיקום נמצא במגרש הם ביעילות $O(1)$.

פיתוח פתרון לבדיקת תקינות מהלך של שחקן:
 בכדי לבדוק תקינות מהלך של שחקן יש לבדוק מספר דברים בשביל לוודא שהמהלך עומד בכל הקריטריונים.
 המהלך מובא בתרשים הזרימה הבא:



סיבוכיות הריצה היא $O(1)$

הערות לתוכנית: במידה ומגיעים לאחד מן הפעולות try to push או try to sidemove ולא ניתן לבצע את המהלך, מהלך הלחיצות מתאפס, הסימונים אצל הלקוח מתאפסים גם והתוכנית מחכה לקליטה של מהלך חדש מאותו שחקן.

פיתוח פתרון ליצירת כל המהלכים האפשריים (של המערכת):
 כחלק מהאלגוריתם של המחשב, עלי ליצור פעולה שמקבלת לוח ומחזירה רשימה של כל המהלכים החוקיים האפשריים שניתן ליצור מלוח זה. כיוון שהפעולה נקראת הרבה מאוד פעמים, עליה להיות יעילה מאוד ומהירה מאוד.

סיבוכיות הריצה היא $O(N)$ כאשר N הוא כמות הכדורים של השחקן שעבורו האלגוריתם פועל.

מצורף פסודו קוד של האלגוריתם:

```
public ArrayList<Move> getmoves(byte currentPlayer)
```

1. צור רשימה חדשה בגודל 60.
2. קבע מהו ה BitSet שעבורו מייצרים את כל המהלכים האפשריים.
3. עבור כל איבר ב BitSet (כל כדור), נסמן כ posA, בצע:
 - 3.1 צור מערך שמכיל את כל האינדקסים של השכנים של posA
 - 3.2 עבור כל איבר במערך, נסמן כ posB, בצע:
 - 3.2.1 אם ערך המשבצת ב posB היא 0:
 - 3.2.1.1 צור מהלך חדש של תזוזת כתור אחד ותוסיף לרשימה את המהלך.
 - 3.2.2 אחרת:
 - 3.2.2.1 אם הערך במשבצת posB הוא של השחקן האויב: קפוץ ללולאה הבאה.
 - 3.2.2.2 posC --> המיקום הבא בשורה של posA - posB
 - 3.2.2.3 אם posC הוא מיקום חוקי, בצע:
 - 3.2.2.3.1 אם הערך ב posC הוא של השחקן המשחק, בצע:
 - 3.2.2.3.1.1 posD --> המיקום הבא בשורה של posB - posC
 - 3.2.2.3.1.2 אם posD הוא מיקום חוקי: קרא לפעולה TryToPush3(posA, posB, posD, posC, posD) ותוסיף לרשימה במידה והמהלך חוקי.
 - 3.2.2.3.2 אחרת:
 - 3.2.2.3.2.1 קרא לפעולה TryToPush2(posA, posB, posC) ותוסיף לרשימה במידה והמהלך חוקי.
 - 3.2.2.4 עבור כל שכן של posB, נסמן כ posSideB, בצע:
 - 3.2.2.4.1 אם posSideB אינו מיקום חוקי: המשך ללולאה הבאה.
 - 3.2.2.4.2 אם posSideB הוא posA או posB או הערך שבמשבצת posSideB אינו 0: המשך ללולאה הבאה.
 - 3.2.2.4.3 posSideA --> מצא את המיקום הצמוד ל posSideB, posA, posB.
 - 3.2.2.4.4 אם posSideA אינו מיקום חוקי או הערך שבמשבצת אינו 0: המשך ללולאה הבאה.
 - 3.2.2.4.5 צור מהלך של תזוזת שני כדורים הצידה (posA, posB, posSideA, posSideB) ותוסיף לרשימה.
 - 3.2.2.4.6 posC --> המיקום הבא בשורה של posA, posB.
 - 3.2.2.4.7 אם posC אינו מיקום חוקי או הערך שבמשבצת אינו 0: המשך ללולאה הבאה.
 - 3.2.2.4.8 posSideC --> המיקום הבא בשורה של posSideA, posSideB.
 - 3.2.2.4.9 אם posSideC אינו מיקום חוקי או הערך שבמשבצת אינו 0: המשך ללולאה הבאה.
 - 3.2.2.4.10 צור מהלך חדש של תזוזת 3 כדורים הצידה (posA, posB, posC, posSideA, posSideB, posSideC) ותוסיף את המהלך לרשימה.

פיתוח פתרון לאלגוריתם המשחק של המחשב

כיוון שאבולון הוא משחק עם מקדם גבוה מאוד (כ-60 אפשרויות משחק שונות בכל תור), פתרון פשוט כמו MiniMax אינו ריאלי למשחק זה.

הפתרון שבחרתי בו הוא שימוש בפונקציית הערכת לוח Evaluation function שמקבלת לוח ונותנת לו ערך מספרי מ 1000000 עד -1000000. כך שכאשר ערך הלוח הוא 1000000 השחקן הכחול ניצח, וכאשר -1000000 השחקן האדום מנצח.

לאחר המחקר שביצעתי, מצאתי מספר עקרונות שאותם מימשתי בפונקציית ה-Evaluation.

לתוך הפונקציה הוספתי חישוב למרחק הכולל של הכדורים מן האמצע, והוספתי גם חישוב לאופן בו הכדורים נמצאים בקבוצה. לאחר החישובים חיסרתי בין השחקן הכחול לאדום ובכך נוצר ערך המשקף את מצב הלוח באופן יחסי לאויב.

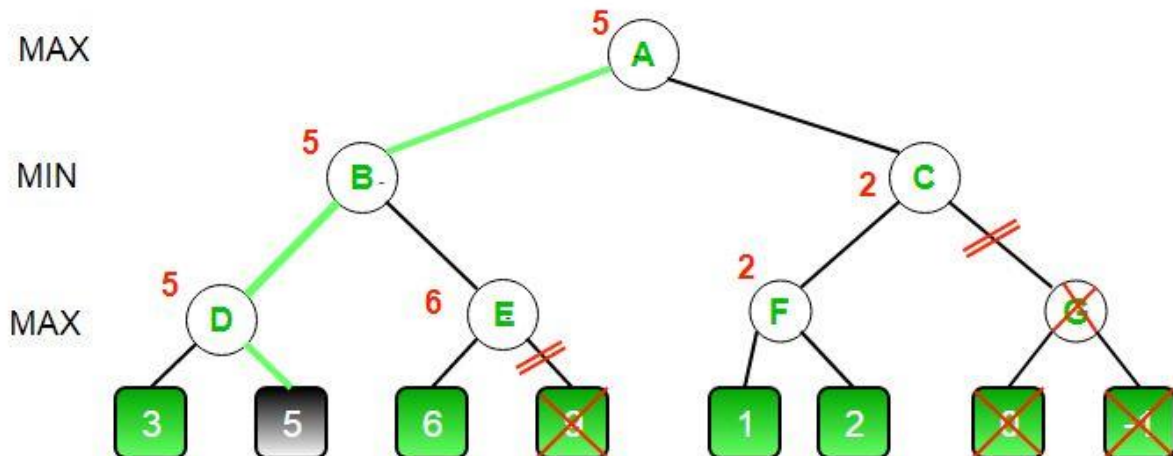
כיוון שפונקציית ה-Evaluation לא 'רואה' מהלכים קדימה אלא רק מדרגת לוח אחד בכל פעם, הייתי צריך להכניס אותה לתוך פונקציית עץ רקורסיבית.

כיוון שמקדם המשחק של אבולון גבוה מאוד, בחרתי באלגוריתם ה-Alpha Beta. אלגוריתם זה מחזיר את אותו ערך סופי כמו MiniMax (שבודק את כל המהלכים האפשריים בעומק מוגדר מראש ומחשב את ערכם) אלא ש-Alpha Beta יודע לחתוך עצי משחק שאין טעם בהמשך בדיקתם.

באלגוריתם ה-Alpha Beta מועברים ערכי הלוח לבדיקה, איזה שחקן אמור לשחק את המהלך ו-Alpha ו-Beta שמסמלים את הערך הכי גבוה (יחסי לשחקן) שנמצאו עד עכשיו שניתן להבטיח את קיומם.

דוגמא לעץ משחק בעומק 3 שלכל צומת ישנם שני בנים (באבולון יש כ-60). בציור מוצג כיצד האלגוריתם יודע לזהות מצבים שאין טעם בהמשך בדיקתם ולכן "חותך" אותם ומפסיק בבדיקה בחלק זה.

ה-Max ו-Min מראה את מה השחקן המשחק מנסה להשיג. השחקן הראשון שב-Max ינסה למקסם את האפשרויות שלו והשחקן שמחשק ב-Min יבחר באפשרות הנמוכה שאותה יכול להבטיח.



Public int AlphaBeta(Board Board, int Depth, int alpha, int beta, CurrentPlayer)

1. אם $Depth \leq 1$ או $board.getScoreBlue() \geq 6$ או $board.getScoreRed() \geq 6$: חשב ותחזיר את הערך הכללי של הלוח.
2. אם $CurrentPlayer == 1$ בצע:
 - 2.1. $Value = -2000000$
 - 2.2. חשב את כל המהלכים האפשריים של השחקן $CurrentPlayer$ והכנס לרשימה.
 - 2.3. חשב את כל הלוחות האפשריים לאחר מימוש המהלכים בהתאמה מהרשימה.
 - 2.4. עבור כל בן של הלוח $board$, נסמן כ $childBoard$, ובצע:
 - 2.4.1. $Value \rightarrow$ הערך המקסימלי מבין $Value$ ו- $AlphaBeta(childBoard, Depth-1, alpha, beta, - CurrentPlayer)$
 - 2.4.2. $Alpha \rightarrow$ הערך הגבוה מבין $Alpha$ ו- $Value$.
 - 2.4.3. אם $Alpha \geq Beta$: צא מהלולאה.
 - 2.5. החזר את $Value$.
3. אחרת:
 - 3.1. $Value = 2000000$
 - 3.2. חשב את כל המהלכים האפשריים של השחקן $CurrentPlayer$ והכנס לרשימה.
 - 3.3. חשב את כל הלוחות האפשריים לאחר מימוש המהלכים בהתאמה מהרשימה.
 - 3.4. עבור כל בן של הלוח $board$, נסמן כ $childBoard$, ובצע:
 - 3.4.1. $Value \rightarrow$ הערך המינימלי מבין $Value$ ו- $AlphaBeta(childBoard, Depth-1, alpha, beta, - CurrentPlayer)$
 - 3.4.2. $Beta \rightarrow$ הערך הנמוך מבין $Alpha$ ו- $Beta$.
 - 3.4.3. אם $Alpha \geq Beta$: צא מהלולאה.
 - 3.5. החזר את $Value$.

טבלה המתארת את ריצת התוכנית הבדיקה היא מתחילת מצב classic עם 50 מהלכים אפשריים:

זמן ריצה	כמות עלים (מצבי לוח לחישוב)	עומק
12 דקות	702338951	5 בלי חתיכה
12 שניות	11034067	4 בלי חתיכה
~ 0.1 שניות	170831	3 בלי חתיכה
1.5 שניות	382798	5 עם חתיכה
< 0.3 שניות	194324	4 עם חתיכה
~ 0.1 שניות	5850	3 עם חתיכה

פונקציית ה-Evaluation היא למעשה לב ליבו של האלגוריתם.

כאמור, לפונקציה הכנסתי את המרחק של הכדורים מן האמצע, את הצורה שבה הכדורים נמצאים בקבוצה ואת הניקוד הנוכחי. לשלושת הערכים הללו נתתי ערך מספרי ושיקללתי ביניהם בעזרת משקלים קבועים בתוכנית ששינתי בהתאם להתנהגות האלגוריתם.

הפונקציה קוראת ל-3 פונקציות שונות אשר אפרט עליהם בהמשך.

התוכנית תמיד מחזירה ערך כך שכל שהערך יותר גבוה כך מצב הלוח עבור השחקן הכחול יותר טוב.

סיבוכיות הריצה היא $O(N)$ כאשר N הוא כמות הכדורים של השחקן שעבורו האלגוריתם פועל.

פסודוקוד התוכנית EvaluateBoard :

```
public int EvaluateBoard(Board board)
```

1. צור משתנה חדש `sum = 0`.
2. `sum += CalculateBoardDistances(board, (byte) 1)`.
3. `sum -= CalculateBoardDistances(board, (byte) -1)`.
4. `sum += CalculateGrouping(board, (byte) 1)`.
5. `sum -= CalculateGrouping(board, (byte) -1)`.
6. `sum += CalculateScore(board, (byte) 1)`.
7. `sum -= CalculateScore(board, (byte) -1)`.
8. החזר את `sum`.

פונקציית `CalculateBoardDistances` מקבלת מצב לוח ושחקן שעבורו תחזיר ערך עבור המרחק הכולל של הכדורים ממרכז הלוח.

כיוון שהלוח קבוע, בחרתי בפתרון שכולל חמישה bitsets שמתפקדים כ-mask. כל bitset מייצג טבעת כך שכל משבצת בלוח נמצאת באחת הטבעות. כל bitset הוא בגודל 109 (חופף לאינדקסי ה-bitsets של הלוח עד לאינדקס הכי גבוה שמייצג משבצת) ודלוקים בו אינדקסי המשבצות בהתאם לטבעת.

עבור כל כדור של השחקן, האלגוריתם מחפש באיזו טבעת הוא נמצא- ובהתאם מוסיף ערך לסכום. לבסוף כופלים ומחזירים את הסכום כפול משקל קבוע כפול -1. הפעולה מחזירה מספר שלילי בכדי שהסכום יהיה יותר גבוה ככל שהכדורים קרובים למרכז ולא ההפך.

כאמור, הלוח מחולק ל-5 טבעות.

הערך שמוסף לכדור שבטבעת החיצונית הוא 5.

הערך שמוסף לכדור שבטבעת ה-4 הוא 4.

הערך שמוסף לכדור שבטבעת ה-3 הוא 2.8.

הערך שמוסף לכדור שבטבעת ה-2 הוא 1.

הערך שמוסף לכדור שבטבעת ה-1 (רק המשבצת האמצעית) הוא 0.

סיבוכיות הריצה היא $O(N)$ כאשר N הוא כמות הכדורים של השחקן שעבורו האלגוריתם פועל.

פסודוקוד:

```
private int CalculateBoardDistances(Board board, byte side)
```

1. קבע `bitset a` כ-`bitset` שמייצג את השחקן `side`.
2. צור משתנה חדש `sum = 0`.
3. עבור כל ביט שדלוק ב `a`, סמן ב `i` את האינדקס ובצע:
 - 3.1. אם `i` נמצא בטבעת הראשונה בצע:
 - 3.1.1. המשך ללולאה הבאה.
 - 3.2. אם `i` נמצא בטבעת השנייה בצע:
 - 3.2.1. `Sum+=1`
 - 3.2.2. המשך ללולאה הבאה.
 - 3.3. אם `i` נמצא בטבעת השלישית בצע:
 - 3.3.1. `Sum+=2.8`
 - 3.3.2. המשך ללולאה הבאה.
 - 3.4. אם `i` נמצא בטבעת הרביעית בצע:
 - 3.4.1. `Sum+=4`
 - 3.4.2. המשך ללולאה הבאה.
 - 3.5. `Sum+=5`
4. החזר `sum*ValueOfCenterDistance*-1`.

פונקציית `CalculateGrouping` מקבלת מצב לוח ושחקן שעבורו תחזיר ערך עבור כמות ההתקבצות של כל כדורי השחקן.

החישוב מבוצע כך שקיים משתנה סופר `counter` שעבור כל כדור של השחקן, מתווסף ל-`counter` 1 עבור כל שכן שיש לו. לאחר מכן המשתנה מוכפל במשקל קבוע (7) והתוצאה מוחזרת.

סיבוכיות הריצה היא $O(N)$ כאשר N הוא כמות הכדורים של השחקן שעבורו האלגוריתם פועל.

פסודוקוד:

```
private int CalculateGrouping(Board board, byte side)
```

1. קבע `bitset a` כ-`bitset` שמייצג את השחקן `side`.
2. צור משתנה חדש `cnt = 0`.
3. עבור כל ביט שדלוק ב `a`, סמן ב `i` את האינדקס ובצע:
 - 3.1. עבור כל ששת השכנים של מיקום מסוים, סמן את האינדקס שלהם כ-`pos` ובצע:
 - 3.1.1. אם `pos` לא נמצא בתחומי המגרש: המשך ללולאה הבאה.
 - 3.1.2. אם הביט שהאינדקס שלו הוא `pos` ב-`bitset a` דלוק:
 - 3.1.2.1. `cnt++`
 - 3.2. החזר את `ValueOfGrouping*cnt`.

פונקציית CalculateScore מקבלת מצב לוח ושחקן שעברו תחזיר ערך עבור מצב הניקוד במצב זה.

בתנאי שאין ניצחון, עבור כל כדור של האויב שנפסל, הערך המוחזר הוא משקל קבוע (1000) * כמות הכדורים שנפסלו. כאשר זוהי מצב ניצחון עבור השחקן, הערך המוחזר הוא 1000000. (גבוה במטרה שבמידה והאלגוריתם יוכל להבטיח את הניצחון, הוא אכן ינצח ולא יישאר במצב הגנה).

סיבוכיות הריצה היא $O(1)$

פסודוקוד:

```
private int CalculateScore(Board b, byte side)
```

1. אם $side == 1$ בצע:

1.1. אם $b.getScoreBlue() \geq 6$ החזר 1000000.

1.2. החזר $b.getScoreBlue() * ValueOfScoredBalls$.

2. אם $b.getScoreRed() \geq 6$ החזר 1000000.

3. החזר $b.getScoreRed() * ValueOfScoredBalls$.

ניתן היה להוסיף ערכים לפונקציית ה-EvaluateBoard כמו חישוב כמות אפשרויות המשחק של כל שחקן וחישוב כמות ההתקפות האפשריות של כל שחקן.

לבסוף בחרתי שלא להוסיף אותם לפונקציה כיוון ששני ערכים אלו נגזרים מערכי ההתקבצות והמרחק מהאמצע כך ששחקן שישלוט במרכז המגרש ויהיה בקבוצה אחת גדולה – יהיו לו משמעותית יותר אפשרויות משחק ואפשרויות התקפה מהאפשרויות של היריב.

מבני נתונים

BitSet

BitSet הוא מבנה נתונים שמממש מערך של תאים מסוג Boolean (ביט אחד), כך שכל תא יכול להחזיק אחד משני הערכים True או False.

הפונקציות שאני משתמש במבנה הנתונים הם:

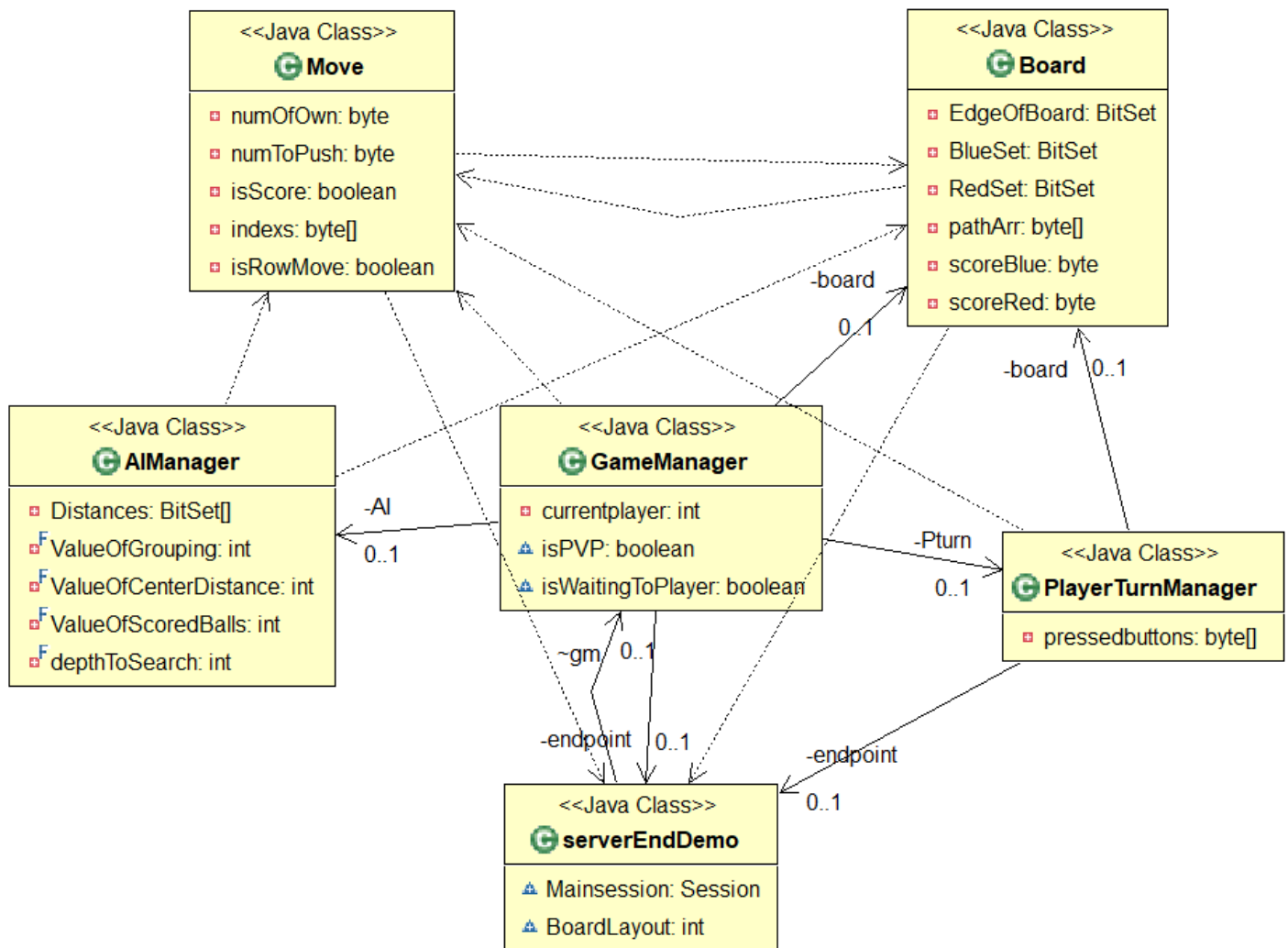
<u>BitSet</u> (int nbits)	יצירת BitSet, מקבל מספר שקובע את כמות האיברים.
<u>Get</u> (int bitindex)	מחזיר את הערך הבוליאני של התא במקום הנתון.
<u>Set</u> (int bitindex, boolean value)	משנה את הערך הבוליאני של התא במקום הנתון, בהתאם לערך הנתון.
<u>Set</u> (int fromindex, int toindex, boolean value)	משנה את הערכים בתאים, מהתא fromindex אל התא (לא כולל) toindex, בהתאם לערך הנתון.
<u>NextSetBit</u> (int bitindex)	מחזיר את האינדקס של המיקום הבא שדלוק אחרי הערך הנתון. מחזיר -1 אם לא מצא.
<u>Clone</u> ()	הפעולה מחזירה BitSet חדש הזהה ל-BitSet שעליו הפונקציה נקראה.

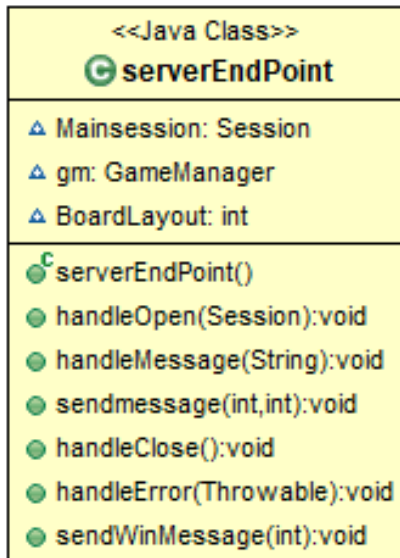
ArrayList

ArrayList הוא מבנה נתונים מסוג רשימה, שמממש תאים מסוג מסוים שאותו ניתן להגדיר. אני משתמש ב-ArrayList בכדי ליצור את הרשימות של כל המהלכים וכל הלוחות האפשריים.

הפונרציות שאני משתמש במבנה הנתונים הם:

<u>ArrayList</u> <E>(int size)	יצירת ה-ArrayList, בגודל size שהתאים שלו מסוג E
<u>add</u> (<u>E</u> e)	מוסיף לרשימה באיבר הריק הבא את האיבר e
<u>forEach</u> (<u>Consumer</u> <? super <u>E</u> > action)	יצירת לולאת ForEach שעבור הרשימה action מוציא בכל ריצה של הלולאה את האיבר הבא ברשימה Consumer.

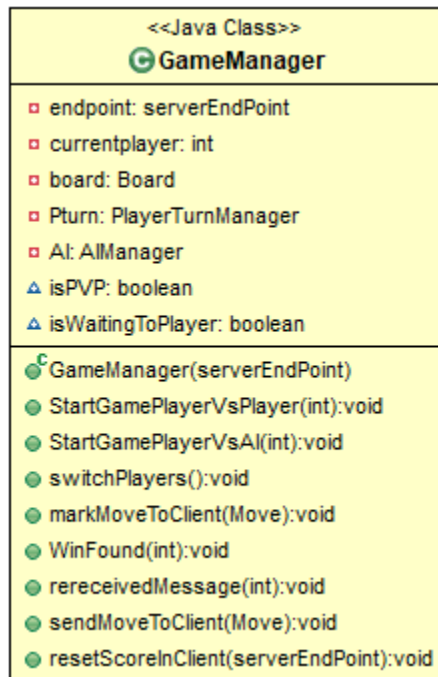




המחלקה היא הקשר בין השרת לבין הלקוח.

השרת מטפל בהקמת הקשר בין הלקוח לשרת, מנהל את קבלת כל ההודעות והשליחה שלהם אל הגורמים הנכונים. המחלקה מטפלת בשליחת כל ההודעות כולל הודעת ניצחון/שינוי ערך במשבצת עבור הלקוח.
































Session Mainsession;	The client.
GameManager gm;	The GameManger.
int BoardLayout = 0;	The board layout index. Default 0 (classic layout).
public void handleOpen(Session ses)	The program handles the start of the connection between server and client. The program gets session ses which is the mainsession of the server. @param ses – the session of client
public void handleMessage(Session ses)	The program handles all the messages which the server receives. The program gets message which can be a simple string or jsonObject. @param message - the message from client
public void sendmessage(int indexx, int val)	the program updates at the client one position. send to client the position and the new value to put inside. @param indexx - the index of position. @param val - the value of position.
public void handleClose()	program prints that sever has disconnected.
public void handleError(Throwable t)	the program prints that error have occurred @param t - the error
public void sendWinMessage(int value)	the program sends the client the player who won the game. 1 for blue, -1 for red. @param value - player number.



המחלקה אחראית על ניהול המשחק. המחלקה אחראית על פתיחת המשחק בהתאם למצב הנלחץ (שחקן נגד שחקן או שחקן נגד המחשב). המחלקה אחראית על בדיקת ניצחון, החלפת השחקנים המשחקים או הפעלת האלגוריתם.

private serverEndPoint endpoint;	Server endpoint to move messages to client.
private int currentPlayer;	Current player.
private Board board;	The main board.
private PlayerTurnManager Pturn;	The Player Turn Manager (receive move from client).
private AIManager AI;	The AI Manger(receive move from computer algorithm).
boolean isPVP;	Game Mode.
boolean isWaitingToPlayer = false;	To make client wait between the start and end of ai move.
public GameManager(serverEndPoint point)	Constructor. @param point - server end point.
public void StartGamePlayerVsPlayer(int BoardLayout)	the program get called once the client pressed the start game button. the program initializes the board and sends messages of the board to the client. @param BoardLayout - the game layout.
public void StartGamePlayerVsAI(int BoardLayout)	program gets called when client presses "start game player vs AI" program initialize required parts in code. * @param BoardLayout - the game layout.
public void switchPlayers()	program switches between players/AI. If the game mode is pvAI than its activates the ai.
public void markMoveToClient(Move aIMove)	program gets move (AI move) and mark move in client. includes time.wait so client can see the move. @param aIMove – AI move

public void WinFound(int value)	program gets called once a win was found. program sends the data to the endpoint to declare a winner to the client. @param value - player number
public void rereceivedMessage(int index)	program receives a press index of client, sends the index to the turn manager which returns a move. if it is not end of turn than move = null. @param index - client press index.
public void sendMoveToClient(Move move)	program gets move and sends client the updated positions of board. @param move - move to implement at client
public void resetScoreInClient(serverEndPoint point)	the program resets the value inside the score positions in client. from -10 to -16 and from -20 to -26 @param point - the server end point

<<Java Class>>	
 Board	
<ul style="list-style-type: none"> EdgeOfBoard: BitSet BlueSet: BitSet RedSet: BitSet pathArr: byte[] scoreBlue: byte scoreRed: byte 	
<ul style="list-style-type: none">  Board()  getEdgeOfBoard():BitSet  setEdgeOfBoard(BitSet):void  getBlueSet():BitSet  setBlueSet(BitSet):void  getRedSet():BitSet  setRedSet(BitSet):void  getScoreBlue():byte  setScoreBlue(byte):void  getScoreRed():byte  setScoreRed(byte):void  initpathArr():void  initializeBoard(int):void  sendIntireBoardToClient(serverEndPoint):void  GetValueInPosition(byte):byte  SetValueInPosition(byte,byte):void  isPositionInBoard(byte):boolean  IsPositionsTogether(byte,byte):boolean  getNextPositionInLine(byte,byte):byte  get4thPosInSideMove(byte,byte,byte):byte  makeMove(Move):int  isPositionsInLine3(byte,byte,byte):boolean  switchPositions(byte,byte):void  TryToPush2(byte,byte,byte):Move  TryToPush3(byte,byte,byte,byte):Move  TryToSideMove2(byte,byte,byte):Move  TryToSideMove3(byte,byte,byte,byte):Move  getmoves(byte):ArrayList<Move>  getNeighborsOfPossition(byte):byte[]  cloneBoard(Board):void 	

המחלקה אחראית על כל הפעולות על לוח המשחק עצמו. במחלקה יש פעולות שבעזרתם ניתן לקבוע תקינות מהלך, מציאת שכנים של מיקום מסוים ומימוש ייצוג הלוח עי שלושת הbitsets.

בנוסף בפונקציה יש פעולה שיוצרת רשימה של כל המהלכים האפשריים שניתן ליצור מהלוח.

private BitSet EdgeOfBoard ;	Edge of board data structure.
private BitSet BlueSet ;	Blues balls data structure.
private BitSet RedSet ;	Reds balls data structure.
private byte pathArr [];	6 directions vectors.
private byte scoreBlue ;	Current blue score.
private byte scoreRed ;	Current red score.
public Board()	Constructor. Initializes pathArr.
public BitSet getEdgeOfBoard ()	Return EdgeOfBoard bitset.
public void setEdgeOfBoard (BitSet edgeOfBoard)	sets EdgeOfBoard bitset.
public BitSet getBlueSet ()	Return BlueSet bitset.
public void setBlueSet (BitSet blueSet)	sets BlueSet bitset.
public BitSet getRedSet ()	Return RedSet bitset.
public void setRedSet (BitSet redSet)	sets RedSet bitset.
public byte getScoreBlue ()	Return ScoreBlue.
public void setScoreBlue (byte scoreBlue)	sets ScoreBlue.
public byte getScoreRed ()	Return ScoreRed.
public void setScoreRed (byte scoreRed)	sets ScoreRed.
public void initpathArr ()	The program initializes the initpathArr
public void initializeBoard (int BoardLayout)	program creates new 3 bitsets - blueSet, RedSet, EdgeOfBoard. @param BoardLayout - sets the start game layout according to the BoardLayout
public void sendIntireBoardToClient (serverEndPoint point)	program sends to server end point all of the positions in the current board. @param point - sends messages to.
public byte GetValueInPosition (byte position)	program return the value of position in board. @param position - position to check value for. @return - -9 if not in board, 1 if in blueset, -1 if in redset , 0 if not pressed (not in both).
public void SetValueInPosition (byte position , byte value)	program sets value in given position @param position - position to change. @param value - the value to change to.
public boolean isPositionInBoard (byte position)	program gets position and return true if in board. @param position - position to check @return - true if in board, false if not.
public boolean IsPositionsTogether (byte posA , byte posB)	program check if teo positions are next to each other. !! does not check if 2 positions are in board. @param posA - first position @param posB - second position @return true if both positions are next to each other.
public byte getNextPositionInLine (byte posA , byte posB)	program gets 2 position and return the next position in line. @param posA - first position @param posB - second position @return posC if position found, -9 if position is not in board.

public byte get4thPosInSideMove(byte posA, byte posB, byte posSideB)	program returns the position of posSideA @param posA - first position @param posB - second position, next to posA @param posSideB - second side position, in side to posB @return -9 if posSideA is not in board, else returns posA.
public int makeMove(Move move)	program gets a valid move and updates current board from the move. @param move - valid move to implement. @return 1 if Winfound, 0 of not.
public boolean isPositionsInLine3(byte posA, byte posB, byte posC)	program gets two points: A and B are next to each other. the programs return if posC is in line after B. @param posA - first position @param posB - second position (next to posA) @param posC - third position @return true if posA, posB and posC are in a line.
public void switchPositions(byte posA, byte posB)	the program switches the valueS of two given positions. @param posA - first position @param posB - second position
public Move TryToPush2(byte posA, byte posB, byte posC)	program tries to create a move of push 2 balls in the direction of posC. @param posA - first position (own) @param posB - second position(next to posA , own) @param posC - third position (in line) can be (valueof posA)*-1 or 0 @return move if found a valid move, null if not.
public Move TryToPush3(byte posA, byte posB, byte posC, byte posD)	program tries to create a move of push 3 balls in the direction of posD. @param posA - first position (own) @param posB - second position(next to posA , own) @param posC - second position(in line , own) @param posD - third position (in line) can be (valueof posA)*-1 or 0 @return move if found a valid move, null if not.
public Move TryToSideMove2(byte posA, byte posB, byte posSideB)	program check if can do a side move of posA and posB to posSideA and posSideB @param posA - first position @param posB - second position (next to posA , own) @param posSideB - first sids position , empty. @return move if found a valid move, null if not.
public Move TryToSideMove3(byte posA, byte posB, byte posC, byte posSideC)	program check if can do a side move of posA, posB and posC to posSideA and posSideB and posSideC @param posA - first position @param posB - second position (next to posA , own) @param posB - second position (in line , own) @param posSideC - first side position , empty. @return move if found a valid move, null if not.

public ArrayList<Move> getmoves(byte currentPlayer)	program finds all the possible moves of Currentplayer from current board. @param currentPlayer - player to search moves for. @return list of all possible moves of current player.
public byte [] getNeighborsOfPossition(byte position)	program creates a bute array of all positions next to a given position. if a positions is not in board, its -1 in the array. @param position - position to creates neighbors array @return byte array of neighbors , -1 for each one is out of board.
public void cloneBoard(Board b)	program gets a board instance and sets current (this) board the values of the given board. @param b - board to clone

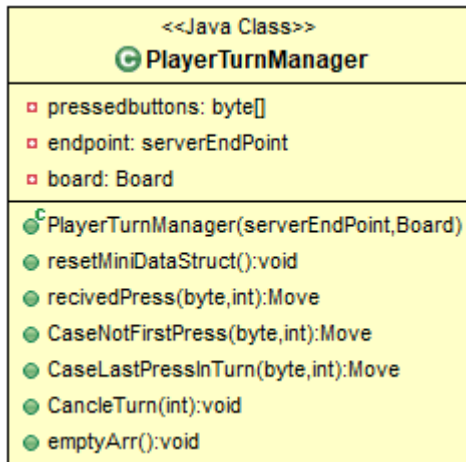
<<Java Class>>	
Move	
<ul style="list-style-type: none"> numOfOwn: byte numToPush: byte isScore: boolean indexs: byte[] isRowMove: boolean 	
<ul style="list-style-type: none"> Move() getIndexs():byte[] setIndexs(byte[]):void isRowMove():boolean setRowMove(boolean):void setNumOfOwn(byte):void setNumToPush(byte):void setScore(boolean):void getNumOfOwn():byte getNumToPush():byte isScore():boolean sendMoveToClient(serverEndPoint,Board):void new1BallMove(byte,byte):void new2BallsSideMove(byte,byte,byte,byte):void new3BallsSideMove(byte,byte,byte,byte,byte,byte):void new2BallsMove0Push(byte,byte,byte):void new2BallsMove1PushNoScore(byte,byte,byte,byte):void new2BallsMove1PushWithScore(byte,byte,byte):void new3BallsMove0Push(byte,byte,byte,byte):void new3BallsMove1PushNoScore(byte,byte,byte,byte,byte):void new3BallsMove1PushWithScore(byte,byte,byte,byte):void new3BallsMove2PushNoScore(byte,byte,byte,byte,byte,byte):void new3BallsMove2PushWithScore(byte,byte,byte,byte,byte):void 	

המחלקה אחראית על אחזקת מידע של מהלך אחד.

בהתאם למשתנים של המחלקה, מוחזק מידע של כמות הכדורים של השחקן, כמות הכדורים הנדחפים של היריב, משתנה בוליאני של האם זוהי ניקוד, משתנה בוליאני של האם זהו מהלך שורה או תזוזה הצידה ומערך האינדקסים של המקומות.

private byte numOfOwn;	number of own balls to move.
private byte numToPush;	number of enemy balls to push .
private boolean isScore;	Is score move.
private byte [] indexes;	Indexes move.
private boolean isRowMove;	Is row move. (can be side move).
public Move()	Constructor.
public byte [] getIndexes()	Return indexes array
public void setIndexes(byte[] indexes)	Set indexes array
public boolean isRowMove()	Return isRowMove.
public void setRowMove(boolean isRowMove)	Set isRowMove.
public void setNumOfOwn(byte numOfOwn)	setNumOfOwn.
public void setNumToPush(byte numToPush)	Set NumToPush
public void setScore(boolean isScore)	Set isScore
public byte getNumOfOwn()	Return NumOfOwn
public byte getNumToPush()	Return NumToPush
public boolean isScore()	Return isScore.
public void sendMoveToClient(serverEndPoint point, Board board)	program sends to server end point the updated board from this move. @param point - endpoint to send move to @param board - board to send indexes update.
public void new1BallMove(byte posA, byte posB)	program updates the current class with those parameters. @param posA - first position, own @param posB - second position, empty
public void new2BallsSideMove(byte posA, byte posB, byte posSideA, byte posSideB)	program updates the current class with those parameters. @param posA - first position, own @param posB - second position, own @param posSideA - first side position, empty @param posSideB - second side position, empty
public void new3BallsSideMove(byte posA, byte posB, byte posC, byte posSideA, byte posSideB, byte posSideC)	program updates the current class with those parameters. @param posA - first position, own @param posB - second position, own @param posC - third position, own @param posSideA - first side position, empty @param posSideB - second side position , empty @param posSideC - third side position , empty
public void new2BallsMove0Push(byte posA, byte posB, byte posC)	program updates the current class with those parameters. @param posA - first position , own @param posB -second position , own @param posC- third position , empty
public void new2BallsMove1PushNoScore(byte posA, byte posB, byte posC, byte posD)	program updates the current class with those parameters. @param posA - first position, own @param posB - second position, own @param posC - third position, enemy @param posD - forth position, empty
public void new2BallsMove1PushWithScore(byte posA, byte posB, byte posC)	program updates the current class with those parameters. @param posA - first position, own @param posB - second position, own

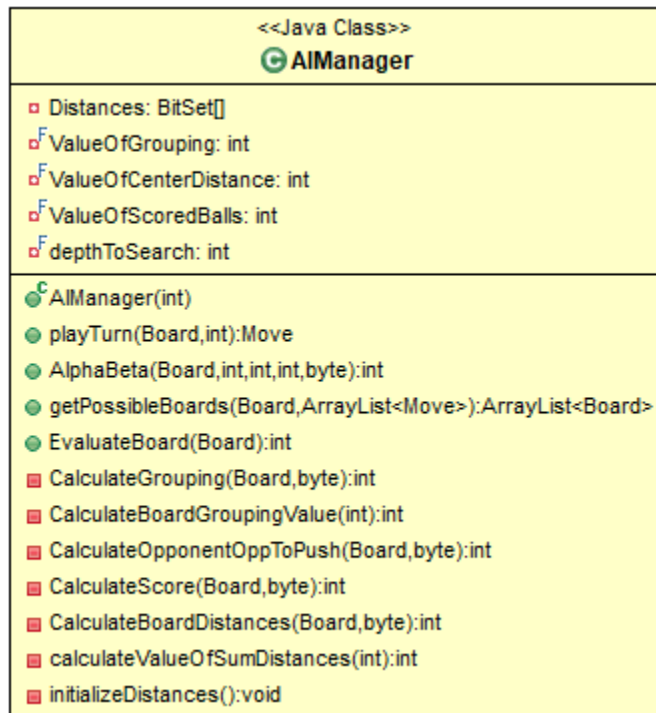
	@param posC - third position, enemy
public void new3BallsMove0Push(byte posA, byte posB, byte posC, byte posD)	program updates the current class with those parameters. @param posA - first position, own @param posB - second position, own @param posC - third position, own @param posD - forth position, empty
public void new3BallsMove1PushNoScore(byte posA, byte posB, byte posC, byte posD, byte posE)	program updates the current class with those parameters. @param posA - first position, own @param posB - second position, own @param posC - third position, own @param posD - forth position, enemy @param posE - fifth position, empty
public void new3BallsMove1PushWithScore(byte posA, byte posB, byte posC, byte posD)	program updates the current class with those parameters. @param posA - first position, own @param posB - second position, own @param posC - third position, own @param posD - forth position, enemy
public void new3BallsMove2PushNoScore(byte posA, byte posB, byte posC, byte posD, byte posE, byte posF)	program updates the current class with those parameters. @param posA - first position, own @param posB - second position, own @param posC - third position, own @param posD - forth position, enemy @param posE - fifth position, enemy @param posF - sixth position, empty
public void <u>new3BallsMove2PushWithScore</u> (byte posA, byte posB, byte posC, byte posD, byte posE)	program updates the current class with those parameters. @param posA - first position, own @param posB - second position, own @param posC - third position, own @param posD - forth position, enemy @param posE - fifth position, enemy



המחלקה אחראית על קליטת מהלך מן הלקוח. המחלקה מחזיקה מערך לחיצות שעבורו שומרת את הלחיצות שהלקוח לחץ. במידה וזוהתה לחיצה שאינה מאפשרת המשך מהלך חוקי, מערך הלחיצות מתאפס וללקוח נשלחת הודעה מתאימה.

private byte [] pressedbuttons;	Pressed places index data structure
private serverEndPoint endpoint ;	Server end point to send messages to.
private Board board ;	The main board.
public <u>PlayerTurnManager</u> (serverEndPoint point, Board b)	Constructor. Initializes pressedbuttons array.
private void resetMiniDataStruct()	program initializes pressedbuttons array.
public Move recivedPress(byte index, int currentplayer)	Main client move i/o. program receives client press index and checks if click is: first click / second click (rowcheck) / third click(rowcheck) / direction click (side or push check) if it is the start of turn then pressedbuttons[0] =[1]= [2] = - 1; @param index - pressed button index @param currentplayer - who pressed. @return Move if end of a valid turn , null if not.
private Move CaseNotFirstPress(byte index, int currentplayer)	program checks if it is a false press, second /third/ 4th press and if it is the last press in turn. @param index - pressed button index @param currentplayer - who pressed. @return Move if end of a valid turn , null if not.
private Move CaseLastPressInTurn(byte index, int currentplayer)	program checks if the last press is part of row/side move. and if it is a valid move. can be 2nd/3rd/4th press. @param index - pressed button index @param currentplayer - who pressed. @return Move if end of a valid turn , null if not.
private void CancleTurn(int currentplayer)	program resets the pressedbuttons array.

	<p>program sends messages to client to reset pressed colors there too.</p> <p>@param currentPlayer - current player.</p>
private void <u>emptyArr</u> ()	<p>program resets pressedbuttons array. (usually after a valid move has found)</p>



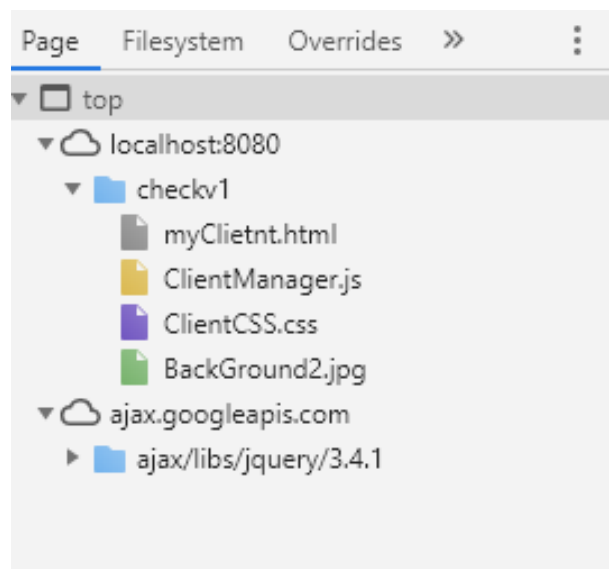
המחלקה אחראית על הפעלת האלגוריתם המרכזי של המערכת. האלגוריתם מקבל מצב לוח ומחזיר עבורו מהלך תקין המייצג את המהלך שהמחשב בחר במצב המשחק שחקן נגד המחשב.

private BitSet [] Distances;	Mask of distances from center data structure
private final int ValueOfGrouping = 7;	Final weight of grouping
private final int ValueOfCenterDistance = 10;	Final weight of distance from center
private final int ValueOfScoredBalls = 1000;	Final weight of points
private final int depthToSearch = 4;	Final AlphaBeta recursion tree depth
public AIManager(int player)	Constructor. Initializes Distances bitsets.
public Move playTurn(Board board, int currentPlayer)	This is the program that activates main algorithm. program uses AlphaBeta and Evaluate function to find the best move of computer. @param board - current board to find move to. @param currentPlayer - player to find move to. @return - best move it can choose.
private int AlphaBeta(Board board, int depth, int alpha, int beta, byte currentPlayer)	This is the main AI recursion algorithm. uses EvaluateBoard to return the best result it can promise. if the program hasn't reached depth 1 than it calculates all possible board and calles again to the function with depth-1 and currentPlayer*-1. @param board - board to calculate from.

	<p>@param depth - current depth to search. if depth =1 than calculates board value and return it.</p> <p>@param alpha - best board value Alpha player can promise</p> <p>@param beta - best board value Beta player can promise</p> <p>@param currentPlayer - player to calculate to</p>
<p>private ArrayList<Board> getPossibleBoards(Board board, ArrayList<Move> ogBoardMoves)</p>	<p>program return list of all possible boards which have implemented the moves from original board and ogBoardMoves list.</p> <p>@param board -board to calculate all its child boards.</p> <p>@param ogBoardMoves - list of all possible moves from originalboard - board.</p> <p>@return list of all possible child boards of board.</p>
<p>private int EvaluateBoard(Board board)</p>	<p>This is the main function of the algorithm.</p> <p>program gets a board and calculates its entire value.</p> <p>the program calculates by the blue player perspective.</p> <p>the program calculates the distancesof the each player balls from center,</p> <p>the grouping of each players balls and the current score.</p> <p>the program returns (value of blue)-(value of red) so it calculates the entire board from blues perspective.</p> <p>@param board - board to calculate</p> <p>@return - board value from blues perspective.</p>
<p>private int CalculateGrouping(Board board, byte side)</p>	<p>program gets move and player, returns value for the grouping of player.</p> <p>@param board - board to calculate</p> <p>@param player - which player to calculate.</p> <p>@return count of grouping*weight (value).</p>
<p>private int <u>CalculateBoardGroupingValue</u>(int cnt)</p>	<p>rogram gets the counter of grouping and return a value of the grouping.</p> <p>@param cnt - neighbors counter.</p> <p>@return cnt*ValueOfGrouping</p>
<p>private int CalculateScore(Board b, byte side)</p>	<p>program calculates the value of scored balls.</p> <p>@param b - board.</p> <p>@param player - player to calculate</p> <p>@return weight*amount of scored balls.</p>
<p>private int CalculateBoardDistances(Board board, byte side)</p>	<p>program balculates the total board distances of given player and returns the value * weight.</p> <p>the program uses 5 bitsets as distances mask.</p> <p>@param board - board</p> <p>@param player</p> <p>@return total value of distances from center * its weight.</p>
<p>private int calculateValueOfSumDistances(int sum)</p>	<p>program gets the sum of distances and rethrn the sum*its weight.</p>

	<p>the program returns the value * -1 so the players will try to get closer to the center and not away from it.</p> <p>@param sum - sum of distances from center.</p> <p>@return - sum*ValueOfCenterDistance*-1</p>
private void initializeDistances()	<p>program initializes and sets 5 bitsets value which help calculate each ball's distance from the center.</p>

קבצים בצד הלקוח:



קובץ myClient.html

הקובץ הוא דף הhtml שהלקוח רואה. לדף זה ישנם 4 כפתורי-רדיו, שני כפתורי בחירת משחק, כותרת אחת, משבצות הניקוד ומשבצות המשחק.

כל משבצת במשחק מיוצגת לפי אינדקס, שחופפים לאינדקסים בשרת.

ייבוא קובץ jquery

ע"י יבוא הקובץ jquery מן האינטרנט, הצלחתי למצוא ביעילות מהו האינדקס של המשבצת בלוח ששנלחצה.

קובץ ClientManager.js

זהו הקובץ שכולל את הפעולות והתקשורת של הלקוח מול השרת. מלבד פעולות ההתחברות וההתנתקות מהשרת, קיימות שתי פעולות עיקריות – קבלת ושליחת הודעה.

<code>function processMessage(message)</code>	program receives a message from server. the message can update one value in position or blue win found or red won found. if it is an position update than message will be JSONObject. if not than simple string.
<code>\$('.btnPVP').click(function()</code>	Once the "start game player vs player" button pressed, the program sends the server message to start the game with this game mode. The function gets the pressed board layout and sends it to server as well.

<code>\$('.btnPVAI').click(function()</code>	Once the "start game player vs Computer" button pressed, the program sends the server message to start the game with this game mode. The function gets the pressed board layout and sends it to server as well
<code>\$(".cell").click(function(evt)</code>	Once the client pressed on one of the board buttons, The program sends the client its index using JSONObject.

בשלושת הפעולות האחרונות לשליחת הודעה לשרת, השתמשתי ב jquery בכדי למצוא את האינדקס המתאים.

קובץ ClientCSS.css

קובץ זה הוא קובץ עזר לעיצוב דף הhtml. בקובץ זה מתוארים ההגדרות והמאפיינים של משבצות הלוח, השורות ביניהם, משבצות הניקוד והכפתורים.

מדריך למשתמש

על מנת לפתוח את הפרויקט ולהריץ את הקוד, יש תחילה לפתוח פרויקט חדש באמצעות IDE (אני משתמש ב-Eclipse).

בתוך Eclipse יש לפתוח פרויקט חדש- new web application. יש להוריד שרת (אני משתמש ב-Tomcat server) ולבחור אותו בתוך הפרויקט החדש.

לאחר מכן יש להעתיק את כל קבצי הקוד לתוך הפרויקט וללחוץ על הכפתור הירוק להפעלת השרת. יש להכניס את שלושת קבצי WebContent לתיקיה הזו ואת שאר ששת הקבצים לתיקיית sec.

ניתן להתחבר לאתר דרך הדפדפן המובנה ב-Eclipse או להעתיק את הכתובת <http://localhost:8080/checkv1/myClientnt.html> לתוך דפדפן כמו כרום.

ניתן להתחבר לאתר דרך דפדפן במחשב אחר, בתנאי שהמחשבים נמצאים באותו LAN. בשביל לעשות זו יש למצוא את כתובת ה ip של המחשב שמריץ את הקוד ולהיכנס לקישור <http://---:8080/checkv1/myClientnt.html> כאשר במקום ה--- יש להכניס את כתובת ה ip.

בסרגל הבחירה ניתן לבחור באיזה צורה יתחיל המשחק (Classic / Pro/ Snake/ Wall) ולבחור על הכפתור Player vs Computer או Player vs Player בהתאמה למה שרוצים לשחק.

במהלך כל תור, יש לבחור את הכדורים של זה שתורו וללחוץ על מיקום שאליו רוצים להזיז את הכדורים. במידה ונלחץ מיקום שאינו מאפשר המשך של מהלך תקין, הכדורים שכבר נבחרו יתאפסו ועל השחקן לבחור מחדש את המהלך שאותו רוצה לבצע.

בכל שלב ניתן לשנות את צורת הלוח וללחוץ מחדש על התחלת משחק.

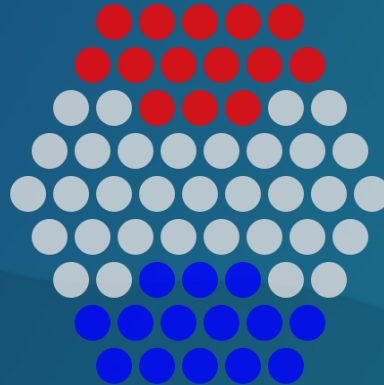
לסגירת הפרויקט יש ללחוץ על הכפתור האדום ב-Eclipse.

Welcome to Abalone!

•Classic •Pro •Snake •Wall

Start Game Player VS Player

Start Game Player VS Computer



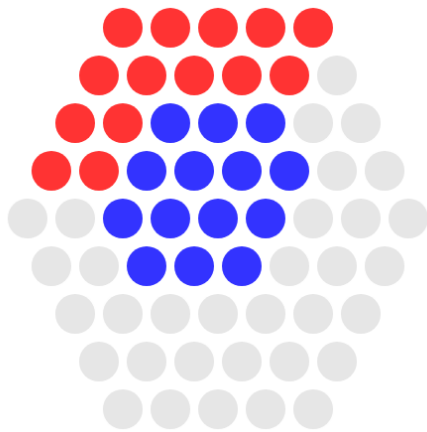
מסמך עיצוב

איתי בנבנישתי – Abalone

כחלק מתכנון הפרויקט, הייתי צריך לחשוב על המחלקות ומבני הנתונים העיקריים בפרויקט. מסמך זה מתייחס למבני הנתונים, המחקר שביצעתי, GUI ותרשים המחלקות UML.

מחקר: כחלק מתכנון הפרויקט חיפשתי מאמרים רבים שאנשים ביצעו לגבי המחקר. מצאתי פרטים רבים לגבי המשחק.

- סיבוכיות המשחק: בכל תור ממוצע יש כ-60 אפשרויות משחק שונים. זוהי סיבוכיות גבוהה מאוד ביחס לשאר המשחקים כך שפתרון אלגוריתם הבנוי על הסתכלות כל המהלכים האפשריים ולפי זה להחליט מהו הלוח הטוב ביותר אינו רלוונטי למשחק זה. אמנם, בניגוד למשחקים כמו שחמט שבהם יש הרבה סוגי שחקנים, באבלון כל חלקי המשחק בעלי אותם תכונות- דבר שמקל משמעותית על ייצוג הלוח.
- אסטרטגיות משחק: צפיתי בהרבה משחקים ומחקרים אשר הראו כי ישנם כמה מצבים אשר באמצעותם ניתן לדרג את מצב המשחק כשחקן נגד שחקן. מצבים אלו ישתקפו בפונקציית ההערכת הלוח (evaluation function) אשר תקבע עבור אלגוריתם ה-Al מהו המהלך הטוב ביותר עבורו.
- 1. מרכז: אחת משתי המטרות העיקריות של השחקן הוא להשיג שליטה יחסית (מול היריב) על אזור אמצע הלוח. דבר זה יקנה לו פחות סיכון בלאבד כדורים אשר יכולים להידחף לגבולות הלוח ואפשרויות דחיפה של כדורי היריב אל גבולות הלוח ובכך להשיג ניצחון.
- 2. קיבוץ: המטרה העיקרית השנייה של השחקן הוא לשמור על כל הכדורים שלו בקבוצה אחד צמודה. דבר זה יאפשר לו יותר אפשרויות דחיפה מול היריב(יותר קבוצות של 3 כדורים בטור) וימנע אפשרויות דחיפה של היריב כלפיו ובכך יעזור להשגת הניצחון.



השאפה העיקרית של כל שחקן היא להגיע למצב בסגנון הבא, ובכך להבטיח את נצחונו. (כחול מנצח).

במצב זה לשחקן האדום יש מעט מאוד אפשרויות משחק ו0 אפשרויות דחיפה, לעומת השחקן הכחול שיש לו משמעותית הרבה יותר אפשרויות משחק כולל אפשרויות להשגת ניקוד באופן מידי.

מבנה נתונים:

האפשרויות העיקריות של מבנה נתונים שעמדו ליפני הם:

BitSet, HashMap, Matrix.

פסלתי את המערך הדו מימדי כיוון שלוח המשחק הינו מוששה ובכך לוח דו מימדי יקשה על הבדיקות השונות על הלוח ולא תהייה יעילות טובה כך.

היעילות של HashMap ו- BitSet יחסית דומות אך בחרתי ב BitSet מה שייקנה לי דרך ויעילות טובה בקריאה לפונקציית AI.

ייצגתי את הלוח כשני BitSet כאשר כל אחד מהם ייצג את החיילים של שחקן מסוים. גודל כל BitSet יהיה 61 כגודל הלוח והביטים הדלוקים בכל אחד מהם יהווה את הכדורים של כל שחקן בהתאמה.

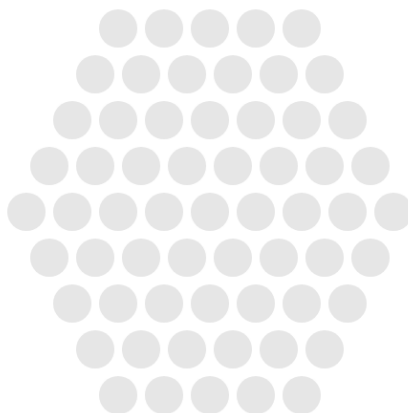
GUI – בחרתי להכין את הפרויקט כתוכנת שרת-לקוח מבוסס WebSocket.

רציתי להתנסות בפיתוח Web ולבנות את ה GUI ב CSS, JavaScript ו- jQuery.

הלוח נראה בשלב זה כך:

welcome to abalone!

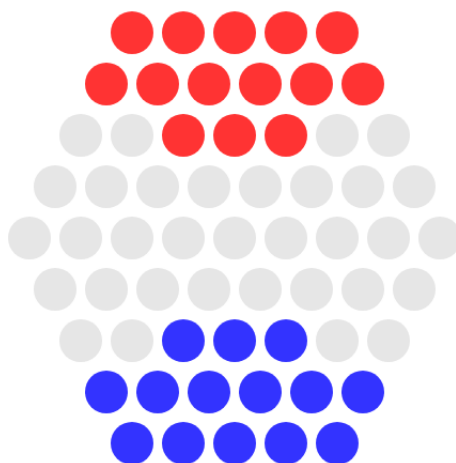
Start Game Player VS Player



וכך לאחר לחיצה על הכפתור.

welcome to abalone!

Start Game Player VS Player



השלב הבא בפיתוח יהיה הכנת מצב של שחקן נגד שחקן ובכך לממש את פעולות בדיקת תקינות המהלך במחלקת Board.

לאחר מכן אתחיל בפיתוח אלגוריתם הAl ואז אחזור לעיצוב הGUI של הלקוח שיראה יותר טוב.

קישורים חיצוניים:

[מאמר 1 : מחקר שנעשה על המשחק Abalone.](#)

[מאמר 2 : מחקר שנעשה על המשחק Abalone.](#)

[מאמר 3: ייצוג לוח Hexagonal במבנה נתונים.](#)

[מקור 4: פעולות שניתן לבצע בBitSet.](#)

[מקור 5: משחק אבולון באינטרנט.](#)

[מקור 6: משחק אבולון באינטרנט.](#)

```
package AbaloneGame;

import java.io.IOException;
import javax.websocket.OnClose;
import javax.websocket.OnError;
import javax.websocket.OnMessage;
import javax.websocket.OnOpen;
import javax.websocket.Session;
import javax.websocket.server.ServerEndpoint;

import org.json.JSONException;
import org.json.JSONObject;

@ServerEndpoint("/serverEndPoint")
public class serverEndPoint
{
    Session Mainsession;
    GameManager gm;
    int BoardLayout = 0;

    @OnOpen
    /**
     * The program handles the start of the connection between server and client.
     * The program gets session ses which is the mainsession of the server.
     * @param ses - the session of client
     */
    public void handleOpen(Session ses)
    {
        gm = new GameManager(this);
        System.out.println("client is now connecting...");
        Mainsession = ses;
    }

    @OnMessage
    /**
     * The program handles all the messages which the server receives.
     * The program gets message which can be a simple string or JSONObject.
     * @param message - the message from client
     */
    public void handleMessage(String message)
    {

        switch(message)
        {
```



```

case "Start Player VS Player":
{
    System.out.println("received message to start game Player vs Player ");
    gm = new GameManager(this);
    gm.StartGamePlayerVsPlayer(BoardLayout);
    break;
}
case "Start Player VS AI":
{
    System.out.println("received message to start game AI vs Player ");
    gm = new GameManager(this);
    gm.StartGamePlayerVsAI(BoardLayout);
    break;
}
case "Classic Board Layout":
{
    BoardLayout = 0;
    break;
}
case "Pro Board Layout":
{
    BoardLayout = 1;
    break;
}
case "Snake Board Layout":
{
    BoardLayout = 2;
    break;
}
case "Wall Board Layout":
{
    BoardLayout = 3;
    break;
}
default:
{
    System.out.println("received from client " + message);

    JSONObject obj = new JSONObject();
    try {
        obj = new JSONObject(message);
    } catch (JSONException e) {
        System.out.println("error json 1234");
        e.printStackTrace();
    }
    try {
        //int posX = Integer.parseInt((String) obj.get("positoinX"));
        //int posY = Integer.parseInt((String) obj.get("positoinY"));
        int posindex = Integer.parseInt((String) obj.get("posIndex"));
        gm.rereceivedMessage(posindex);

    } catch (NumberFormatException | JSONException e) {
        System.out.println("error json 122334");
        e.printStackTrace();
    }
}

```

```

        }
        break;
    }
}

```

```

/**
 * the program updates at the client one position.
 * send to client the position and the new value to put inside.
 * @param indexx - the index of position.
 * @param val - the value of position.
 */
public void sendmessage(int indexx, int val)
{
    JSONObject obje = new JSONObject();
    try {
        obje.put("index", indexx);
        obje.put("value", val);

    } catch (JSONException e) {
        e.printStackTrace();
    }

    String mess = obje.toString();

    try {
        Mainsession.getBasicRemote().sendText(mess);
        System.out.println("sent to client " + mess);

    } catch (IOException e) {
        e.printStackTrace();
    }
}

```

```

@OnClose
/**
 * program prints that sever has disconnected
 */
public void handleClose()
{
    System.out.println("client is now disconnecting...");
}

```

```

@OnError
/**

```

```

    * the program prints that error have occurred
    * @param t - the error
    */
    public void handleError(Throwable t)
    {
        System.out.println("error1111");
        t.printStackTrace();
    }

    /**
     * the program sends the client the player who won the game.
     * 1 for blue, -1 for red.
     * @param value - player number.
     */
    public void sendWinMessage(int value)
    {
        String a = Integer.toString(value);
        try {
            Mainsession.getBasicRemote().sendText("win found "+ a );
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
}

```

```

package AbaloneGame;

import org.apache.tomcat.jni.Time;

public class GameManager
{
    // Server endpoint to move messages to client
    private serverEndPoint endpoint;
    //
    private int currentplayer;
    //board
    private Board board;
    //player (client) turn manager
    private PlayerTurnManager Pturn;
    //AI manager
    private AIManager AI;
    //game mode
    boolean isPVP;
    //is waiting for user input.
    boolean isWaitingToPlayer = false;

    /**
     * Constructor.
     * @param point - server end point
     */
    public GameManager(serverEndPoint point)
    {
        this.currentplayer = 1;
        this.endpoint = point;
        this.board = new Board();
    }

    /**
     * the program get called once the client pressed the start game button.
     * the program initializes the board and sends messeges of the board to the client.
     * @param BoardLayout - the game layout.
     */
    public void StartGamePlayerVsPlayer(int BoardLayout)
    {
        //init the board values
        //and send data to client

        board.initializeBoard(BoardLayout);
        Pturn = new PlayerTurnManager(endpoint, board);
        isPVP = true;
        board.sendIntireBoardToClient(endpoint);
        resetScoreInClient(endpoint);
        isWaitingToPlayer = true;
    }
}

```

```

/**
 * program gets called when client presses "start game player vs AI"
 * program initialize required parts in code.
 * @param BoardLayout - the game layout.
 */
public void StartGamePlayerVsAI(int BoardLayout)
{
    board.initializeBoard(BoardLayout);
    Pturn = new PlayerTurnManager(endpoint, board);
    isPVP = false;
    board.sendIntireBoardToClient(endpoint);
    resetScoreInClient(endpoint);
    AI= new AIManager(-1);
    isWaitingToPlayer = true;
}

/**
 * program switches beetween players/AI.
 * If the game mode is pvAI than its activates the ai.
 */
public void switchPlayers()
{
    if(isPVP== true)
    {
        System.out.println("swithced player from " +currentplayer + "to " + currentplayer*-
1);
        currentplayer = currentplayer *-1;
    }
    else
    {
        //switch to player -> AI -> AI or AI-> player
        //1 is player
        if(currentplayer == 1)
        {
            //switch from player to AI
            System.out.println("swithced player from " +currentplayer + " to AI turn as " +
currentplayer*-1);
            isWaitingToPlayer = false;

            //gets move from AI
            Move AIMove = AI.playTurn(board, currentplayer*-1);

            //mark move to client
            markMoveToClient(AIMove);

            // implementing move in game board
            int iswin = board.makeMove(AIMove);

```

```

        sendMoveToClient(AIMove);

        if(iswin==1)
            this.WinFound(currentplayer*-1);

        isWaitingToPlayer = true;
    }
    //after turn has been committed, its player turn again.
}

/**
 * program gets move (AI move) and mark move in client.
 * includes time.wait so client can see the move.
 * @param aIMove - AI move
 */
public void markMoveToClient(Move aIMove)
{
    byte [] indexes = aIMove.getIndexs();
    for(int i=0;i<aIMove.getNumOfOwn();i++)
    {
        endpoint.sendMessage(indexes[i], board.GetValueInPosition(indexes[i])*2);
    }
    Time.sleep(700000);
}

/**
 * program gets called once a win was found.
 * program sends the data to the endpoint to declare a winner to the client.
 * @param value - player number
 */
public void WinFound(int value)
{
    System.out.println("GameManager.WinFound()");
    endpoint.sendWinMessage(-value);
}

/**
 * program recives a press index of client, sends the index to the turn
 * manager which returns a move.
 * if it is not end of turn than move = null.
 * @param index - client press index.
 */
public void rereceivedMessage(int index)
{
    int iswin=0;
    if(isWaitingToPlayer)
    {

```

```

//manages the player turn input.
Move move= Pturn.recivedPress((byte)index, currentplayer);
if(move != null)
{
    //implements the move inside the board.
    iswin = board.makeMove(move);
    //sends the given play to the client to see.
    sendMoveToClient(move);

    if(iswin==1)
        this.WinFound(currentplayer);

    //switches player.
    switchPlayers();
}
}
}

```

```

/**
 * program gets move and sends client the updated positions of board.
 * @param move - move to implement at client
 */
public void sendMoveToClient(Move move)
{
    move.sendMoveToClient(endpoint, this.board);
}

```

```

/**
 * the program resets the value inside the score positions in client.
 * from -10 to -16
 * and from -20 to -26
 * @param point - the server end point
 */
public void resetScoreInClient(serverEndPoint point)
{
    for(int i=0;i<=6;i++)
    {
        point.sendmessage(-10-i, 4);
        point.sendmessage(-20-i, 4);
    }
}

```

```

}

```

```

package AbaloneGame;

import java.util.BitSet;
import java.util.ArrayList;

public class Board
{
    //data structure:
    private BitSet EdgeOfBoard;
    private BitSet BlueSet;
    private BitSet RedSet;

    //directions matrix
    private byte pathArr[];

    //score count
    private byte scoreBlue;
    private byte scoreRed;

    /**
     * Constructor
     * initializes patharr
     */
    public Board()
    {
        pathArr = new byte[6];
        initpathArr();
    }

    public BitSet getEdgeOfBoard() {
        return EdgeOfBoard;
    }

    public void setEdgeOfBoard(BitSet edgeOfBoard) {
        EdgeOfBoard = edgeOfBoard;
    }

    public BitSet getBlueSet() {
        return BlueSet;
    }

    public void setBlueSet(BitSet blueSet) {
        BlueSet = blueSet;
    }

    public BitSet getRedSet() {
        return RedSet;
    }
}

```



```

public void setRedSet(BitSet redSet) {
    RedSet = redSet;
}

    public byte getScoreBlue() {
        return scoreBlue;
    }

public void setScoreBlue(byte scoreBlue) {
    this.scoreBlue = scoreBlue;
}

public byte getScoreRed() {
    return scoreRed;
}

public void setScoreRed(byte scoreRed) {
    this.scoreRed = scoreRed;
}

```

```

/**
 *program initializes the initpathArr
 */
public void initpathArr()
{
    pathArr = new byte[6];
    pathArr[0] = 1;
    pathArr[1]= -12;
    pathArr[2]= -11;
    pathArr[3]= -1;
    pathArr[4]= 12;
    pathArr[5]= 11;
}

```

```

/**
 * program creates new 3 bitsets -
 * blueSet, RedSet, EdgeOfBoard.
 * @param BoardLayout - sets the start game layout according to the BoardLayout
 */
public void initializeBoard(int BoardLayout)
{
    initpathArr();
    scoreBlue = 0;
    scoreRed = 0;

    EdgeOfBoard = new BitSet(121);
    BlueSet = new BitSet(121);
    RedSet = new BitSet(121);

    ///board edges layout.

```

```

EdgeOfBoard.set(12, 17);
EdgeOfBoard.set(23, 29);
EdgeOfBoard.set(34, 41);
EdgeOfBoard.set(45, 53);
EdgeOfBoard.set(56, 65);
EdgeOfBoard.set(68, 76);
EdgeOfBoard.set(80, 87);
EdgeOfBoard.set(92, 98);
EdgeOfBoard.set(104, 109);

//classic layout
if(BoardLayout==0)
{
    ///blueSet, player 1
    BlueSet.set(12,14);
    BlueSet.set(23,25 );
    BlueSet.set(34,37 );
    BlueSet.set(45,48);
    BlueSet.set(56,59);
    BlueSet.set(68);

    //Redset, player -1
    RedSet.set(52);
    RedSet.set(62,65);
    RedSet.set(73, 76);
    RedSet.set(84, 87);
    RedSet.set(96, 98);
    RedSet.set(107, 109);
}

//pro layout
if(BoardLayout ==1)
{
    //top left
    BlueSet.set(39,41);
    BlueSet.set(50,53);
    BlueSet.set(62,64);

    //top right
    RedSet.set(83, 85);
    RedSet.set(94, 97);
    RedSet.set(106, 108);

    //bottom left
    RedSet.set(13, 15);
    RedSet.set(24, 27);
    RedSet.set(36, 38);

    //bottom right
    BlueSet.set(57,59);
    BlueSet.set(68,71);
    BlueSet.set(80,82);
}

//snake layout

```

```

if(BoardLayout == 2)
{
    //start in bottom left
    BlueSet.set(12);
    BlueSet.set(23);
    BlueSet.set(34);
    BlueSet.set(45);
    BlueSet.set(56);
    BlueSet.set(68);
    BlueSet.set(80);
    BlueSet.set(92);
    BlueSet.set(93);
    BlueSet.set(94);
    BlueSet.set(83);
    BlueSet.set(71);
    BlueSet.set(59);
    BlueSet.set(48);

    //starts in top right
    RedSet.set(108);
    RedSet.set(97);
    RedSet.set(86);
    RedSet.set(75);
    RedSet.set(64);
    RedSet.set(52);
    RedSet.set(40);
    RedSet.set(28);
    RedSet.set(27);
    RedSet.set(26);
    RedSet.set(37);
    RedSet.set(49);
    RedSet.set(61);
    RedSet.set(72);
}

//Wall layout
if(BoardLayout ==3)
{
    //top left
    RedSet.set(28);
    RedSet.set(39);
    RedSet.set(50,52);
    RedSet.set(61,63);
    RedSet.set(72,74);
    RedSet.set(83,85);
    RedSet.set(94,96);
    RedSet.set(105);
    RedSet.set(86);

    //bottom left
    BlueSet.set(15);
    BlueSet.set(25, 27);
}

```

```

        BlueSet.set(36,38);
        BlueSet.set(47,49);
        BlueSet.set(58,60);
        BlueSet.set(69,71);
        BlueSet.set(81);
        BlueSet.set(92);
        BlueSet.set(34);
    }
}

```

```

/**
 * program sends to server end point all of the positions in the current board.
 * @param point - sends messages to.
 */
public void sendIntireBoardToClient(serverEndPoint point)
{
    for (int i = EdgeOfBoard.nextSetBit(0); i != -1; i = EdgeOfBoard.nextSetBit(i+1))
    {
        point.sendMessage(i, 0);
    }
    for (int i = BlueSet.nextSetBit(0); i != -1; i = BlueSet.nextSetBit(i+1))
    {
        point.sendMessage(i, 1);
    }
    for (int i = RedSet.nextSetBit(0); i != -1; i = RedSet.nextSetBit(i+1))
    {
        point.sendMessage(i, -1);
    }
}

```

```

/**
 * function return the value of position in board.
 * @param position - position to check value for.
 * @return - -9 if not in board, 1 if in blueset, -1 if in redset ,0 if not pressed (not in
both).
 */
//function gets bit position and return 1 if in blueset,
//-1 if in redset and 0 if not pressed (not in both).
//return -9 if not in board.
public byte GetValueInPosition(byte position)
{
    //not in board
    if(!isPositionInBoard(position))
        return -9;

    if(BlueSet.get(position))
        return 1;
}

```

```

        if(RedSet.get(position))
            return -1;
        //if reached than no one is true so empty.
        return 0;
    }

/**
 * program sets value in given position
 * @param position - position to change.
 * @param value - the value to change to.
 */
public void SetValueInPosition(byte position, byte value)
{
    switch (value) {
        case 1:
        {
            BlueSet.set(position, true);
            RedSet.set(position, false);
            break;
        }
        case 0:
        {
            BlueSet.set(position, false);
            RedSet.set(position, false);
            break;
        }
        case -1:
        {
            BlueSet.set(position, false);
            RedSet.set(position, true);
            break;
        }
    }
}

/**
 *program gets position and return true if in board.
 * @param position - position to check
 * @return - true if in board, false if not.
 */
public boolean isPositionInBoard(byte position)
{
    return EdgeOfBoard.get(position);
}

/**
 * program check if teo positions are next to each other.

```

```

* !! does not check if 2 positions are in board.
* @param posA - first position
* @param posB - second position
* @return true if both positions are next to each other.
*/
public boolean IsPositionsTogether(byte posA, byte posB)
{
    byte givenDirection = (byte) (posB - posA);
    for (byte direction : pathArr)
    {
        //checks if direction is in directions array
        if(direction==givenDirection)
            return true;
    }
    //if reached than they are not together.
    return false;
}

/**
* program gets 2 position and return the next position in line.
* @param posA - first position
* @param posB - second position
* @return posC if position found, -9 if position is not in board.
*/
public byte getNextPositionInLine(byte posA, byte posB)
{
    byte posC = (byte) (posB- posA+ posB);
    if(isPositionInBoard(posC))
        return posC;

    return -9;
}

/**
* program returns the position of posSideA
* @param posA - first position
* @param posB - second position, next to posA
* @param posSideB - second side position, in side to posB
* @return -9 if posSideA is not in board, else returns posA.
*/
public byte get4thPosInSideMove(byte posA, byte posB, byte posSideB)
{
    byte direction = (byte) (posSideB- posB);
    if(!isPositionInBoard((byte)(posA+direction)))
        return -9;
    return (byte) (posA+direction);
}

```

```

/**
 * program gets a valid move and updates current board from the move.
 * @param move - valid move to implement.
 * @return 1 if Winfound, 0 of not.
 */
public int makeMove(Move move)
{
    //System.out.println("started MakMove fnc.");
    //move index:
    byte indexesarray [] = move.getIndexes();
    //in all moves the first position will be 0;
    byte ownV = GetValueInPosition(indexesarray[0]);

    if(move.isRowMove())
    {
        SetValueInPosition(indexesarray[0], (byte) 0);
        //send all own moves including next.
        for(int i=1 ; i<move.getNumOfOwn()+1; i++)
        {
            SetValueInPosition(indexesarray[i], ownV);
        }

        ///if(move.isScore)
        //    move.numToPush--; // updates -1 balls.

        //updates enemy balls.
        for(int i1=0;i1<move.getNumToPush();i1++)
        {
            //System.out.println("moves enemy to index " + move.numOfOwn+1+i1);
            //System.out.println(" in " + move.indexs[move.numOfOwn+i1]);
            SetValueInPosition(indexesarray[move.getNumOfOwn()+1+i1], (byte) (ownV*-1));
        }
    }
    else
    {
        // side move.
        for(int i = 0 ; i<move.getNumOfOwn(); i++)
        {
            SetValueInPosition(indexesarray[i], (byte)0);
            SetValueInPosition(indexesarray[i+move.getNumOfOwn()],(byte) ownV);
        }
    }

    //checks if win found
    if(move.isScore())

```

```

{
    if(ownV==1)
    {
        scoreBlue++;
        if(scoreBlue>=6)
            return 1;
    }
    else
    {
        scoreRed++;
        if(scoreRed>=6)
            return 1;
    }
}
//win not found
return 0;
}

```

```

//////////////////////////////////////Moves check//////////////////////////////////////
//////////////////////////////////////

```

```

/**
 * program gets two points: A and B are next to each other.
 * the programs return if posC is in line after B.
 * @param posA - first position
 * @param posB - second position (next to posA)
 * @param posC - third position
 * @return true if posA, posB and posC are in a line.
 */
public boolean isPositionsInLine3(byte posA, byte posB, byte posC)
{
    //checks if the change between x and y values is the same.
    return ((posB-posA) == (posC - posB));
}

```

```

/**
 * the program switches the values of two given positions.
 * @param posA - first position
 * @param posB - second position
 */
public void switchPositions(byte posA, byte posB)
{
    byte posAValue = GetValueInPosition(posA);
    byte posBValue = GetValueInPosition(posB);

    BlueSet.set(posA, posBValue);
    RedSet.set(posB, posAValue);
}

```



```
}
```

```
/**
```

```
* program tries to create a move of push 2 balls in the direction of posC.  
* @param posA - first position (own)  
* @param posB - second position(next to posA , own)  
* @param posC - third position (in line) can be (valueof posA)*-1 or 0  
* @return move if found a valid move, null if not.  
*/
```

```
public Move TryToPush2(byte posA, byte posB, byte posC)  
{  
    //if empty then switch posA and posC  
    byte VposC = GetValueInPosition(posC);  
  
    //2 balls push.  
    if(VposC==0)  
    {  
        //creates new Move  
        Move mov = new Move();  
        mov.new2BallsMove0Push(posA, posB, posC);  
  
        return mov;  
    }  
  
    //posC is Enemy  
    //to check where is posD  
  
    byte posD = getNextPositionInLine(posB, posC);  
    if(posD == -9)  
    { //point push.  
        //creates new Move  
        Move mov = new Move();  
        mov.new2BallsMove1PushWithScore(posA, posB, posC);  
        return mov;  
    }  
  
    byte VposD = GetValueInPosition(posD);  
  
    //2 own and 2 enemy -> cannot push  
    if(VposD != 0)  
        return null;  
  
    //if reached than 2 own 1 enemy 1 empty, can push.  
    Move mov = new Move();  
    mov.new2BallsMove1PushNoScore(posA, posB, posC, posD);  
    return mov;  
}
```

```

/**
 * program tries to create a move of push 3 balls in the direction of posD.
 * @param posA - first position (own)
 * @param posB - second position(next to posA , own)
 * @param posC - second position(in line , own)
 * @param posD - third position (in line) can be (sizeof posA)*-1 or 0
 * @return move if found a valid move, null if not.
 */
public Move TryToPush3(byte posA,byte posB,byte posC,byte posD)
{
    if(GetValueInPosition(posD)==0)
    { //3 balls move no push.
        Move mov = new Move();
        mov.new3BallsMove0Push(posA, posB, posC, posD);
        return mov;
    }

    byte Vown = GetValueInPosition(posA);
    //checks if 4 balls same color.
    if(GetValueInPosition(posD) == Vown)
        return null;

    //if reached than 3 own 1 enemy.
    byte posE = getNextPositionInLine(posC, posD);
    if(posE == -9)
    { //3 balls move 1 push with score.
        Move mov = new Move();
        mov.new3BallsMove1PushWithScore(posA, posB, posC, posD);
        return mov;
    }

    //checks if 3 own 1 enemy 1 own.
    byte VposE = GetValueInPosition(posE);
    if(VposE==Vown)
        return null;

    if(VposE == 0)
    { //3 own 1 enemy push no score.
        Move mov = new Move();
        mov.new3BallsMove1PushNoScore(posA, posB, posC, posD, posE);
        return mov;
    }

    //if reached than 3 own 2 enemy.

    byte posF = getNextPositionInLine(posD, posE);
    if(posF == -9)
    { //3 balls move 2 push with score.
        Move mov = new Move();
        mov.new3BallsMove2PushWithScore(posA, posB, posC, posD, posE);
        return mov;
    }
}

```

```

    }

    //checks if 3 own 2 enemy 1 own.
    byte VposF = GetValueInPosition(posF);
    if(VposF==Vown)
        return null;

    if(VposF == 0)
    { //3 own 2 enemy push no score.
        Move mov = new Move();
        mov.new3BallsMove2PushNoScore(posA, posB, posC, posD, posE, posF);
        return mov;
    }

    //3 own 3 enemy -> cannot do anything.
    return null;
}

/**
 * program check if can do a side move of posA and posB to posSideA and posSideB
 * @param posA - first position
 * @param posB - second position (next to posA , own)
 * @param posSideB - first sids position , empty.
 * @return move if found a valid move, null if not.
 */
public Move TryToSideMove2(byte posA, byte posB, byte posSideB)
{
    ///both side poses must be 0;
    if(GetValueInPosition(posSideB)!=0)
        return null;

    //checks if possideB is next to one of given positions.
    if(!IsPositionsTogether(posB, posSideB))
        return null;

    //if reached than posSideB is Valid.

    //checks posSideA.
    byte posSideA = get4thPosInSideMove(posA, posB, posSideB);
    if(posSideA== -9 || GetValueInPosition(posSideA)!=0)
        return null;

    //if reached than it is a valid 2 side move.
    //checks if 2nd move position is in board
    Move mov = new Move();
    mov.new2BallsSideMove(posA, posB, posSideA, posSideB);
    return mov;
}

```

```

/**
 * program check if can do a side move of posA, posB and posC to posSideA and posSideB and
posSideC
 * @param posA - first position
 * @param posB - second position (next to posA , own)
 * @param posB - second position (in line , own)
 * @param posSideC - first side position , empty.
 * @return move if found a valid move, null if not.
 */
public Move TryToSideMove3(byte posA, byte posB, byte posC, byte posSideC)
{
    ///both side poses must be 0;
    if(GetValueInPosition(posSideC)!=0)
        return null;

    //checks if possideC is next to one of given positions.
    if(!IsPositionsTogether(posC, posSideC))
        return null;

    //if reached than posSideB is Valid.

    //checks posSideB.
    byte posSideB = get4thPosInSideMove(posB, posC, posSideC);
    if(posSideB==-9 || GetValueInPosition(posSideB)!=0)
        return null;

    //checks posSideA.
    byte posSideA = get4thPosInSideMove(posA, posB, posSideB);
    if(posSideA==-9 || GetValueInPosition(posSideA)!=0)
        return null;

    //if reached than it is a valid 2 side move.
    //checks if 2nd move position is in board
    Move mov = new Move();
    mov.new3BallsSideMove(posA, posB, posC, posSideA, posSideB, posSideC);
    return mov;
}

```

```

////////////////////////////////////
////
////////////////////////////////////AI
functions////////////////////////////////
////////////////////////////////////
////

```

```

/**
 * program finds all the possible moves of Currentplayer from current board.
 * @param currentPlayer - player to search moves for.
 * @return list of all possible moves of current player.
 */
public ArrayList<Move> getmoves(byte currentPlayer)
{
    //ArrayList<Move> MoveList = new ArrayList<Move>();
    ArrayList<Move> MoveList = new ArrayList<Move>(60);

    Move move = new Move();
    byte possiblePoositionsArray [], sideArray[];

    //set own and enemy sets
    BitSet ownSet = (currentPlayer==1)? BlueSet: RedSet;
    //BitSet enemySet = (currentPlayer==1)? RedSet: BlueSet;

    //go ovet all balls in bitset to put as first click.
    for (byte posA = (byte)ownSet.nextSetBit(0); posA != -1; posA = (byte)ownSet.nextSetBit(posA
+ 1))
    {
        //gest all neighbors positions.
        possiblePoositionsArray = getNeighborsOfPossition(posA);
        /// go over each of neighbors positions in posB.
        for (byte posB : possiblePoositionsArray)
        {
            //if position is not a valid neighbor.
            if(posB ==-1)
                continue;
            //checks if posB is Empty
            byte VposB = GetValueInPosition(posB);
            if(VposB==0)
            {
                //add 2 points, 1 of own, 0 push, 0 score
                move = new Move();
                move.new1BallMove(posA, posB);
                //adds move to list
                MoveList.add(move);
            }
            else
            {
                //posB is not empty

                //if enemy then cannot push (move already checked) then break.
                if(VposB == currentPlayer *-1) continue;

                //next postition in line, return -1 if not in board
                byte posC = getNextPositionInLine(posA, posB);
                if(posC!=-9)
                {
                    //posC is inBoard.
                    if(GetValueInPosition(posC) == currentPlayer)

```

```

{
    byte posD = getNextPositionInLine(posB, posC);
    if(posD!=-9)
    {
        move = new Move();
        move = TryToPush3(posA, posB, posC, posD);
        if(move != null)
            MoveList.add(move);
    }
}
else
{
    //can be 2 own push.
    move = new Move();
    move = TryToPush2(posA, posB, posC);
    if(move != null)
        MoveList.add(move);
}
}

```

```

//to check all possible side moves of 2 balls.
sideArray = getNeighborsOfPosition(posB);
// go over each of neighbors positions from posB.
for (byte posSideB : sideArray)
{

```

```

    //if position is not a valid neighbor.
    if(posSideB ==-1)
        continue;

```

```

    //doesn't check posA (already pressed) or posC (already checked in

```

line move).

```

    if( posSideB == posA || posSideB == posB)
        continue;

```

```

    if(GetValueInPosition(posSideB) != 0)
        continue;

```

```

    //gets posSideA index, if not in board then -1.
    byte posSideA = get4thPosInSideMove(posA, posB, posSideB);
    if(posSideA == -9 || GetValueInPosition(posSideA) != 0)
        continue;

```

```

    //if reached then it is a sideMove in positions posA, posB to

```

posSideA and posSideB

```

    //
    move = new Move();
    move.new2BallsSideMove(posA, posB, posSideA, posSideB);
    MoveList.add(move);

```

```

    //3 SIDE MOVE
    //checks if the next position in posC and posSideC are empty.
    byte posCC = getNextPositionInLine(posA, posB);

```

```

        if(posC!= VposB)
            continue;
        byte posSideC = getNextPositionInLine(posSideA, posSideB);
        if(posSideC==-9)
            continue;

        //checks if both posC and sideC are empty
        if(GetValueInPosition(posCC) == 0 &&
        GetValueInPosition(posSideC)==0)
        {
            posSideB, posSideC);
            {
                {
                    }
                }
            }
        }
        return MoveList;
    }

    /**
     * program returns a byte arr of all positions next to the given position.
     * if position is not in board than it is -1.
     */
    /**
     * program creates a bute array of all positions next to a given position.
     * if a positions is not in board, its -1 in the array.
     * @param position - position to creates neighbors array
     * @return byte array of neighbors , -1 for each one is out of board.
     */
    public byte [] getNeighborsOfPossition(byte position)
    {
        byte pos;
        byte arr [] = new byte [6];
        for(byte i=0;i<6;i++)
        {
            //neighbor position
            pos = (byte) (position+pathArr[i]);
            if(isPositionInBoard(pos))
                arr[i] = pos;
            else
                arr[i] = -1;
        }

        return arr;
    }
}

```

```

/**
 * program gets a board instance and sets current (this) board the values of the given board.
 * @param b - board to clone
 */
public void cloneBoard(Board b)
{
    this.BlueSet = (BitSet) b.BlueSet.clone();
    this.RedSet = (BitSet) b.RedSet.clone();

    //to find a better way for edge of board
    this.EdgeOfBoard = (BitSet) b.EdgeOfBoard.clone();
    this.scoreBlue = b.scoreBlue;
    this.scoreRed = b.scoreRed;
    //initpathArr();
}

```

}


```

package AbaloneGame;

public class Move
{
    //number of own balls to move.
    private byte numOfOwn;

    // number of enemy balls to push
    private byte numToPush;

    // is point move
    private boolean isScore;

    //indexes array.
    private byte [] indexs;

    //isrowMove
    private boolean isRowMove;

    /**
     * Constructor.
     */
    public Move()
    {

    }

    /**getters and setters***/

    public byte [] getIndexs()
    {
        return indexs;
    }

    public void setIndexs(byte[] indexs)
    {
        this.indexs = indexs;
    }

    public boolean isRowMove()
    {
        return isRowMove;
    }
}

```

```
}
```

```
public void setRowMove(boolean isRowMove)
{
    this.isRowMove = isRowMove;
}
```

```
public void setNumOfOwn(byte numOfOwn)
{
    this.numOfOwn = numOfOwn;
}
```

```
public void setNumToPush(byte numToPush)
{
    this.numToPush = numToPush;
}
```

```
public void setScore(boolean isScore)
{
    this.isScore = isScore;
}
```

```
public byte getNumOfOwn() {
    return numOfOwn;
}
```

```
public byte getNumToPush() {
    return numToPush;
}
```

```
public boolean isScore() {
    return isScore;
}
```

```
/*******functions*****
```

```

/**
 * program sends to server end point the updated board from this move.
 * @param point - endpoint to send move to
 * @param board - board to send indexes update.
 */
public void sendMoveToClient(serverEndPoint point, Board board)
{
    for(int i=0;i<this.indexs.length;i++)
    {
        point.sendmessage(indexs[i], board.GetValueInPosition(indexs[i]));
    }
    if(this.isScore)
    {
        int index =0;
        //always indexes[i] is own ball.
        if(board.GetValueInPosition(this.indexs[1]) == 1)
        { //player 1 score
            index = -9 -board.getScoreBlue();
            point.sendmessage(index, -3);
        }
        else
        { // player -1 score
            index = -19 -board.getScoreRed();
            point.sendmessage(index, 3);
        }
    }
}

////////////////////////////////create moves////////////////////////////////
/**
 * program updates the current class with those parameters.
 * @param posA - first position, own
 * @param posB - second position, empty
 */
public void new1BallMove(byte posA, byte posB)
{
    //System.out.println("Move.new1BallMove()");
    //initialize array
    indexs = new byte [2];

    //put positions in array
    indexs[0] = posA;
    indexs[1] = posB;

    //put other parameters to class
    numToPush = 0;
    numOfOwn = 1;
    isRowMove = false; ///maybe to change
    isScore = false;
}

```

```

/**
 * program updates the current class with those parameters.
 * @param posA - first position, own
 * @param posB - second position, own
 * @param posSideA - first side position, empty
 * @param posSideB - second side position, empty
 */
public void new2BallsSideMove(byte posA, byte posB, byte posSideA, byte posSideB)
{
    //System.out.println("Move.new2BallsSideMove()");
    //initialize array
    indexs = new byte [4];

    //put positions in array
    indexs[0] = posA;
    indexs[1] = posB;
    indexs[2] = posSideA;
    indexs[3] = posSideB;

    //put other parameters to class
    numToPush = 0;
    numOfOwn = 2;
    isRowMove = false;
    isScore = false;
}

```

```

/**
 * program updates the current class with those parameters.
 * @param posA - first position, own
 * @param posB - second position, own
 * @param posC - third position, own
 * @param posSideA - first side position, empty
 * @param posSideB - second side position , empty
 * @param posSideC - third side position , empty
 */
public void new3BallsSideMove(byte posA, byte posB, byte posC, byte posSideA,
    byte posSideB, byte posSideC)
{
    //System.out.println("Move.new3BallsSideMove()");
    //initialize array
    indexs = new byte [6];

    //put positions in array
    indexs[0] = posA;
    indexs[1] = posB;
    indexs[2] = posC;
    indexs[3] = posSideA;
    indexs[4] = posSideB;
    indexs[5] = posSideC;
}

```

```

        //put other parameters to class
        numToPush = 0;
        numOfOwn = 3;
        isRowMove = false;
        isScore = false;
    }

```

```

/**
 * program updates the current class with those parameters.
 * @param posA - first position , own
 * @param posB -second position , own
 * @param posC- third position , empty
 */
public void new2BallsMove0Push(byte posA, byte posB, byte posC)
{
    //System.out.println("Move.new2BallsMove0Push()");
    //initialize array
    indexs = new byte [3];

    //put positions in array
    indexs[0] = posA;
    indexs[1] = posB;
    indexs[2] = posC;

    //put other parameters to class
    numToPush = 0;
    numOfOwn = 2;
    isRowMove = true;
    isScore = false;
}

```

```

/**
 * program updates the current class with those parameters.
 * @param posA - first position, own
 * @param posB - second position, own
 * @param posC - third position, enemy
 * @param posD - forth position, empty
 */
public void new2BallsMove1PushNoScore(byte posA, byte posB, byte posC, byte posD)
{
    //System.out.println("Move.new2BallsMove1PushNoScore()");
    //initialize array
    indexs = new byte [4];

    //put positions in array
    indexs[0] = posA;
    indexs[1] = posB;
    indexs[2] = posC;
}

```

```

        indexs[3] = posD;

        //put other parameters to class
        numToPush = 1;
        numOfOwn = 2;
        isRowMove = true;
        isScore = false;
    }

/**
 * program updates the current class with those parameters.
 * @param posA - first position, own
 * @param posB - second position, own
 * @param posC - third position, enemy
 */
public void new2BallsMove1PushWithScore(byte posA, byte posB, byte posC)
{
    //System.out.println("Move.new2BallsMove1PushWithScore()");
    //initialize array
    indexs = new byte [3];

    //put positions in array
    indexs[0] = posA;
    indexs[1] = posB;
    indexs[2] = posC;

    //put other parameters to class
    numToPush = 0;
    numOfOwn = 2;
    isRowMove = true;
    isScore = true;
}

/**
 * program updates the current class with those parameters.
 * @param posA - first position, own
 * @param posB - second position, own
 * @param posC - third position, own
 * @param posD - forth position, empty
 */
public void new3BallsMove0Push(byte posA, byte posB, byte posC, byte posD)
{
    //System.out.println("Move.new3BallsMove0Push()");
    //initialize array
    indexs = new byte [4];

    //put positions in array
    indexs[0] = posA;
    indexs[1] = posB;

```

```

        indexs[2] = posC;
        indexs[3] = posD;

        //put other parameters to class
        numToPush = 0;
        numOfOwn = 3;
        isRowMove = true;
        isScore = false;
    }

```

```

/**
 * program updates the current class with those parameters.
 * @param posA - first position, own
 * @param posB - second position, own
 * @param posC - third position, own
 * @param posD - forth position, enemy
 * @param posE - fifth position, empty
 */
public void new3BallsMove1PushNoScore(byte posA, byte posB, byte posC, byte posD, byte posE)
{
    //System.out.println("Move.new3BallsMove1PushNoScore()");
    //initialize array
    indexs = new byte [5];

    //put positions in array
    indexs[0] = posA;
    indexs[1] = posB;
    indexs[2] = posC;
    indexs[3] = posD;
    indexs[4] = posE;

    //put other parameters to class
    numToPush = 1;
    numOfOwn = 3;
    isRowMove = true;
    isScore = false;
}

```

```

/**
 * program updates the current class with those parameters.
 * @param posA - first position, own
 * @param posB - second position, own
 * @param posC - third position, own
 * @param posD - forth position, enemy
 */

```

```

public void new3BallsMove1PushWithScore(byte posA, byte posB, byte posC, byte posD)
{
    //System.out.println("Move.new3BallsMove1PushWithScore()");
    //initialize array
    indexs = new byte [4];

    //put positions in array
    indexs[0] = posA;
    indexs[1] = posB;
    indexs[2] = posC;
    indexs[3] = posD;

    //put other parameters to class
    numToPush = 0;
    numOfOwn = 3;
    isRowMove = true;
    isScore = true;
}

```

```

/**
 * program updates the current class with those parameters.
 * @param posA - first position, own
 * @param posB - second position, own
 * @param posC - third position, own
 * @param posD - forth position, enemy
 * @param posE - fifth position, enemy
 * @param posF - sixth position, empty
 */

```

```

posF) public void new3BallsMove2PushNoScore(byte posA, byte posB, byte posC, byte posD, byte posE, byte
{
    //System.out.println("Move.new3BallsMove2PushNoScore()");
    //initialize array
    indexs = new byte [6];

    //put positions in array
    indexs[0] = posA;
    indexs[1] = posB;
    indexs[2] = posC;
    indexs[3] = posD;
    indexs[4] = posE;
    indexs[5] = posF;

    //put other parameters to class
    numToPush = 2;
    numOfOwn = 3;
    isRowMove = true;
    isScore = false;
}

```



```

/**
 * program updates the current class with those parameters.
 * @param posA - first position, own
 * @param posB - second position, own
 * @param posC - third position, own
 * @param posD - forth position, enemy
 * @param posE - fifth position, enemy
 */
public void new3BallsMove2PushWithScore(byte posA, byte posB, byte posC, byte posD, byte posE)
{
    //System.out.println("Move.new3BallsMove2PushWithScore()");
    //initialize array
    indexs = new byte [5];

    //put positions in array
    indexs[0] = posA;
    indexs[1] = posB;
    indexs[2] = posC;
    indexs[3] = posD;
    indexs[4] = posE;

    //put other parameters to class
    numToPush = 1;
    numOfOwn = 3;
    isRowMove = true;
    isScore = true;
}

```

}

```

package AbaloneGame;

public class PlayerTurnManager
{
    //mini datastruct.
    private byte [] pressedbuttons;

    private serverEndPoint endpoint;

    private Board board;

    public PlayerTurnManager(serverEndPoint point, Board b)
    {
        resetMiniDataStruct();
        this.endpoint = point;
        this.board = b;
    }

    /**
     * program initializes pressedbuttons array.
     */
    private void resetMiniDataStruct()
    {
        pressedbuttons = new byte [6];
        pressedbuttons[0] = -1;
        pressedbuttons[1] = -1;
        pressedbuttons[2] = -1;
        pressedbuttons[3] = -1;
        pressedbuttons[4] = -1;
        pressedbuttons[5] = -1;
    }

    /**
     * program receives client press index and checks if click is:
     * first click / second click (rowcheck) / third click(rowcheck) / direction click (side or push
check)
     * if it is the start of turn then pressedbuttons[0] =[1]= [2] = -1;
     * @param index - pressed button index
     * @param currentplayer - who pressed.
     * @return Move if end of a valid turn , null if not.
     */
    public Move recivedPress(byte index, int currentplayer)
    {
        System.out.println("reacehd recive press.");
        //first press
        if(pressedbuttons[0] == -1)
        {
            //own ball press
            if(board.GetValueInPosition(index)== currentplayer)

```

```

    {
        pressedbuttons[0] = index;
        endpoint.sendmessage(index, currentplayer*2);
        //finished click and in middle of turn.
        return null;
    }
}
return CaseNotFirstPress(index, currentplayer);
}

```

```

/**
 * program checks if it is a false press, second /third/ 4th press and if it is the last
press in turn.
 * @param index - pressed button index
 * @param currentplayer - who pressed.
 * @return Move if end of a valid turn , null if not.
 */
private Move CaseNotFirstPress(byte index, int currentplayer)
{
    int CurrentPressValue = board.GetValueInPosition(index);

    if(CurrentPressValue == currentplayer)
    { //this is own ball press, 2 3 or 4 press

        //case 2nd press
        if(pressedbuttons[1] == -1)
        { //this is the second press.

            //checks if the 2nd position is near first press.
            if(board.IsPositionsTogether(pressedbuttons[0],index))
            {
                //legal 2nd press in line same own color.
                pressedbuttons[1] = index;
                endpoint.sendmessage(index, currentplayer*2);
                return null;
            }
            else
            {
                CanceTurn(currentplayer); // not close both presses.
                return null;
            }
        }
        else
        { //can be 3rd press with same color
            //checks if positions are in line and it is the 3rd press.
            if(pressedbuttons[2] == -1 &&
board.isPositionsInLine3(pressedbuttons[0],
                pressedbuttons[1],index))
            {
                //legal 3rd press in line same own color.
                pressedbuttons[2] = index;

```

```

        endpoint.sendMessage(index, currentplayer*2);
        return null;
    }
    else
    {
        //4 presses with same color
        CanceTurn(currentplayer);
        return null;
    }
}
}
else
{
    // not own ball pressed in 2/3/4 press.

    //ball pressed (not first turn) is not own color
    return CaseLastPressInTurn(index, currentplayer);
}
}

/**
 * program checks if the last press is part of row/side move. and if it is a valid move.
 * can be 2nd/3rd/4th press.
 * @param index - pressed button index
 * @param currentplayer - who pressed.
 * @return Move if end of a valid turn , null if not.
 */
private Move CaseLastPressInTurn(byte index, int currentplayer)
{
    //Move to fillin and return.
    Move m = null;
    //only one own ball pressed
    if(pressedbuttons[1] == -1)
    {
        //check if close to each other and target position empty
        if(board.IsPositionsTogether(pressedbuttons[0],index) &&
            board.GetValueInPosition(index)==0)
        {
            m = new Move();
            m.new1BallMove(pressedbuttons[0], index);
            emptyArr();
            System.out.println("1ballmove");
            return m;
        }
        else
        {
            CanceTurn(currentplayer);
            return null;
        }
    }

    //if reached then only more then one ball already pressed

```

```

if(pressedbuttons[2]==-1)
{
    //only two own balls pressed
    //System.out.println("reached 22222");

    if(board.isPositionsInLine3(pressedbuttons[0], pressedbuttons[1], index))
    {
        // case front summo

        m = board.TryToPush2(pressedbuttons[0], pressedbuttons[1], index);

        //if null than not a valid push.
        if(m==null)
        {
            CanceledTurn(currentplayer);
            return null;
        }

        //is a valid move.
        emptyArr();
        return m;
    }

    ///if reached than possible side move. try to side move 2 balls.
    m = board.TryToSideMove2(pressedbuttons[0],pressedbuttons[1], index);
    if(m==null)
    {
        CanceledTurn(currentplayer);
        return null;
    }
    emptyArr();
    return m;
}

```

```

///if reached than three own balls pressed
if(board.isPositionsInLine3(pressedbuttons[1],pressedbuttons[2],index))
{
    //try to summo 3.

```

```

        m = board.TryToPush3( pressedbuttons[0], pressedbuttons[1],
pressedbuttons[2],index);
        if(m==null)
        {
            CanceledTurn(currentplayer);
            return null;
        }

        emptyArr();
        return m;
    }
    else
    {
        ///possible side move. try to side move 3 balls.

        m = board.TryToSideMove3(pressedbuttons[0], pressedbuttons[1],
pressedbuttons[2], index);
        if(m==null)

```

```

        {
            CanceTurn(currentplayer);
            return null;
        }
        emptyArr();
        return m;
    }
}

/**
 * program resets the pressedbuttons array.
 * program sends messages to client to reset pressed colors there too.
 * @param currentplayer - current player.
 */
private void CanceTurn(int currentplayer)
{
    System.out.println("reached cancel turn");
    if(pressedbuttons[0] != -1)
    {
        //send message to cancel press
        endpoint.sendMessage(pressedbuttons[0] , currentplayer);
        //reset array
        pressedbuttons[0] = -1;
    }

    if(pressedbuttons[1] != -1)
    {
        //send message to cancel press
        endpoint.sendMessage(pressedbuttons[1], currentplayer);
        //reset array
        pressedbuttons[1] = -1;
    }

    if(pressedbuttons[2] != -1)
    {
        //send message to cancel press
        endpoint.sendMessage(pressedbuttons[2], currentplayer);
        //reset array
        pressedbuttons[2] = -1;
    }
}

/**
 * program resets pressedbuttons array. (usually after a valid move has found)
 */
private void emptyArr()
{
    System.out.println("reached empty arr");
    //reset array
    pressedbuttons[0] = -1;
    pressedbuttons[1] = -1;
    pressedbuttons[2] = -1;
}
}

```

```

package AbaloneGame;

import java.util.BitSet;
import java.util.ArrayList;

import org.apache.tomcat.jni.Time;

public class AIManager
{

    private BitSet [] Distances;

    // variables.
    private final int ValueOfGrouping = 7;
    private final int ValueOfCenterDistance = 10;
    private final int ValueOfScoredBalls = 1000;
    //Alpha Beta recursion depth
    private final int depthToSearch = 4;

    public AIManager(int player)
    {
        initializeDistances();
    }

    /**
     * This is the program that activates main algorithm.
     * program uses AlphaBeta and Evaluate function to find the best move of computer.
     * @param board - current board to find move to.
     * @param currentplayer - player to find move to.
     * @return - best move it can choose.
     */
    public Move playTurn(Board board, int currentplayer)
    {
        System.out.println("started AI turn!!!");
        //dataStruct of possible moves
        ArrayList<Move> ogBoardMoves = board.getmoves((byte) currentplayer);
        System.out.println("-----found " + ogBoardMoves.size() + "board possible moves");
        // implementing the moves to boards.
        ArrayList<Board> AllBoards = getPossibleBoards(board, ogBoardMoves);
        // starts as worst possible move

        // - infinity
        int bestValue = -1* currentplayer *2000000;
    }

```

```

        int tempvalue, bestindex = 0; // cnt=0;
        for (Board childBoard : AllBoards)
        {
            tempvalue = AlphaBeta(childBoard, depthToSearch, -2100000, 2100000, (byte)
(currentplayer * -1));
            //searches for the minimum value by blues point of view. (red is AI).
            if(tempvalue < bestValue)
            {
                bestValue = tempvalue;
                bestindex = AllBoards.indexOf(childBoard);
            }
        }

        return ogBoardMoves.get(bestindex);
    }
    //int cnt =0;
    //System.out.println("----- calculated boards: " + cnt);

    /**
     * This is the main AI recursion algorithm.
     * uses EvaluateBoard to return the best result it can promise.
     * if the program hasn't reached depth 1 than it calculates all possible board and calles
     * again to the function with depth-1 and currentPlayer*-1.
     * @param board - board to calculate from.
     * @param depth - current depth to search. if depth =1 than calculates board value and return it.
     * @param alpha - best board value Alpha player can promise
     * @param beta - best board value Beta player can promise
     * @param currentPlayer - player to calculate to
     * @return
     */
    private int AlphaBeta(Board board, int depth,int alpha,int beta, byte currentPlayer)
    {
        //cnt++;
        //System.out.println("started alpha betha");
        // checks if needs to return the value of board.
        if(depth <=1 || board.getScoreBlue()>=6 || board.getScoreRed()>=6)
            return EvaluateBoard(board);

        // if 1 than try to maximize board value.
        if(currentPlayer == 1)
        {
            // worst value, cannot be under -1000000.
            int value = -2000000;
            //dataStruct of possible moves
            ArrayList<Move> allBoardMoves = board.getmoves((byte) currentPlayer);
            ArrayList<Board> AllBoards = getPossibleBoards(board, allBoardMoves);

            for (Board childBoard : AllBoards)
            {
                //recursion.
                value = Math.max(value, AlphaBeta(childBoard, depth -1, alpha, beta,(byte)-1));

                alpha = Math.max(value, alpha);
            }
        }
    }

```



```

        //pruning
        if(alpha >= beta)
        {
            //System.out.println("pruning accured");
            break;
        }
    }
    return value;
}
else
{
    // try to minimize board value.

    // worst value, cannot be more than 1000000.
    int value = 2000000;
    //dataStruct of possible moves
    ArrayList<Move> allBoardMoves = board.getmoves((byte) currentPlayer);
    ArrayList<Board> AllBoards = getPossibleBoards(board, allBoardMoves);

    for (Board childBoard : AllBoards)
    {
        //recursion.
        value = Math.min(value, AlphaBeta(childBoard, depth -1, alpha, beta,(byte)1));

        beta = Math.min(value, alpha);
        //pruning
        if(alpha >= beta)
        {
            //System.out.println("pruning accured");
            break;
        }
    }
    return value;
}
}
}

```

```

/**
 * program return list of all possible boards which have implemented the moves from
 * original board and ogBoardMoves list.
 * @param board -board to calculate all its child boards.
 * @param ogBoardMoves - list of all possible moves from originalboard - board.
 * @return list of all possible child boards of board.
 */
private ArrayList<Board> getPossibleBoards(Board board, ArrayList<Move> ogBoardMoves)
{
    //dataStruct of boards to return
    ArrayList<Board> PossibleBoards = new ArrayList<Board>(60);
    Board tempBoard;
    int NumberOfMoves = ogBoardMoves.size();

```

```

    //goes over each Move
    for(int i=0;i<NumberOfMoves;i++)
    {
        tempBoard = new Board();
        tempBoard.cloneBoard(board);
        tempBoard.makeMove(ogBoardMoves.get(i));
        PossibleBoards.add(tempBoard);
    }
    return PossibleBoards;
}

```

```

/**
 * This is the main function of the algorithm.
 * program gets a board and calculates its entire value.
 * the program calculates by the blue player perspective.
 *
 * the program calculates the distances of the each player balls from center,
 * the grouping of each players balls and the current score.
 * the program returns (value of blue)-(value of red) so it calculates the entire board from
 * blues perspective.
 * @param board - board to calculate
 * @return - board value from blues perspective.
 */
private int EvaluateBoard(Board board)
{
    int sum = 0;

    // distances from center.
    sum += CalculateBoardDistances(board, (byte) 1);
    sum -= CalculateBoardDistances(board, (byte) -1);
    //balls grouping
    sum+= CalculateGrouping(board, (byte) 1);
    sum-= CalculateGrouping(board, (byte) -1);
    //current score.
    sum+= CalculateScore(board,(byte) 1);
    sum-= CalculateScore(board,(byte) -1);

    return sum;
}

```

```

/**
 * program gets move and player, returns value for the grouping of player.
 * @param board - board to calculate
 * @param player - which player to calculate.
 * @return count of grouping*weight (value).
 */
private int CalculateGrouping(Board board, byte player)
{
    int cnt=0;

```

```

    BitSet a = (player ==1)? board.getBlueSet() : board.getRedSet();
    //goes over each ball and count all neighbours.
    for (int i = a.nextSetBit(0); i != -1; i = a.nextSetBit(i+1))
    {
        for(byte pos :board.getNeighborsOfPosition((byte)i))
        {
            if(pos == -1)
                continue;

            if(a.get(pos))
                cnt++;
        }
    }

    int sum = CalculateBoardGroupingValue(cnt);
    return sum;
}

/**
 * rogram gets the counter of grouping and return a value of the grouping.
 * @param cnt - neighbors counter.
 * @return cnt*ValueOfGrouping
 */
private int CalculateBoardGroupingValue(int cnt)
{
    return cnt*ValueOfGrouping;
}

/**
 * program calculates the value of scored balls.
 * @param b - board.
 * @param player - player to calculate
 * @return weight*amount of scored balls.
 */
private int CalculateScore(Board b, byte player)
{
    if(player == 1)
    {
        //win
        if( b.getScoreBlue() >= 6)
            return 1000000;
        return b.getScoreBlue()*ValueOfScoredBalls;
    }
    if( b.getScoreRed() >= 6)
        return 1000000;
    return b.getScoreRed()*ValueOfScoredBalls;
}

```

```

/**
 * program calculates the total board distances of given player and
 * returns the value * weight.
 * the program uses 5 bitsets as distances mask.
 * @param board - board
 * @param player
 * @return total value of distances from center * its weight.
 */
private int CalculateBoardDistances(Board board, byte player)
{
    BitSet a = (player == 1)? board.getBlueSet() : board.getRedSet();
    int sum=0;
    for (int i = a.nextSetBit(0); i != -1; i = a.nextSetBit(i+1))
    {
        //if in circle 0.
        if(Distances[0].get(i))
        {
            sum+=0;
            continue;
        }
        //if in circle 0.
        if(Distances[1].get(i))
        {
            sum+=1;
            continue;
        }
        //if in circle 0.
        if(Distances[2].get(i))
        {
            sum+=2.8;
            continue;
        }
        //if in circle 0.
        if(Distances[3].get(i))
        {
            sum+=4;
            continue;
        }
        //if in circle 0.
        if(Distances[4].get(i))
        {
            sum+=5;
            continue;
        }
        System.out.println("couldnt find position");
    }

    int result = calculateValueOfSumDistances(sum);
    return result;
}

```

```

/*
 * first version
 * program gets the counter of distance to center
 * program returns the value of the
 */
/**
 * program gets the sum of distances and rethrn the sum*its weight.
 * the program returns the value * -1 so the players will try to get closer to the center and not
away from it.
 * @param sum - sum of distances from center.
 * @return - sum*ValueOfCenterDistance*-1
 */
private int calculateValueOfSumDistances(int sum)
{
    return sum*ValueOfCenterDistance*-1;
}

/**
 * program initializes and sets 5 bitsets value which help calculate each ball's distance from the
center.
 */
private void initializeDistances()
{
    Distances = new BitSet [5];
    Distances[0] = new BitSet(109);
    Distances[1] = new BitSet(109);
    Distances[2] = new BitSet(109);
    Distances[3] = new BitSet(109);
    Distances[4] = new BitSet(109);

    //only middle
    Distances[0].set(60);

    //first circle
    Distances[1].set(48);
    Distances[1].set(49);
    Distances[1].set(61);
    Distances[1].set(59);
    Distances[1].set(71);
    Distances[1].set(72);

    //second circle
    Distances[2].set(36, 39, true);
    Distances[2].set(50);
    Distances[2].set(62);
    Distances[2].set(73);
    Distances[2].set(82,85, true );
    Distances[2].set(70);
    Distances[2].set(58);

```

```

Distances[2].set(47);

//third circle
Distances[3].set(24,28, true);
Distances[3].set(39);
Distances[3].set(51);
Distances[3].set(63);
Distances[3].set(74);
Distances[3].set(85);
Distances[3].set(93, 97, true);
Distances[3].set(81);
Distances[3].set(69);
Distances[3].set(57);
Distances[3].set(46);
Distances[3].set(35);

//forth circle
Distances[4].set(23);
Distances[4].set(34);
Distances[4].set(45);
Distances[4].set(56);
Distances[4].set(12, 17);
Distances[4].set(28);
Distances[4].set(40);
Distances[4].set(52);
Distances[4].set(64);
Distances[4].set(75);
Distances[4].set(86);
Distances[4].set(97);
Distances[4].set(104, 109, true);
Distances[4].set(92);
Distances[4].set(80);
Distances[4].set(68);
Distances[4].set(56);

```

```

}

```

```

}

```