



TECHNION - ISRAEL INSTITUTE OF TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE

Geometric Image Processing Laboratory

דו"ח פרויקט

מימוש משחק איקס-עיגול מגע

מגישים:

חסיין עותמאן

עבדל מסיח מבאריקי

מנחה:

רג'א ג'יריס

11.08.2013

תוכן העניינים

3.....	1. תקציר	3
4.....	2. מבוא	4
5.....	3. החלק הראשון	5
6.....	3.1 עבודה עם <i>Microsoft DirectX</i>	6
7.....	3.2 דוגמא של פלט	7
8.....	4. החלק השני	8
8.....	4.1 הקלט	8
9.....	4.2 שלבי הביצוע	9
10.....	4.3 מציאת המישור שבו נמצא לוח המשחק ביחס ל <i>Kinect</i>	10
10.....	4.3.1 חיפוש תמונה קטנה בתוך תמונה גדולה	10
13.....	4.3.2 מיפוי הנקודות של תמונת ה <i>RGB</i> לתמונת ה <i>depth</i>	13
14.....	4.3.3 מיפוי הנקודות של תמונת ה <i>depth</i> למרחב של השלד (<i>skeleton</i>)	14
15.....	4.3.4 מציאת המישור הקרוב ביותר	15
17.....	4.3.5 מציאת מרחק נקודה ממישור הלוח	17
18.....	4.4 מציאת המיקום של היד ביחס ללוח המשחק	18
19.....	4.5 תמונות הדגמה	19
20.....	5. סיכום	20
21.....	6. ביבליוגרפיה	21

1. תקציר

מטרת הפרויקט היא מימוש משחק איקס עיגול מגע בשימוש ב
Microsoft DirectX ומקרן ומצלמת *Kinect*.

משחק איקס עיגול הינו משחק קלאסי לשני משתתפים שבו נתון לוח בגודל
שלוש על שלוש. ומשתמש אחד מסמן X והשני מסמן O לסירוגין. מטרת כל
משתמש היא ליצור רצף והראשון שמצליח ליצור רצף מנצח.

מטרת פרוייקט זה הינה לממש גרסא של משחק זה על לוח מגע ללא מסך
מגע. מקרן יקרין את המשחק על לוח לבן ואז המשתתפים יסמנו את המיקום
שבו הם רוצים לשים את סימונם. זיהוי התנועה והסימונים נעשה בעזרת
מצלמת *Kinect*.

מטרת הפרויקט : מימוש משחק איקס-עיגול מגע.

לוח המשחק מוקרן על לוח בעזרת מקרן. העבודה עם המקרן ובניית המשחק נעשית ע"י מנשק *Microsoft DirectX*.

זיהוי התנועה של השחקנים נעשה ע"י מצלמת *Kinect*. דבר זה מאפשר הדמיה של מסך מגע עבור השחקנים בלי קיומו של מסך מגע אמיתי.

Microsoft DirectX:

ספרייה שפותחה ע"י *Microsoft* מנגישה פונקציות לגרפיקה דו-ממדית ותלת-ממדית. הוספת ספרייה זו גרמה לכך שפיתוח משחקי מחשב יהיה יותר קל ומהיר מאשר קודם.

Kinect:

בקר משחקים שמיוצר ע"י *Microsoft*. מכיל מצלמת עומק. מאפשר זיהוי גוף האדם ובונה לו שלד דמיוני ומדויק.

הפרויקט כולל שני חלקים. נעשה שימוש במנשק *Microsoft DirectX* לשם הצגת הפלט.
חלקי הפרויקט:

החלק הראשון:

הקלט מתקבל מהעכבר והפלט מוצג למסך רגיל.

החלק השני:

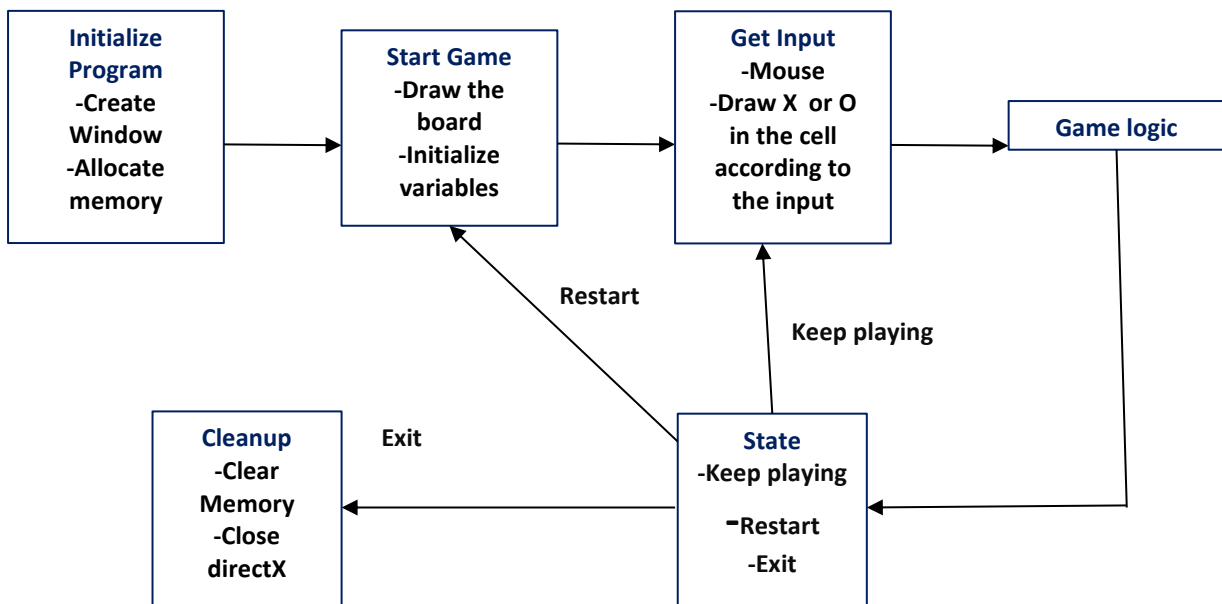
הקלט מתקבל ע"י מצלמת *Kinect* אשר מזהה סימון ידני של השחקנים על לוח והפלט מוצג ע"י מקרן.

3. תיאור החלק הראשון

בחלק הראשון המטרה הייתה לממש משחק איקס-עיגול עם קלט מעכבר ולכן השתמשנו ב *Microsoft DirectX*.

החלק הראשון של הפרויקט כלל שלושה שלבים עיקריים:

1. ייצור חלון המשחק ב *Microsoft DirectX* במסך מלא.
2. המתנה בלולאה אינסופית לקלט מהעכבר עד לקבלת הודעת סיום המשחק.
3. סימון X או O בהתאם לקלט.



תרשים 1: תרשים זרימה לחלק הראשון בפרוייקט.

3.1 עבודה עם Microsoft DirectX

- הפונקציה הראשית:

- `int WINAPI wWinMain(HINSTANCE hInst, HINSTANCE, LPWSTR, INT nCmdShow)`

פונקציה זו כוללת את פתיחת החלון והמתנה להודעות החלון.

- פונקציית פתיחת החלון:

`CreateWindowW`

- פונקציית טיפול בהודעות:

- `LRESULT WINAPI MsgProc(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam)`

בפונקציה זו קבלנו את המיקום החדש של העכבר בתוך החלון.

הפונקציה `void PutPixel(int x, int y)` מקבלת את הפיקסל שרוצים לצבוע בלבן וצובעת בלבן באופן הבא:

```
((D3DCOLOR *)rect.pBits)[x + (rect.Pitch >> 2) * y]
= D3DCOLOR_XRGB(255,255,255);
```

- ציר הקווים:

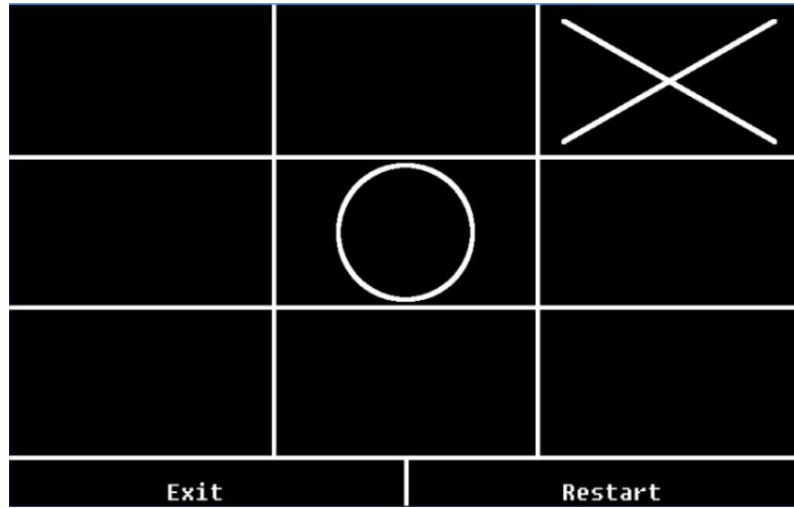
לכל קו קבענו נקודת התחלה וסיום ועל סמך נקודות אלה חשבנו את משוואתו ואז עברנו על כל הנקודות (עם x, y שלמים) מנקודת ההתחלה לנקודת הסיום וקבענו את הצבע של פיקסלים אלה ללבן.

```
DrawLine:(x1,y1),(x2,y2)
dx=x2-x1,dy=y2-y1
if(|dy|>|dx|)
{
    if(y1>y2)
    {
        replace between y1 and y2,x1 and x2
    }
    m=dx/dy;
    b=x1-m*y1;
    for y=y1;y<=y2;y++
        x=m*y+b;
        PutPixel(x,y);
}
else
{
    if(x1>x2)
    {
        replace between x1,x2 and y1,y2;
    }

    m=dy/dx;
    b=y1-m*x1;
    for x=x1;x<=x2;x++
        y=m*x+b;
```

```
} PutPixel(x,y);
```

3.2 דוגמא של פלט

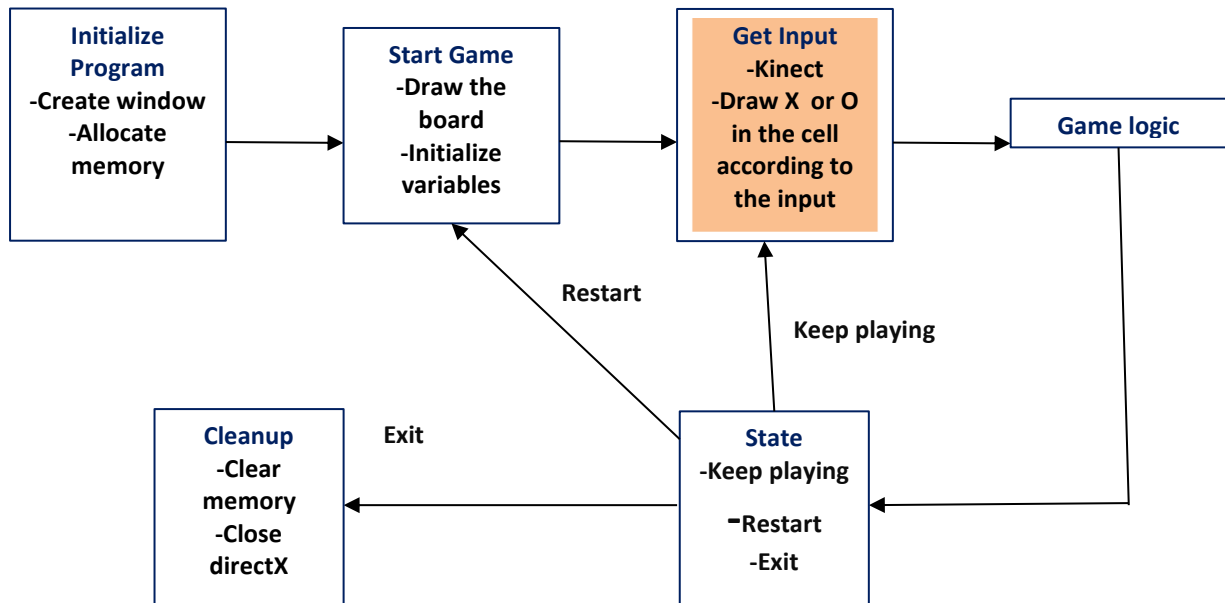


תמונה 1: הדגמת המשחק.

4. תיאור החלק השני

4.1 הקלט בחלק השני

השינוי בין החלק הראשון והשני הוא שהקלט בחלק השני ממצלמת *Kinect*, אנו מקבלים את מיקום יד השחקן ביחס ל *Kinect*, ומיקום זה מייצג את המשבצת הרצויה לסימון.



תרשים 2: תרשים זרימה לחלק השני בפרוייקט.

4.2 תיאור שלבי ביצוע החלק השני בפרויקט

1. הקרנת תמונה ללוח המשחק ששומרים אצלנו , תמונה זו תשמש אותנו לשם זיהוי מיקום הלוח ביחס ל *Kinect*.



תמונה 2: קליברציה

2. קבלת תמונה *RGB* מה *Kinect* שכוללת את לוח המשחק (התמונה שהקרנו) עם הסביבה.
3. חיפוש התמונה שלנו בתוך תמונת ה *Kinect* וקבלת המיקום הד-ממדי ללוח המשחק ביחס ל *Kinect*.
4. קבלת תמונה *depth* מה *Kinect* שכוללת את לוח המשחק (התמונה שהקרנו) עם הסביבה.



תמונה 3 : *depth frame*

5. מיפוי הנקודות של תמונת ה *RGB* לתמונת ה *depth*.
6. מיפוי הנקודות של תמונת ה *depth* למרחב של השלד (*skeleton*)(נסמן נקודות אלו ב *P*).
7. מציאת המישור הקרוב ביותר שעובר בנקודות *P* ע"י אלגוריתם מבוסס *SVD*.
8. ביצוע לולאה אינסופית. בכל פעם מקבלים מיקום היד של השחקן ביחס ל *Kinect*.

4.3 מציאת המישור שבו נמצא לוח המשחק ביחס ל Kinect

אנו רוצים למצוא את המישור שבו נמצא לוח המשחק ביחס ל Kinect , לשם כך נעזר בארבעה שלבים:

- הקרנת תמונה מהמחשב וקבלת תמונה מה Kinect שבה נמצאת תמונה זו. ואז חיפוש את התמונה הקטנה שהקרנו בתוך התמונה הגדולה שקבלנו מ Kinect.
- מיפוי תמונת RGB לתמונת Depth.
- מיפוי תמונת Depth למישור השלד (Skeleton).
- מציאת המישור הקרוב ביותר שעובר בנקודות שקבלנו במישור השלד.

תיאור המימוש

4.3.1 חיפוש תמונה קטנה בתוך תמונה גדולה

להקטנת זמן הריצה הקטנו את גודל שתי התמונות פי 9 .

הקטנת גודל התמונות התבצע באופן הבא:

$$img = \begin{bmatrix} x_{11} & \cdots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nn} \end{bmatrix} - \text{נסמן את התמונה}$$

כל פיקסל מכיל שלושה ערכים (R, G, B) .

עוברים על המטריצה img בבלוקים בגודל 3×3 :

עבור כל בלוק כזה נחשב ערך שייצג בלוק זה במטריצה של התמונה החדשה.

ערך זה שווה לממוצע של הערכים של כל הפיקסלים בבלוק זה , כלומר :

$$block - value = (avg \ of \ R, \ avg \ of \ G, \ avg \ of \ B)$$

כאשר :

$$avg \ of \ R = \frac{1}{9} \cdot \sum_{i=1}^9 t_i \cdot R$$

(כאשר t_i מסמן פיקסל בבלוק זה).

באותו אופן מגדירים את avg of G , avg of B .

כל בלוק בגודל 3×3 מיוצג ע"י פיקסל בודד שהוא $block - value$ בתמונה החדשה, ולכן הקטנו את התמונה ב 9 פעמים.

לאחר שהקטנו את גודל התמונות בצענו נרמול לתמונות.

תיאור הנרמול:

עוברים על כל הפיקסלים בתמונה ומחשבים את גורם הנרמול שהוא סקלר ששווה ל:

$$normal_factor = \sqrt{\sum_{i=1}^{height} \sum_{j=1}^{width} x_{ij} \cdot r^2 \cdot x_{ij} \cdot g^2 \cdot x_{ij} \cdot b^2}$$

לאחר חישוב גורם הנרמול עוברים על הפיקסלים בתמונה ומחלקים אותם בגודל גורם הנרמול. כלומר עבור פיקסל i, j הערך החדש יהיה:

$$x_{ij} = \left(\frac{x_{ij} \cdot r}{normal_factor}, \frac{x_{ij} \cdot g}{normal_factor}, \frac{x_{ij} \cdot b}{normal_factor} \right)$$

מציאת התמונה הקטנה בתוך הגדולה:

אנו רוצים למצוא את מיקום התמונה הקטנה בתמונה הגדולה. לשם כך נרצה למצוא את הבלוק בתמונה הגדולה אשר קרוב ביותר לתמונה הקטנה. לשם כך, נרצה למצוא את הבלוק בתמונה הגדולה הקרוב ביותר מבחינת הפרש ריבועים לתמונה הקטנה. לשם כך נעזר בעובדה שאם השניים מנורמלים אז מזעור הפרש ריבועים שקול להבאה למקסימום של מכפלה פנימית.

בהתחלה מבצעים נרמול לתמונה הקטנה.

עוברים על המטריצה של התמונה הגדולה בבלוקים בגודל התמונה הקטנה. עבור כל בלוק כזה מבצעים נרמול כמו שתואר קודם.

נסמן את מטריצת התמונה הקטנה (לאחר הנרמול) ב- $small_img$ ואת הבלוק שעוברים עליו עכשיו מהתמונה הגדולה ב $block$.

מחשבים את הערך הבא:

$$mult = \sum_{i=1}^n \sum_{j=1}^m (small_img[i][j]) \cdot block[i][j]$$

כאשר הכפלת פיקסלים x, y שווה ל:

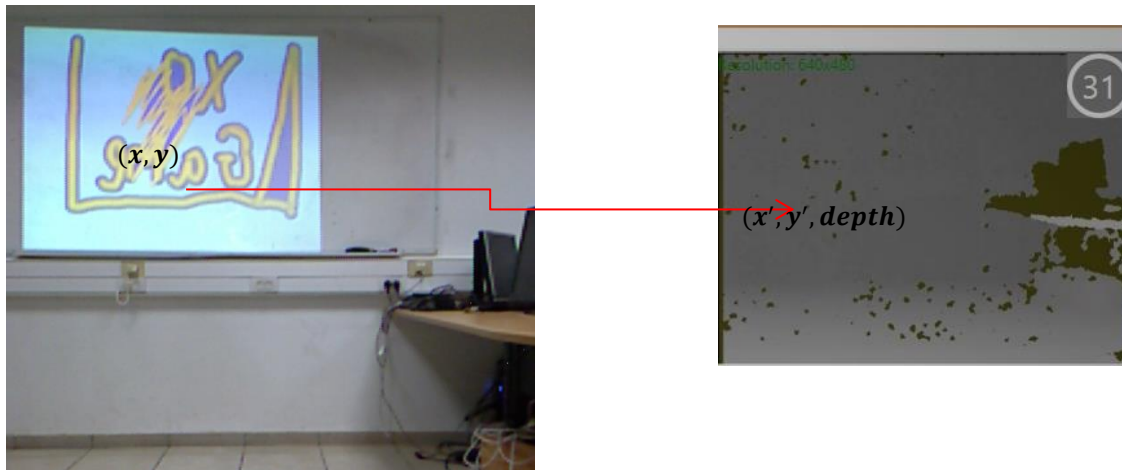
$$x * y = x.r \cdot y.r + x.g \cdot y.g + x.b \cdot y.b$$

הבלוק שמקבלים עבורו ערך $mult$ מקסימלי הוא הבלוק שמכיל את התמונה הקטנה שמחפשים אותה. ולכן נחזיר אותו.

4.3.2 מיפוי הנקודות של תמונת ה *RGB* לתמונת ה *depth*
עוברים על הנקודות של ה *depth frame* ומשתמשים בפונקציה

MapColorFrameToDepthFrame שממפה מ *depth frame* ל *RGB frame*.
בודקים את הנקודות שקבלנו, עבור כל נקודת RGB מחפשים את הנקודה ב *depth* שממופה אליה.

תמונה להמחשה:



תמונה 4: מיפוי מ *RGB Frame* ל *DepthFrame*

4.3.3 מיפוי הנקודות של תמונת ה *depth* למרחב של השלד (*skeleton*) עוברים על הנקודות שמייצגים את לוח המשחק ב *depth frame* . עבור כל נקודה כזו מקבלים את הנקודה המתאימה ב *skeleton plane* ע"י השימוש בפונקציה : *MapDepthPointToSkeletonPoint* .

נקודה ב *skeleton frame* מהצורה:

$$(x, y, z)$$

כאשר מייצג את מרחק הנקודה במישור התלת ממדי ביחס ל Kinect . מספר הנקודות שמקבלים הוא בגודל התמונה הקטנה (לאחר שהקטנו פי 9) שזה כמעט:

$$\frac{(320 * 240)}{9} \cong 8534$$

נשים נקודות אלו במטריצה ונסמן אותה ב *P* .

4.3.4 מציאת המישור הקרוב ביותר

המטרה שלנו היא למצוא את המישור הקרוב ביותר שעובר בנקודות שבמטריצה P שבנינו בתת הפרק הקודם. משיגים את זה בשימוש באלגוריתם מבוסס SVD .

יש לנו המטריצה :

$$P = \begin{pmatrix} p_1 \\ p_2 \\ \cdot \\ \cdot \\ \cdot \\ p_n \end{pmatrix}$$

כאשר כל נקודה p_i מיוצגת ע"י:

$$p_i = (x, y, z)$$

משתמשים באלגוריתם SVD למציאת המישור הקרוב ביותר שעובר בנקודות אלו.

השימוש ב SVD התבצע באופן הבא:

ראשית נחשב את מרכז הנקודות :

$$p_c = \frac{1}{n} \cdot \sum_{i=1}^n p_i$$

נבנה את המטריצה הבאה:

$$P_{SVD} = \begin{pmatrix} p_1 - p_c \\ p_2 - p_c \\ \vdots \\ p_n - p_c \end{pmatrix}$$

משתמשים באלגוריתם SVD לפירוק מטריצה :

$$P_{SVD} = UDV$$

הנורמל למישור שלנו הוא העמודה ב V שמתאימה לערך הסינגולרי הקטן ביותר ב P_{SVD} . כלומר הכי ימינית ב V שמתאימה לערך הסינגולרי האחרון שאינו אפס. העמודה הזו תיתן לנו את (a, b, c) ממשוואת המישור : $ax + by + cz + d = 0$. מציאת ערך d ע"י הצבת הנקודה p_c במשוואה שקבלנו.

4.3.5 מציאת מרחק נקודה ממישור הלוח

לאחר שמצאנו את המישור שבו נמצא הלוח ביחס ל *Kinect* , אנו נשתמש בו ע"מ לקבוע את מיקום היד של השחקן ביחס ללוח המשחק. למציאת אם היד של השחקן במישור הלוח אנו צריכים למצוא את מרחק היד שלו שמיוצגת ע"י נקודה ב *Kinect* מהמישור שקבלנו.

האלגוריתם מבוסס *SVD* נותן לנו וקטור נורמל מנורמל כלומר

$$\sqrt{a^2 + b^2 + c^2} \text{ שווה ל } 1 .$$

ולכן למציאת מרחק נקודה מהמישור מספק לחשב :

$$|ax + by + cz + d|$$

עוברים על כל הנקודות במטריצה *P* ומחשבים את המרחק המקסימלי של נקודות אלו מהמישור , נסמן מרחק זה ב *max_distance*.

כאשר מקבלים מיקום היד של השחקן , מחשבים את מרחק היד מהמישור . אם קטן או שווה ל *max_distance* אז מאפשרים סימון על הלוח במקום המתאים.

ההחלטה אם יש נגיעה בלוח ע"י כך שבודקים אם מרחק היד קטן או שווה ל *max_distance* . אחרי הנגיעה , השחקן צריך להרחיק את ידו מהלוח (יהיה מרחק ידו גדול מ *max_distance*) ע"מ שנזהה עוד נגיעה.

4.4 מציאת המיקום של היד ביחס ללוח המשחק

לאחר שמצאנו שהיד של השחקן נמצאת במישור הלוח וצריך לאפשר סימון, אנו צריכים למצוא את המיקום (x, y) של היד ביחס ללוח המשחק.

משיגים מטרה זו באופן הבא:

- מיפוי הנקודה שמייצגת את מיקום היד במישור השלד לנקודה שמייצגת אותה ב *color frame* ע"י הפונקציה *MapSkeletonPointToColorPoint*.
- מיפוי הנקודה שקבלנו ב *color frame* לנקודה המתאימה בחלון המשחק:
נסמן :

$(xColorFrame, yColorFrame)$

את הנקודה שקבלנו ב *color frame*.

ונסמן ב :

$(x0, y0)$

את הנקודה ב *color frame* שבה נמצאת הנקודה שמאלית עליונה של לוח המשחק ב *color frame* (מצאנו בשלב מציאת תמונת הלוח היחס ל *Kinect*).

ממפים מ *color frame* ללוח המשחק ע"י הנוסחה:

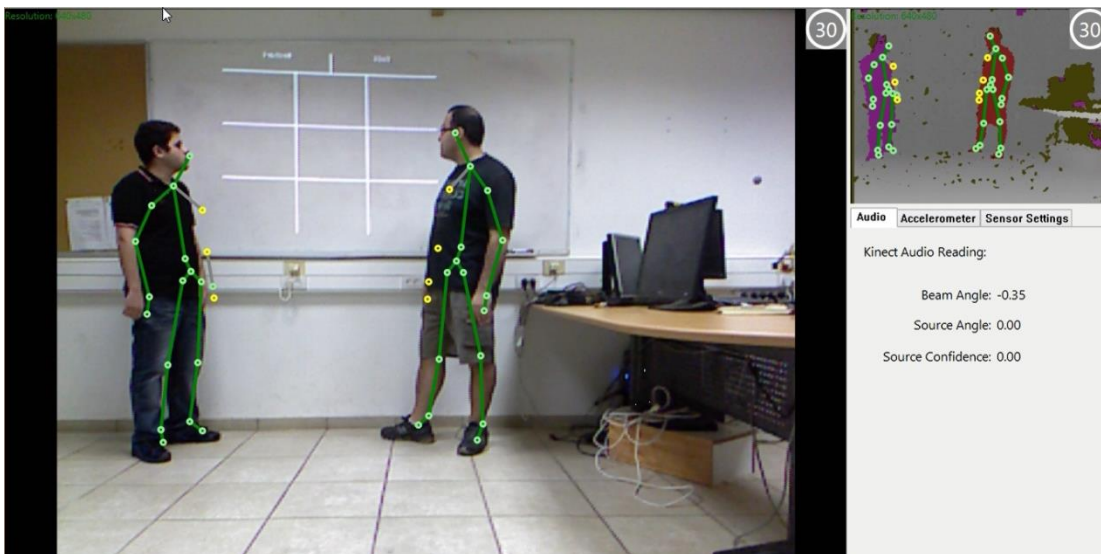
$$x_{board} = |xColorFrame - x0| \cdot (board_width/colorFrame_width);$$

$$y_{board} = |yColorFrame - y0| \cdot (board_height/colorFrame_height);$$

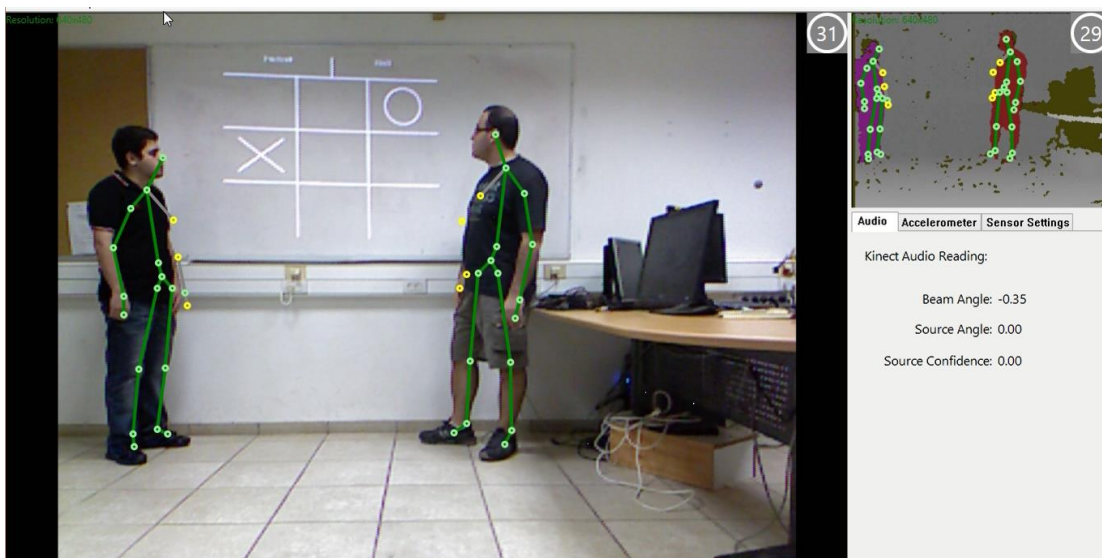
4.5 תמונות הדגמה

הערה:

הקווים הירוקים הם ה-skeleton אשר בונה ה-kinect עבור כל שחקן. מצד ימין למעלה אנו רואים את תמונת העומקים שמחושבת ע"י ה-Kinect.



תמונה 5: המשחק לאחר עליית הלוח ולפני שהשחקנים התחילו לסמן



תמונה 6: המשחק לאחר שני סימונים

5. סיכום

בפרויקט זה מימשנו משחק איקס-עיגול מגע בשימוש במקרן עם מנשק *DirectX* ובמצלמת *Kinect*.

במשחק שני השחקנים מזהים לפי המיקום האופקי שלהם מ *Kinect* והם יכולים לסמן את המשבצת הרצויה בכף היד שלהם.

שיפורים אפשריים לעבודה עתידית:

- זיהוי השחקנים לפי הפנים שלהם .
- סימון המשבצת הרצויה באצבע ולא בכף היד.
- סימון המשבצת הרצויה ע"י ציור X או O במשבצת המתאימה.
- שחקן אחד ישחק במחשב והשני בשימוש *Kinect*.

- <http://www.directxtutorial.com/> תרגולים ב *directX* מהאתר:
- *Microsoft Kinect SDK*
<http://msdn.microsoft.com/en-us/library/hh855348.aspx>
- [http://web.mit.edu/be.400/www/SVD/Singular Value Decomposition.htm](http://web.mit.edu/be.400/www/SVD/Singular%20Value%20Decomposition.htm)
- <http://www.public.iastate.edu/~dicook/JSS/paper/code/svd.c>