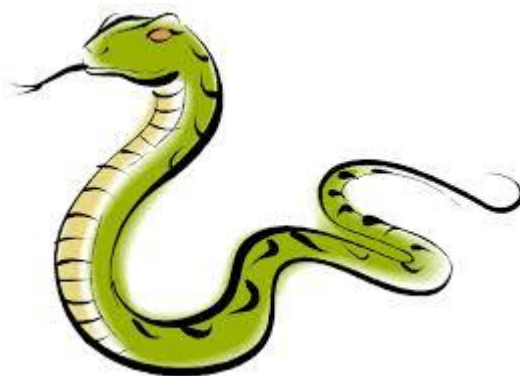
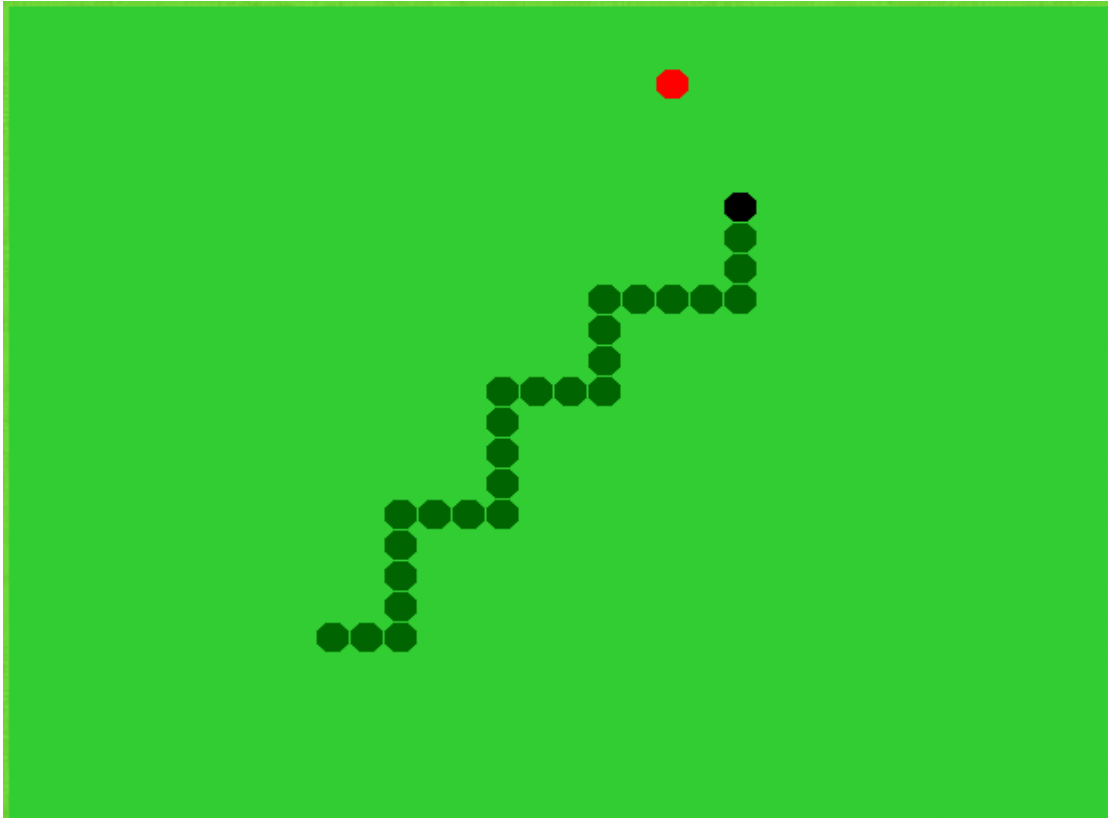


פרוייקט גמר – הנדסת תכנה



Snake



מגיש: בן-אל מוסיוב לבאן

תעודת זהות: 318653300

כיתה: יב'3

בית ספר: הגימנסיה "העברית" הרצליה , תל אביב-יפו

מנחה: שלומי אח-נין

שפת עבודה: #C

2015

תוכן עניינים:

4.....	מבוא
5.....	מטרות עבודה
6.....	תהליך פיתוח
7.....	קצת על ה-Snake
8.....	הדרכה ליצירת פרויקט ב-C#
15.....	פירוט והסבר על המשחק והפעלתו
23.....	קשיים בהם נתקלתי במהלך הפרויקט
25.....	בעיות שלא תוקנו
26.....	תוכנות עזר בהם נעזרתי בבניית הפרויקט
29.....	תיאור משתני התכנית והפונקציות
48.....	סיכום
49.....	תודות
50.....	ביבליוגרפיה
51.....	תדפיס קוד התכנית

מבוא

כשהייתי בן 7 בערך, בזמן שהיו את הפלאפונים הישנים (דמויי הבלוקים), אהבתי לשחק במשחק שהכי אתגר אותי שם – הסנייק. התמכרתי אליו, ביליתי המון זמן בלשחק בו. אהבתי אותו מאוד, ממגוון סיבות, אחת מהן – המשחק דורש ריכוז ושליטה נכונה במקשים. כשנודע לי על הפרויקט של 5 יח"ל במגמת הנדסת תכנה, החלטתי שהגיע הזמן שאני אבנה את משחק הסנייק שלי, שייראה ויתוכנן בצורה שאני אבחר.

בהתחלה, לא בדיוק הבנתי למה אני מכניס את עצמי, חשבתי שבניית הפרויקט תהיה קלה בהרבה יותר, אבל עם הזמן הבנתי שהפרויקט שבחרתי לא כל כך תמים כמו שהוא נראה, למרות שזה "רק סנייק", ודרושה מחשבה והשקעה רבה על מנת ליישם אותו כמו שצריך.

חשבתי רבות איך לא לבנות סתם עוד משחק סנייק שדומה לאחרים, ומה אפשר להוסיף על מנת לגרום להפקת הנאה מרבית למשתמש, לכן עיצבתי את הפרויקט בצורה נוחה שתסב תחושה של הנאה למשתמש והוספתי אלמנטים ייחודיים מיוחדים – שיר המתנגן ברקע, והרעדת מסך המשחק (בשני מצבי המשחק).

פרויקט משחק Snake נבנה במסגרת פרויקט הגמר (5 יח"ל) בשפת #C בעזרת התוכנה Microsoft Visual C# 2010. #C הינה שפת תכנות מונחת עצמים שפותחה ע"י מיקרוסופט.

תכנות הפרויקט הייתה עבודה לא קלה ודרשה תכנון נכון, נאלצתי להתגבר על מכשולים רבים ולהשתמש בכל הידע אותו רכשתי בשפת #C, שפה אשר התחלתי ללמוד כבר מכיתה י'. בכיתה י' עד י"א למדתי לתכנת בסביבת ה-Console, ובפרויקט הנוכחי למדתי להשתמש ב-windows form.

מטרות העבודה

מטרות הפרויקט (5 יח"ל) הוא בנייה והשלמת משחק מתחילתו עד סופו. המשחק שבחרתי הינו משחק לילדים, snake (נחש).

אני מאמין שבעתיד אעבוד בתחום ההיי-טק. תחום המחשוב יהיה הדבר העיקרי בחיי וזו סיבה נוספת לעבוד בתכנות.

מטרה נוספת הייתה ללמוד ולהרחיב את ידיעותינו בשפת התכנות C#, בעזרת למידה עצמית, תוך כדי התמקדות וחיפוש מקורות מהם יהיה ניתן להפיק ולקבל מידע הרלוונטי לפרויקט שלי. המנחה שלנו נוהג לומר "אני זורק אתכם אל הים כדי שתשחו כמו כרישים" - על מנת להדגיש שעיקר הצלחתנו טמונה בלמידה העצמית שלנו.

תהליך הפיתוח

לוח זמנים:

בזמן בניית הפרויקט נקבעו מטרות בהתאם להנחיות משרד החינוך, ובהנחיית מנחה הפרויקטים, שלומי אח-נין.

תכנון הפרויקט:

לשם תכנון הפרויקט נאלצתי לחשוב רבות ולעמוד במסגרת הזמן בה היה עלי לסיים את הפרויקט. השתדלתי לעבוד בצורה היעילה ביותר, ולסיים את תכנות הפרויקט תוך התחשבות בלוח הזמנים.

תהליך בניית הפרויקט:

- תכנון וחשיבה על הדרך שבה הפרויקט הולך להיראות.
- תחילת היישום מרעיון, לתכנית כתובה.
- תכנות הפרויקט בשפת C# וסיומו.
- שיפורים קטנים בעיצוב הפרויקט ובתכנותו.
- תהליך כתיבת הספר.

קצת על ה-Snake...

סנייק (באנגלית: Snake; תרגום: נחש) הוא משחק וידאו שיצא באמצע שנות השבעים של המאה העשרים. הפופולריות של המשחק גברה בשנות התשעים כאשר הוא הוסף לחלק ניכר מהטלפונים הסלולריים.

במשחק שולט השחקן על נחש ארוך ודק המשוטט באזור מבודד, אוסף אוכל (או פריט אחר), ומנסה להימנע מלהתנגש בזנב שלו או בקירות המקיפים את אזור המשחק. הנחש מתארך כשהוא אוכל, וכך קשה יותר להימנע מלהיתקע בזנב או בגוף עצמו. השחקן שולט על כיוון תנועת ראש הנחש (למעלה, ימינה, למטה או שמאלה), ויתר גופו של הנחש עוקב אחרי ראשו. הנחש נמצא בתנועה מתמדת ולא ניתן לעצור אותו במהלך המשחק.

קיימות גרסאות רבות ושונות של המשחק, ביניהן גרסאות שבהם ישנם מכשולים שונים שמקשים על המשחק כגון מוקשים ואוכל רעיל, וגרסאות שבהם יש יותר מנחש יחיד במסך אחד והנחשים השונים מתחרים על המזון. נכון ל-2007, בגרסאות האחרונות ביותר של המשחק ניתן לשחק את הנחש בגוף ראשון ולא רק לראות אותו ממבט-על כמו ברוב הגרסאות האחרות.

היסטוריה:

מקורן של מגוון גרסאותיו של הסנייק באות ממשחק הארקייד "Hustle (עוין)" שהופץ על ידי חברת "גרמלין" בשנת 1977. גרסה מוקדמת של המשחק סנייק שתוכננה למחשבים אישיים, פותחה על ידי התוכניתן הגרמני סאגר בשנת 1979 על מחשב מדגם TRS-80 לאחר מכן נוצרו גרסאות נוספות שפעלו על שנים מדגמי המחשבים האישיים הראשונים Commodore VIC-

20 TI-99/4A- בשנת 1997 נוקיה החלה לצרף גרסה של המשחק לטלפונים הסלולריים שלה, מה שהעלה מאוד את הפופולריות שלו ברחבי העולם. הגרסה של נוקיה תוכנתה בידי טנאלי ארמאנטו והמכשיר הראשון שלה שזכה במשחק הוא הנוקיה 6110.



הדרכה ליצירת פרויקט בC#

המשתנים העיקריים בהם השתמשתי:

קבוצות:

private – קבוצה זו אומרת שהמשתנה הוא משתנה פרטי של אותה מחלקה, המשתנה שנגדיר בתוך המחלקה, יהיה זמין אך ורק במחלקה שבה הגדרנו אותו.

public – קבוצה זו אומרת שהמשתנה אינו משתנה פרטי, ושלא משנה באיזה מחלקה נגדיר אותו, הוא יהיה זמין בכל שאר המחלקות.

const – קבוצה זו הינה קבוצה של משתנים קבועים, הפתוחים לכל המחלקות, ערכם יהיה הערך שנקבע בהתחלה והוא אינו יכול להשתנות במשך כל התוכנית.

סוגי המשתנים העיקריים בהם השתמשתי:

int – משתנה מסוג int הינו משתנה מסוג מספרים ממשיים, התוכן שלו הוא אך ורק מספרים שלמים.

double – משתנה מסוג מספרים ממשיים אשר תוכנו מסוגל להכיל מספרים שלמים וגם מספרים עשרוניים, כגון 0.5 .

bool – משתנה מסוג בוליאני, יכול להכיל אך ורק ערכים של "true" או "false".

string – משתנה מסוג מחרוזת, המסוגל לשמור בתוכו ערך של משפט/מילה וכו.

הגדרת המשתנים:

לשם הגדרת המשתנים נכתוב קודם כל את קבוצת המשתנה, לאחר מכן את סוג המשתנה, ולבסוף את שם המשתנה, לדוגמא:

Private int Number;

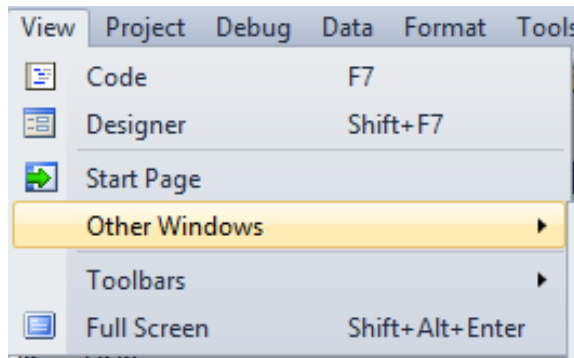
ניתן גם להוסיף למשתנה ערך:

Private int Number = 2;

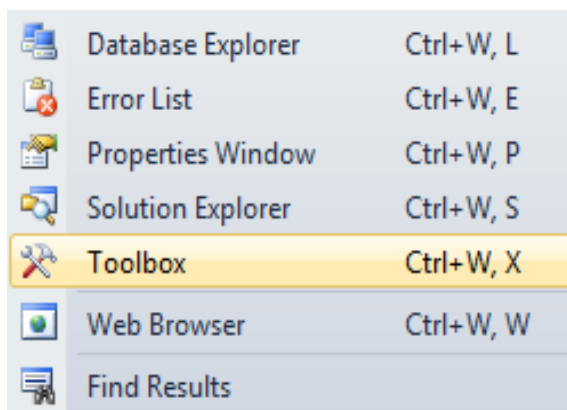
כעת יצרנו משתנה ששמו Number . המשתנה הינו מקבוצת private, מסוג int, המכיל בתוכו את המספר 2.

:Toolbox

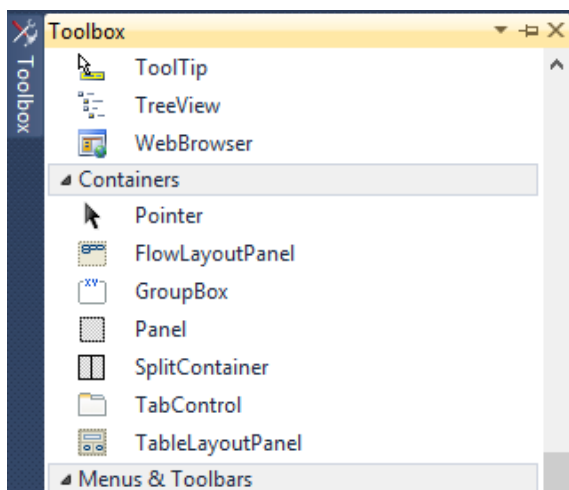
ה - Toolbox הוא כלי חשוב מאוד ב#C שמכיל אפשרויות רבות (כגון: עיצוב הפרויקט, הצגת תמונות, הצגת טקסט, פונקציה שהתוכנית פונה אליה כאשר היא מקבלת לחיצה על העכבר ועוד). השימוש ב - Toolbox אפשרי רק בלשונית הDesign, שבה אנחנו יכולים לעצב בקלות את הפרויקט שלנו.
הערה: להצגת הToolbox נפנה ל View ← Other Windows:



נלחץ על :Toolbox ←



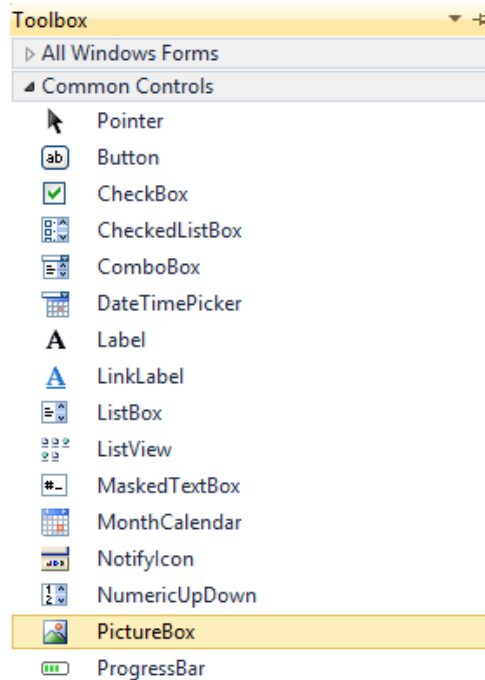
לאחר מכן ה - Toolbox ייפתח בלשונית הDesign:



כלים בסיסיים ב - Toolbox אשר השתמשתי בהם לבניית הפרויקט:

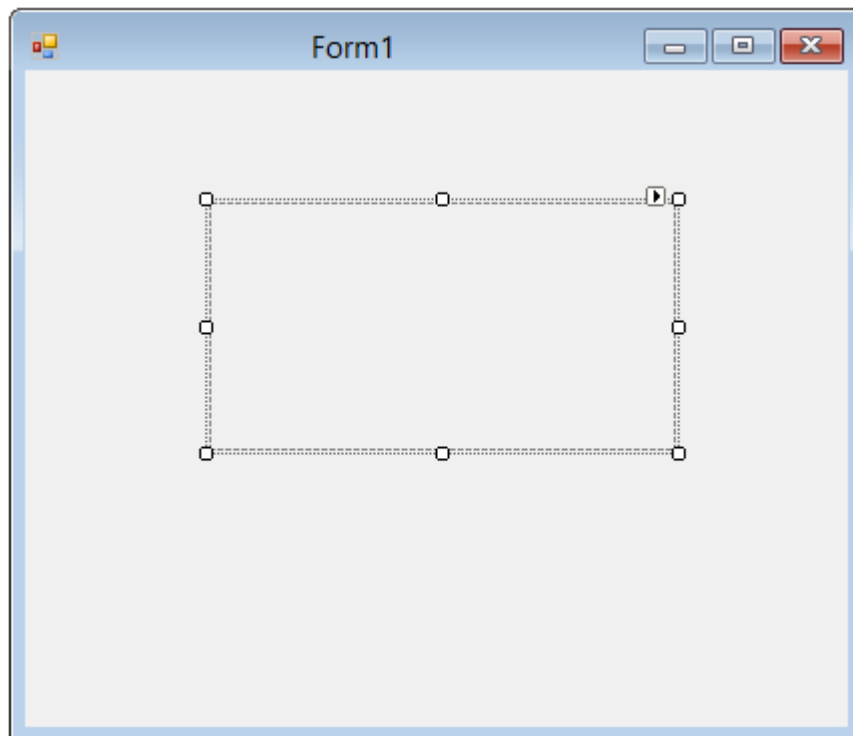
להצגת מסגרת המשחק (ה - canvas) שעליה ה-snake נמצא בתנועה, נפנה

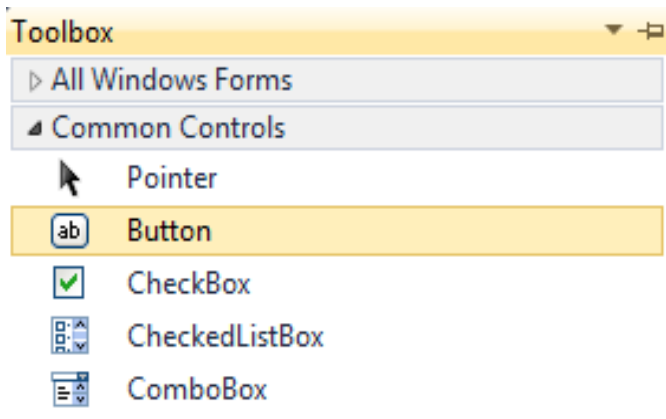
לכלי PictureBox:



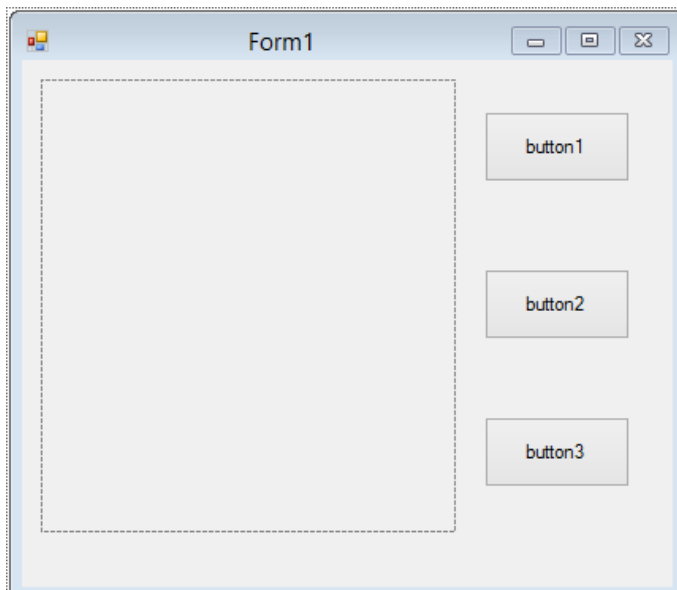
לאחר מכן, נסמן מרובע על החלון/הטופס (form) בגודל שנרצה ונמקם אותו

כרצוננו:





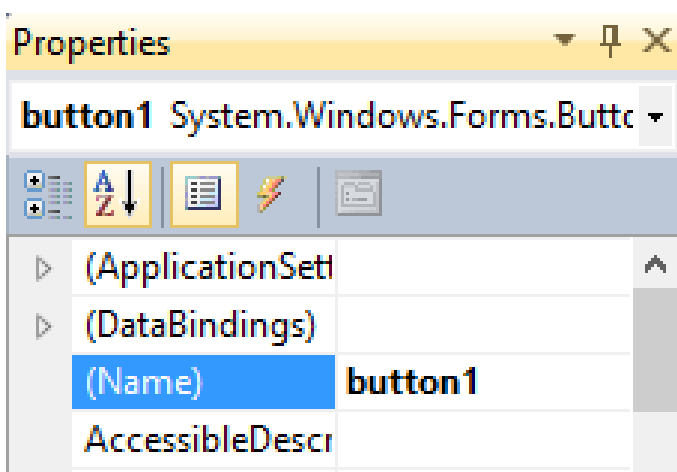
להצגת כפתורים, נפנה לכלי Button:



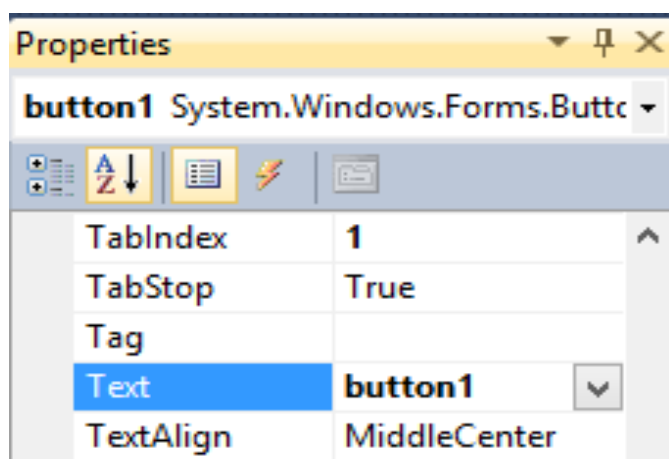
נסמן את גודל הכפתורים שנרצה
ונמקם אותו כרצוננו:

(מלמעלה למטה):

button1
button2
(button3

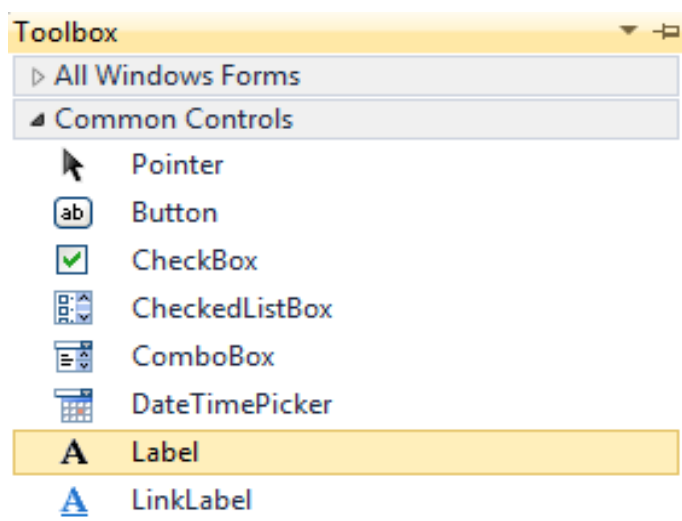


אם נרצה לשנות את השם של אחת
הכפתורים, ניגש לחלון
האפשרויות/מאפיינים (בדרך כלל
מופיע בצד ימין למטה) ונבחר
באפשרות "(Name)".

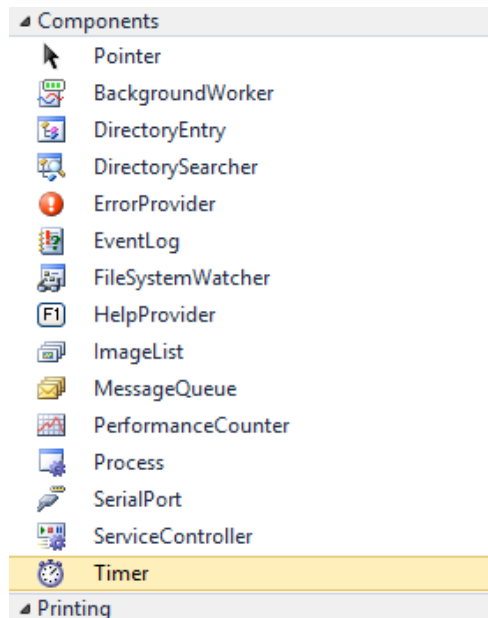


לשינוי טקסט הכפתור (הכיתוב
שיופיע למשתמש), ניגש לאפשרות
:"Text"

להצגת תווית (או תוויות), נפנה לכלי Label:



לשינוי השם של אחת מהתוויות או שינוי הטקסט שלה, נחזור על הפעולות
 שהוצגו לגבי שימוש הכפתור, המתוארות לעיל.

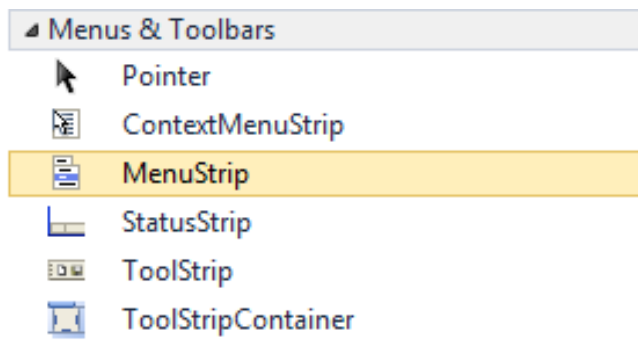


לשימוש בטיימר נפנה לכלי Timer:

טיימר מחכה מספר מסוים של אלפיות שנייה, ולאחר מכן מתחיל/מפעיל אירוע, המכונה "תקתוק" (Tick). זה שימושי להתחלת פעולה, או לחזור על פעולה על בסיס קבוע. להלן:

```
GameTimer.Start();
GameTimer.Tick += UpdateScreen;
```

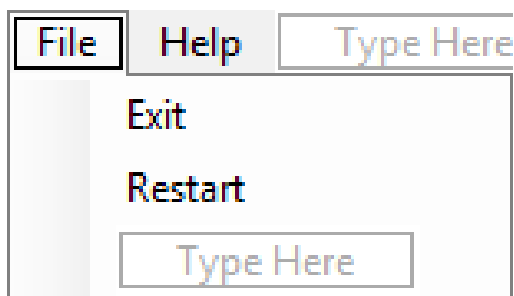
* (עיקרון פעולת עדכון המסך (UpdateScreen) יוסבר בהמשך).



לשימוש ברצועת תפריט נפנה לכלי MenuStrip:

MenuStrip מוסיף שורת תפריט לתכנית בטופס (Form), שניתן לערוך אותה

כרצוננו על מנת להקנות תפעול קל יותר למשתמש:



לדוגמה:

במשבצת המלבנית הריקה שעליה רשום "Type Here", ניתן לרשום אפשרויות/פקודות שונות. לצורך ההדגמה, אבחר ב – Exit.

לוחצים פעמיים ("דאבל-קליק") על המשבצת עליה רשמנו את האפשרות/הפקודה, ונפתחת לנו אוטומטית (בלשונית ה - cs) הפעולה הבאה:

```
private void exitToolStripMenuItem_Click(object sender, EventArgs e)
{
}
}
```

אנו רוצים שברגע שנלחץ על הפקודה Exit, שפירושה יציאה, התכנית/המשחק ייסגר. בעזרת שורת קוד קטנה ניתן להשיג זאת. להלן:

```
private void exitToolStripMenuItem_Click(object sender, EventArgs e)
{
    Application.Exit();
}
}
```

*לשינוי וקביעת ערכים/מאפיינים שונים לגבי כל אחד מהכלים (כפתורים/תוויות/טיימרים וכו') יש לגשת לחלון האפשרויות/המאפיינים (Properties) המופיע בצד ימין למטה, ולשנות/לקבוע את השינוי הרצוי בהתאם.

פירוט והסבר על המשחק והפעלתו

*ניתן למזער או לשנות את גודלם של מסכי המשחק השונים.

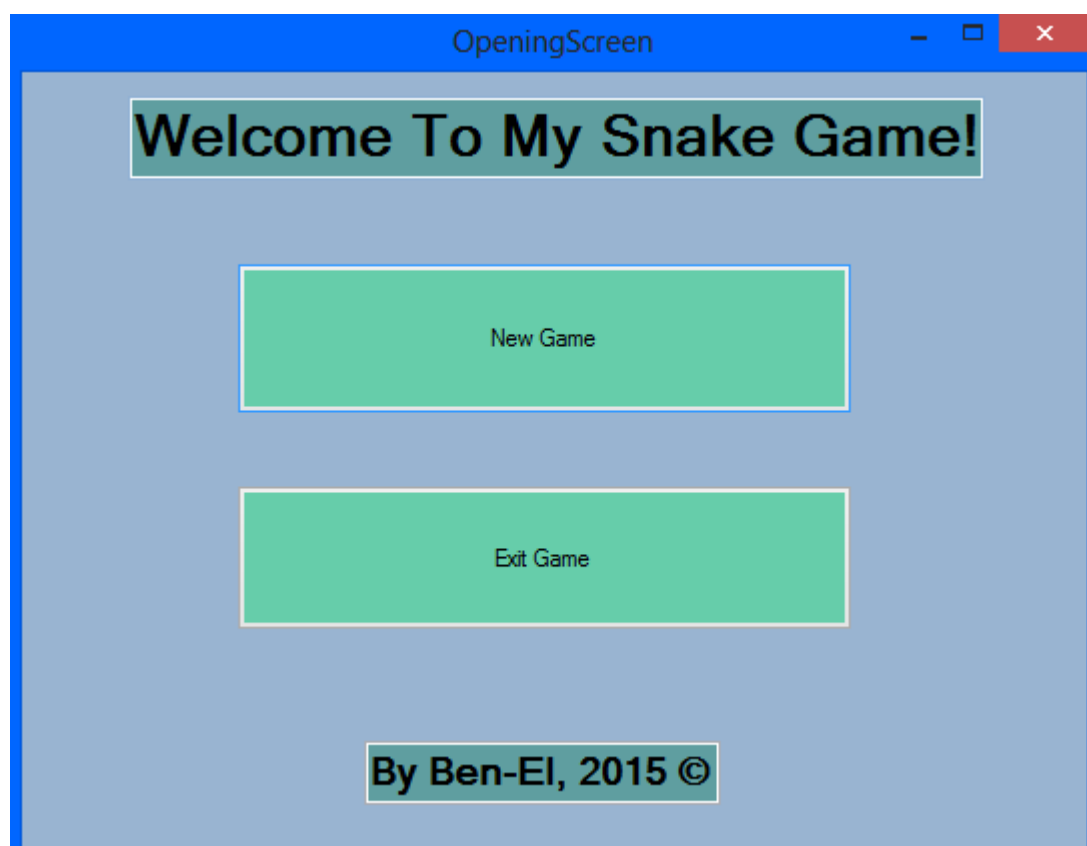
הוראות המשחק:

במשחק הסנייק שלי, ישנם שני מצבי משחק אפשריים – משחק לשחקן יחיד (Single Player) ומשחק לשני שחקנים (Two Players).

המטרה במשחק היא להגיע לכך שגודל הנחש יהיה כגודל מסגרת המשחק, כלומר לאכול את ה"תפוחים", המסומנים כנקודות אדומות (Red dots), לגרום לנחש לגדול כמה שאפשר ובכך לסיים את המשחק. ישנו טיימר בצד מסך המשחק המראה למשתמש כמה זמן הוא שיחק.

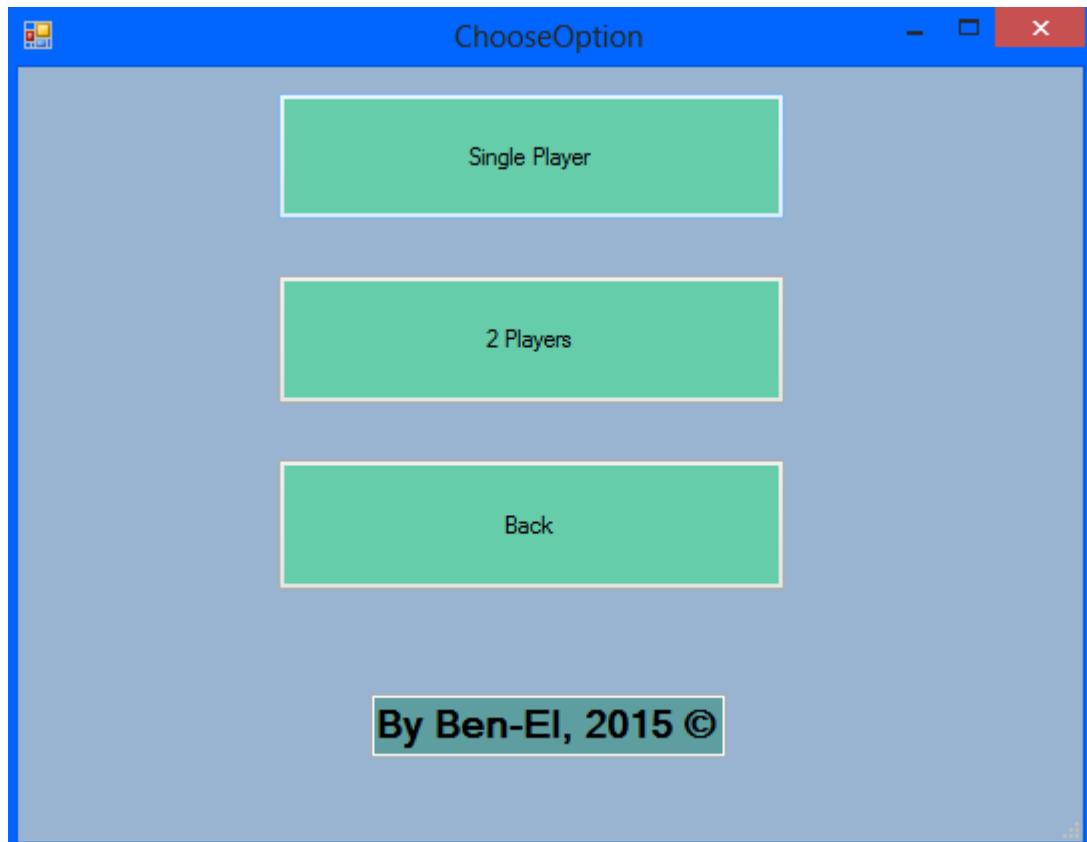
במצב שבו שני שחקנים משחקים המנצח הוא בעל הניקוד הגבוה יותר – גם אם נפסל ראשון.

מסך הפתיחה:



בשביל להתחיל לשחק לוחצים על כפתור המשחק החדש ← "New Game".

לאחר מכן, המשתמש מופנה אל מסך בחירת האופציה, כלומר משחק לשחקן יחיד או משחק לשני שחקנים:



"Single Player" ← למשחק של שחקן יחיד.

"Two Players" ← למשחק של דו-שחקני/שני שחקנים.

"Back" ← כדי לחזור לתפריט ההתחלה.

מקשי המשחק:

במצב שחקן יחיד: מקשי החצים: למעלה, למטה, ימינה, שמאלה.
במצב של שני שחקנים: מקשי החצים: למעלה, למטה, ימינה, שמאלה,
ומקשי המקלדת הבאים: ← W,D,S,A). (A-שמאלה, D-ימינה, W-למעלה, S-
למטה).

*בשני מצבי המשחק: כפתור הרווח (Space) מתחיל את המשחק מחדש.
* כל חצי דקה (30 שניות) מסכי המשחק רועדים (בשני המצבים) כחלק
מהאפקטים הייחודיים שהוספתי לפרויקט.

* לא ניתן לעצור את המשחק, באף אחד מהמצבים.

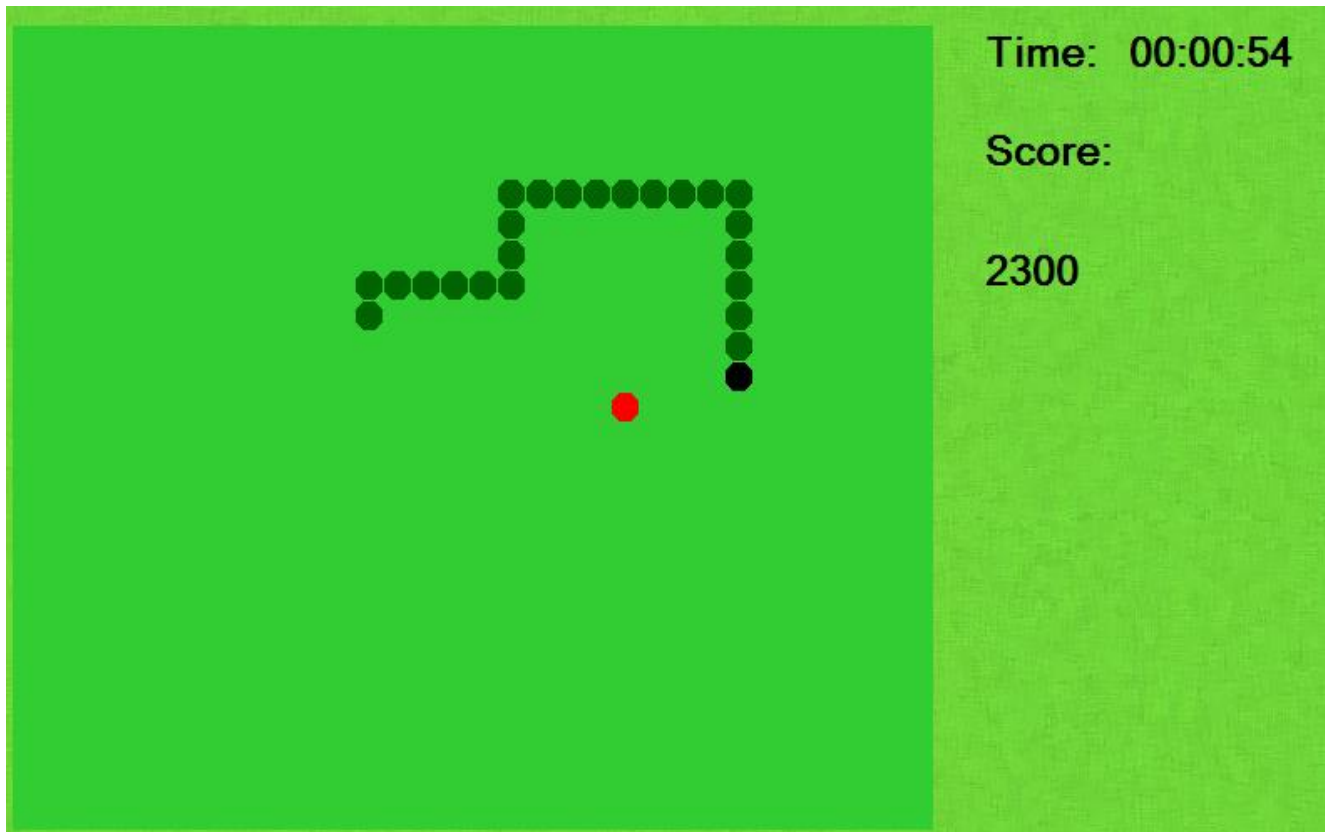
*בשני מצבי המשחק: המקשים ← H - מגביר את מהירות המשחק (High),

L - מוריד את מהירות המשחק (Low),

M - משנה למהירות ברירת המחדל (Medium).

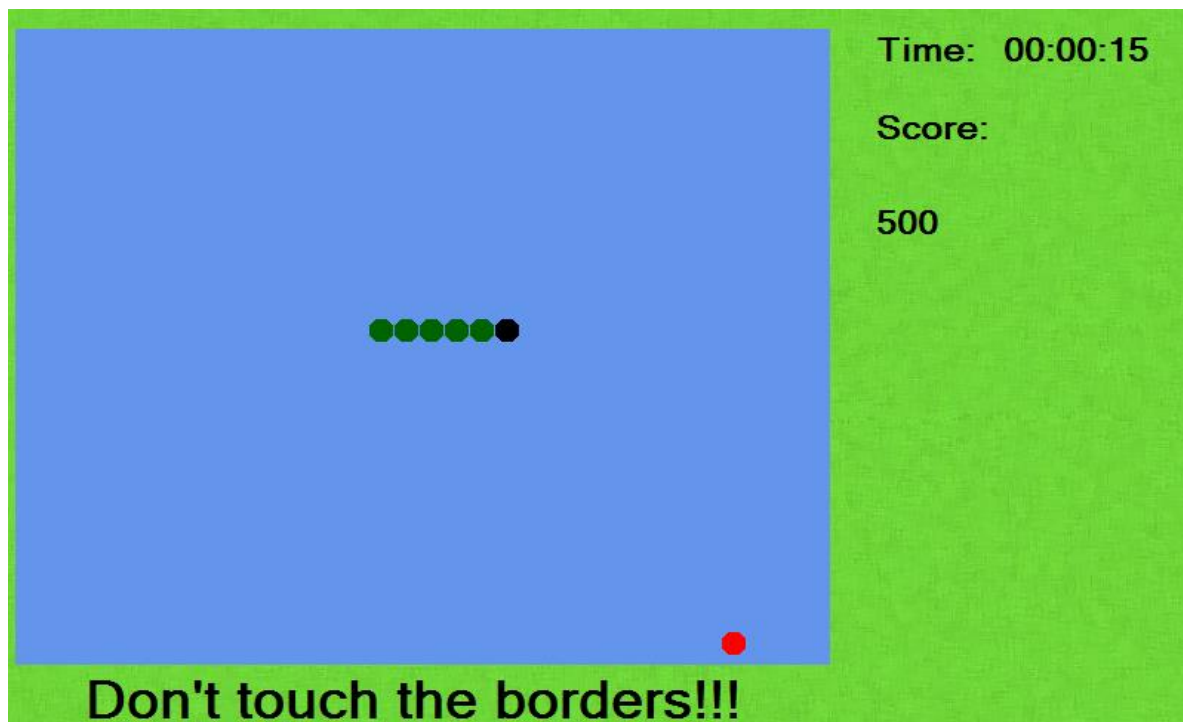
מצב שחקן יחיד:

תמונת מצב שחקן יחיד:

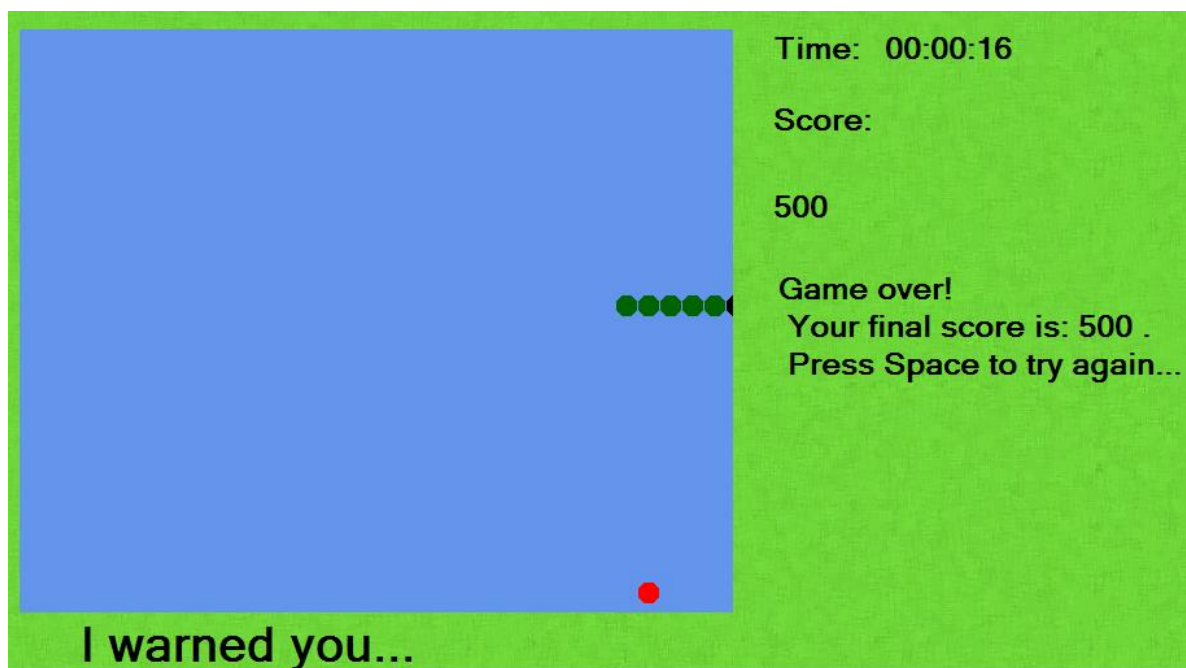


*כל פעם שהשחקן צובר ניקוד המתחלק ב-500, כלומר 500,1000,1500 וכו', גבולות מסגרת המשחק מהווים מכשול. כלומר שחקן העומד על ניקוד המתחלק ב-500, והסנייק שלו נוגע בגבולות מסגרת המשחק – יפסיד. על מנת להתריע את המשתמש, מסגרת המשחק (canvas) משנה את צבעה לצבע כחול-סגול, ובנוסף, מופיעה תווית האזהרה בתחתית מסגרת המשחק עם הכיתוב: "Don't touch the borders", כלומר "לא לגעת בגבולות".

תמונה הממחישה מצב זה:



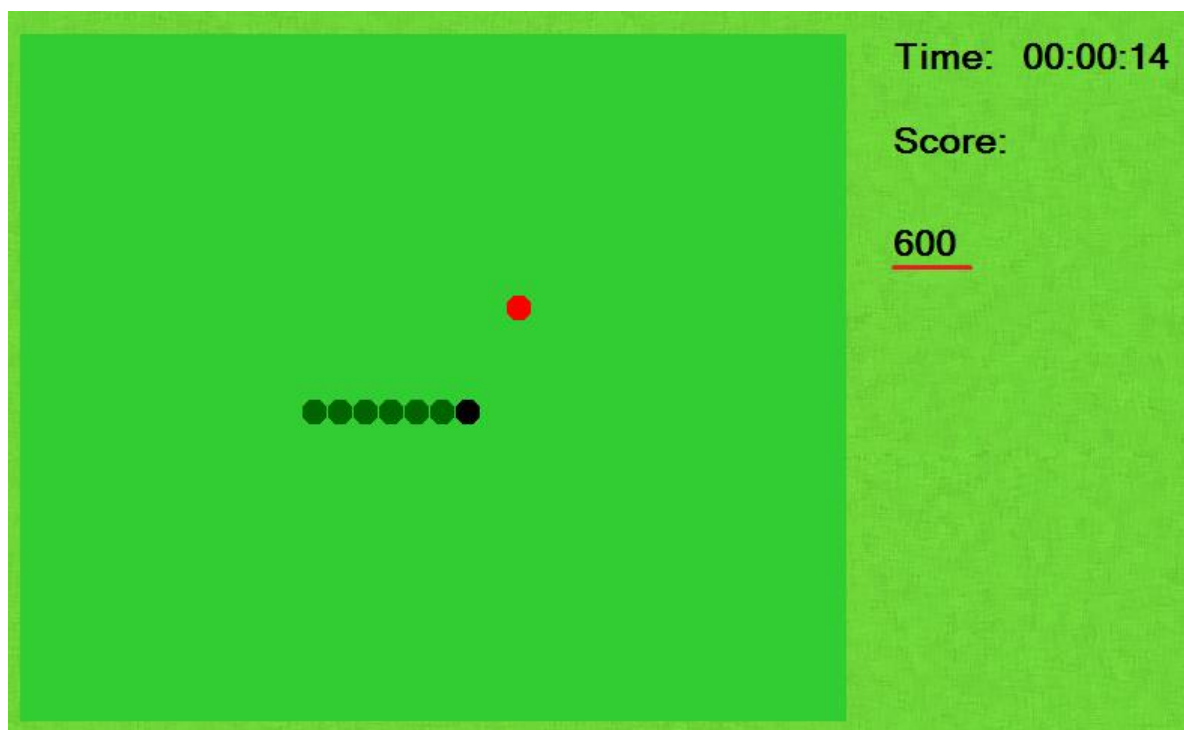
וכאשר נוגעים בגבולות, המשחק נגמר:



כפי שניתן לראות בתמונה לעיל, כאשר המשתמש נפסל במצב זה, משתנה תווית האזהרה בתחתית המסך ל "I warned you...", כלומר "הזהרתי אותך...".

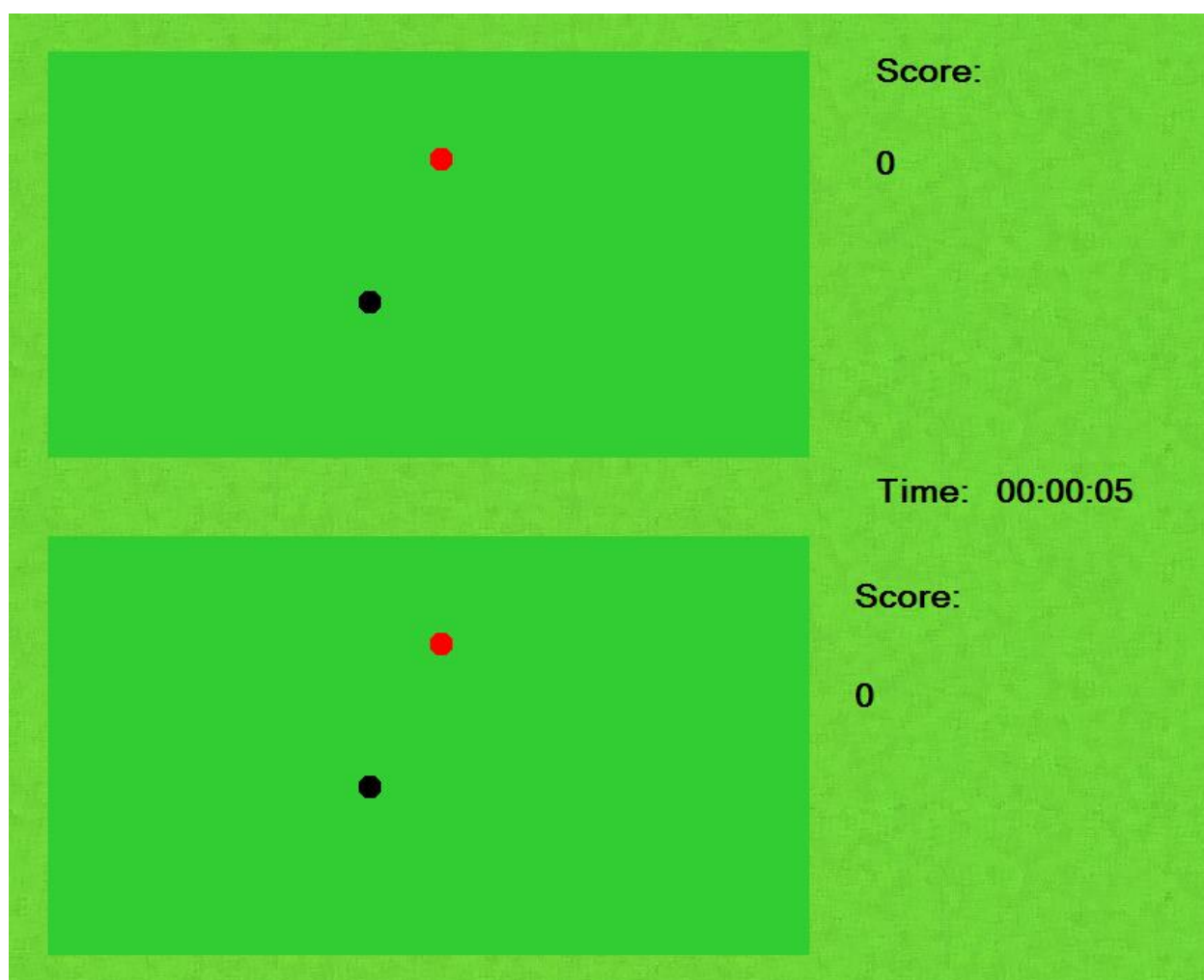
כאשר השחקן עובר את הניקוד המתחלק ב-500, משתנה צבע מסגרת המשחק חזרה לירוק הכהה המוגדר כברירת המחדל.

לדוגמה:



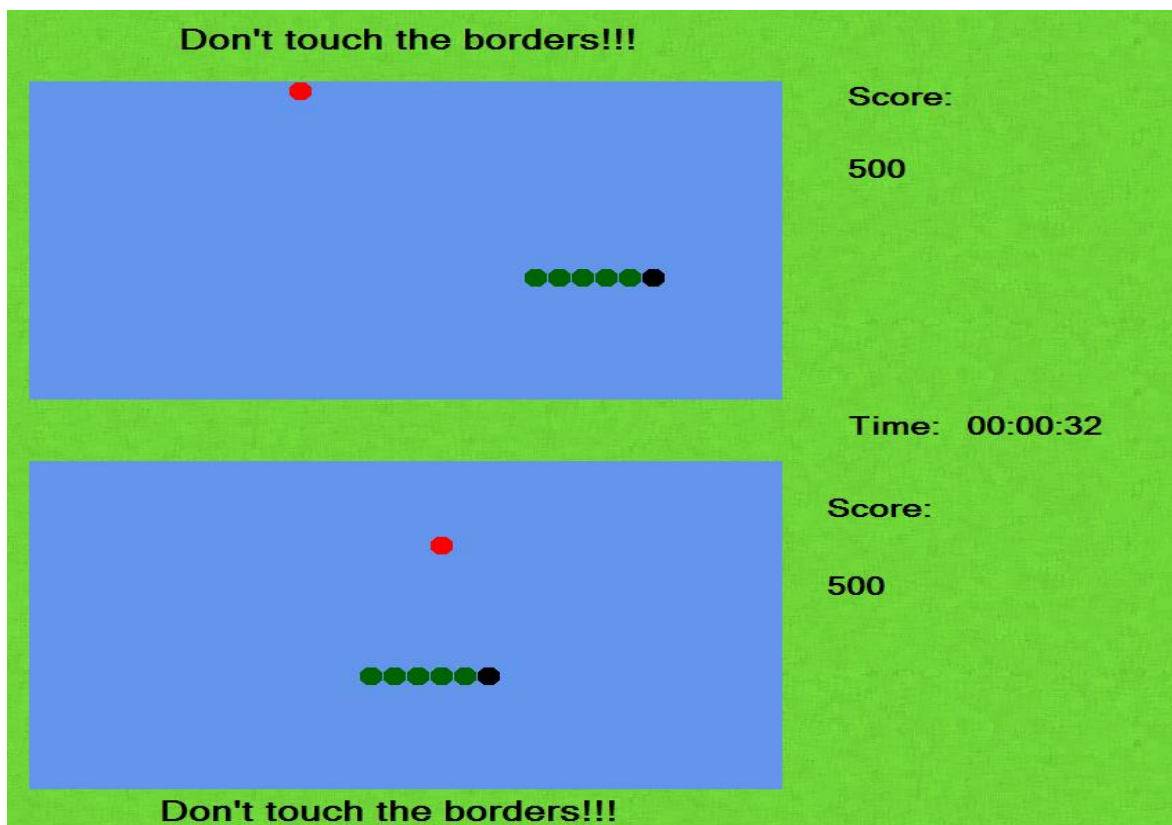
מצב של שני שחקנים (מצב דו-שחקני):

תמונת מצב של שני שחקנים:



*גם כאן, כל פעם שאחד השחקנים צובר ניקוד המתחלק ב-500, כלומר 500, 1000, 1500 וכו', גבולות מסגרות המשחק מהוות מכשול. כלומר שחקן העומד על ניקוד המתחלק ב-500, והסנייק שלו נוגע בגבולות מסגרת המשחק – יפסיד. על מנת להתריע את המשתמש, מסגרות המשחק (canvas) משנות את צבען לצבע כחול-סגול, ובנוסף, מופיעות תוויות האזהרה בתחתית מסגרות המשחק עם הכיתוב: "Don't touch the borders", כלומר "לא לגעת בגבולות".

תמונה הממחישה מצב זה:




כאשר אחד מהשחקנים נוגע בגבולות המשחק, המשחק עבורו נגמר:



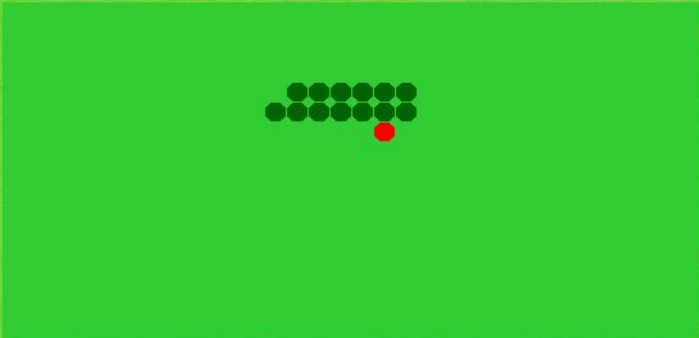
במצב אשר אחד השחקנים נפסל בניקוד מסוים, והשני עוד לא נפסל ועקף את הניקוד של החשקן הראשון, תווית ה- "Loser", שפירושה "מפסידן", מופיעה.

תמונה הממחישה זאת:



The screenshot shows a game interface with a green background. On the left, a black rectangular area contains a 3x3 grid of black dots and a single red dot. Below this grid, the text "Loser!" is displayed in a yellow box. To the right of this area, the text "Score:" is followed by the number "800". Below the score, the text "The game is over for you...!" is displayed, followed by "Your final score is: 800 ." and "Wait until your opponent is done...". Below this, the text "Time: 00:00:30" is shown. At the bottom of the screenshot, another black rectangular area contains a horizontal row of 10 black dots and a single red dot. To the right of this area, the text "Score:" is followed by the number "900".

במצב של תיקו, תווית ה- "It's a draw!!!", שפירושה "זה תיקו!!!", מופיעה:

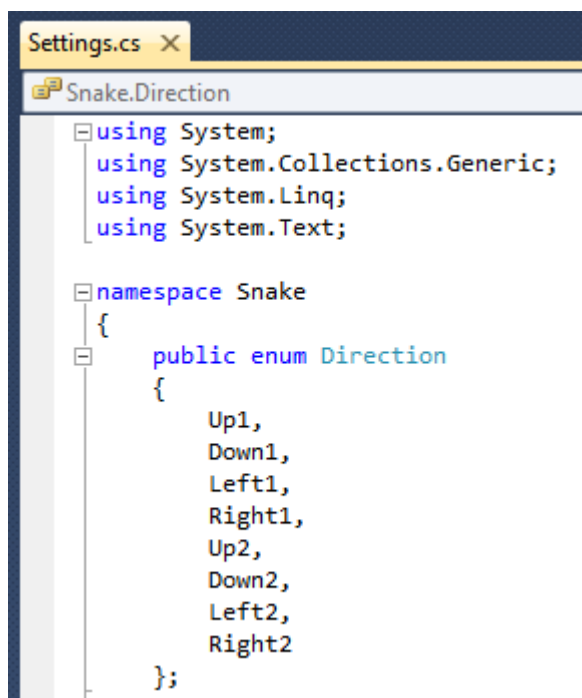


The screenshot shows a game interface with a green background. On the left, a black rectangular area contains a 3x3 grid of black dots and a single red dot. Below this grid, the text "It's a draw!!!" is displayed. To the right of this area, the text "Score:" is followed by the number "1300". Below the score, the text "Game over!" is displayed, followed by "Your final score is: 1300 ." and "Press Space to try again...". Below this, the text "Time: 00:00:44" is shown. At the bottom of the screenshot, another black rectangular area contains a horizontal row of 10 black dots and a single red dot. To the right of this area, the text "Score:" is followed by the number "1300". Below the score, the text "Game over!" is displayed, followed by "Your final score is: 1300 ." and "Press Space to try again...".

קשיים בהם נתקלתי במהלך הכנת הפרויקט

תכנות הפרויקט היה החלק הקשה ביותר בעבודה, בו נתקלתי במספר קשיים:

*הקושי המרכזי בפרויקט היה לתכנת את מצב המשחק הדו-שחקני. הבעיה העיקרית הייתה שליטה אינדיבידואלית של כל שחקן ב ← snake שלו. בהתחלה, הצלחתי להגיע לכך שייווצר מצב דו שחקני. היו שני מסגרות משחק נפרדות, שני נחשים (snakes) נפרדים בכל מסגרת, אך התקשיתי בכך שאחד השחקנים ישלוט על הסנייק העליון, ושהאחר ישלוט על הסנייק התחתון. כלומר, להגיע למצב שהשליטה על הנחש הראשון תתבצע ע"י אחד מהשחקנים בעזרת מקשי החצים, ושהנחש השני יושלט ע"י השחקן השני בעזרת המקשים W,D,S,A. בהתחלה, לחיצה על אחד מהמקשים (מקשי החצים/המקשים-W,D,S,A) הייתה משתלטת על התזוזה של שני הנחשים. לדוגמה, כשהייתי לוחץ על המקש "למטה", שני הנחשים זזו למטה, במקום שרק אחד מהם יזוז למטה והאחר יישאר כפי שהוא. לבסוף, לאחר ניסיונות רבים, הצלחתי לפתור בעיה זו. למחלקה Settings, הוספתי תכונות (ציבוריות ← "Public") שיהיו רלוונטיים לתפעול אינדיבידואלי הנחש הראשון (העליון) והשני (התחתון). להלן:



```
Settings.cs X
Snake.Direction
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Snake
{
    public enum Direction
    {
        Up1,
        Down1,
        Left1,
        Right1,
        Up2,
        Down2,
        Left2,
        Right2
    };
}
```

← Right1 ,Left1 ,Down1 ,Up1
לתזוזה של הנחש הראשון (העליון).

← Right2 ,Left2 ,Down2 ,Up2
לתזוזה של הנחש הראשון (התחתון).

וכן, כיווני ברירת מחדל לכל אחד מהנחשים (כיווני ברירת מחדל \leftarrow הכוונה, שכשמתחילים את המשחק במצב הדו-שחקני, לאיזה כיוון הנחשים יזוזו), להלן:

```
public static Direction direction1 { get; set; }
public static Direction direction2 { get; set; }
```

direction1 \leftarrow לכיוון ברירת המחדל של הנחש העליון.

direction2 \leftarrow לכיוון ברירת המחדל של הנחש התחתון.

*קושי נוסף היה לבדוק מי המנצח ומי המפסיד (במצב הדו-שחקני) גם לאחר שאחד השחקנים נפסל, והאחר עדיין בתזוזה ("עדיין חי"). על קושי זה התגברתי בכך שאת הפעולה CheckForLoser() מיקמתי גם בתוך הפעולה Die() (אשר בודקת אם האם הנחש "מת"), על מנת לבדוק למי מהשחקנים יש את הניקוד הגבוה ביותר, לאחר כל אכילה, גם אחרי פסילה של אחד השחקנים.

*עוד קושי היה לשנות את צבע מסגרת המשחק (בשני מצבי המשחק) בכל פעם שהשחקן מגיע לניקוד אשר מתחלק ב $\leftarrow 500$. לאחר חקירה וחיפוש, מצאתי באינטרנט את התשובה לכך. להלן קטע הקוד המבצע זאת בפרויקט שלי:

```
canvas.BackColor = System.Drawing.Color.CornflowerBlue;
```

*עוד קושי היה להרעיד את מסכי המשחק (בכל 30 שניות בשני מצבי המשחק). לאחר חיפוש מעמיק הצלחתי לפתור גם קושי זה, להלן קטע הקוד:

```
private void Shake(Form form)
{
    var original = form.Location;
    Random rnd = new Random();
    const int shake_amplitude = 5;
    for (int i = 0; i < 10; i++)
    {
        form.Location = new Point(original.X + rnd.Next(-shake_amplitude, shake_amplitude),
            original.Y + rnd.Next(-shake_amplitude, shake_amplitude));
    }
}
```


בעיות שלא תוקנו

הבעיה היחידה שלא הצלחתי לתקן היא שכשהנחש מתחיל להגיע ממדים יחסית גדולים, חתיכות עיגולי האוכל נמצאים על גופו. בכך בעצם, סיום המשחק אינו אפשרי. מפאת קוצר הזמן והעומס הרב, כרגע בזמן כתיבת שורות אלה (17/03/15), לא אוכל לעבוד על תיקון בעיה זו, אך אמשיך לנסות לתקן זאת בעתיד הקרוב, בתקווה שהבעיה תיפתר.

באגים שונים:

הפעלתי המון פעמים את פרויקט ה - Snake שלי, ניסיתי לאתר כמה שיותר באגים ותיקנתי את כל הבאגים שאותרו (חוץ מהבאג שלעיל), אך אולי ישנה אפשרות שעדיין קיימים כמה באגים במשחק שאותם לא הצלחתי לאתר.

תוכנות עזר בהם נעזרתי בבניית הפרויקט

:Microsoft Visual C# 2008

בעזרת תוכנה זו בנית את פרויקט Snaken שלי, כל הרעיונות ודרכי החשיבה מסתורות בתוך שורות הקוד אשר נכתבו בתוכנה זו.

:Google Chrome

בעזרת האינטרנט מצאתי מידע על שפת ה-C#, על בעיות שנתקלתי בהם ועל נושאים שונים שעזרו לי לשם ביצוע, תכנון וסיום בניית הפרויקט.

:Format Factory

Format Factory היא תכנת המרה חינמית, בעלת מאפיינים רבים המאפשרת להמיר אודיו, וידיאו ואף תמונות בין רוב הפורמטים הפופולריים.

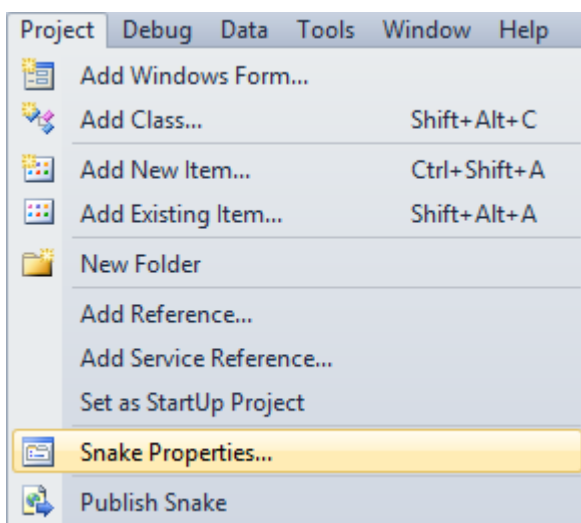
הפורמטים הנתמכים הקיימים ב ← Format Factory כוללים:

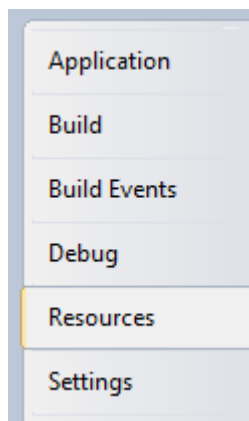
WMV, AVI MPG, MP3, WMA, AAC, JPG, PNG, GIF ועוד רבים אחרים. ויותר מכך, Format Factory תומך בפורמטים של מדיה הקיימים ברוב המכשירים הניידים העיקריים, והם כוללים PSP ואת iPhone.

בתכנה זו נעזרתי על מנת להכניס לפרויקט את השיר המתנגן ברקע (impossible). המרתי את קובץ האודיו של השיר מפורמט mp3 לפורמט wav, התכנה c# תומכת בסוגי אודיו אך ורק מפורמט wav. לאחר ההמרה לפורמט זה, נעתיק את הקובץ לתיקיה bin שנמצאת בתוך תיקיית הפרויקט שלנו.

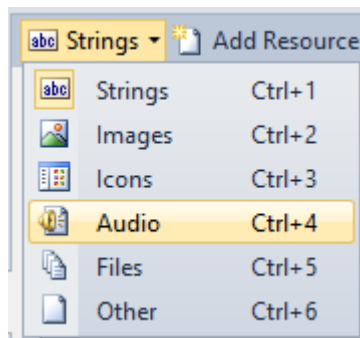
הכנסת השיר:

לוחצים על לשונית Project המופיעה למעלה, ונכנסים למאפייני הפרויקט (במקרה שלי ← Snake Properties). כך:

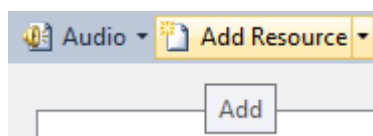




לאחר מכן בחלון שנפתח, לוחצים על Resources:

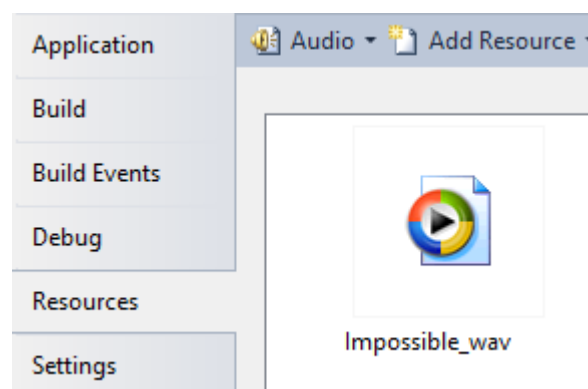


לוחצים על Audio:



ואז על Add Resource:

לאחר מכן נפתח חלון שבו נצטרך את המיקום ממנו נבחר את השיר שלנו. ניגש לתיקיית הפרויקט שלנו, משם לתיקיה bin ונבחר בשיר ששמרנו שם. במקרה שלי, הוספת השיר "Impossible". להלן תמונה הממחישה זאת:



כעת על מנת להפעיל זאת מיד כאשר המשתמש מפעיל את התכנית שלי, ניגש ללשונית OpeningScreen.cs, ונוסיף שם את השורה: `using System.Media`.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Media; // שורת הקוד הנחוצה להפעלת קבצי אודיו בפרויקט
```

בלשונית OpeningScreen.cs, כתבתי את הפעולה הבאה, אשר מאפשרת לי להשמיע את השיר מיד כשהמשתמש מפעיל את המשחק (השיר מתנגן כל הזמן, כלומר בתום השיר, הוא מתנגן מחדש):

```
private void OpeningScreen_Load(object sender, EventArgs e)
{
    SoundPlayer sp = new
    SoundPlayer(Snake.Properties.Resources.Impossible_wav);
    sp.PlayLooping();
}
```

תיאור משתני התכנית והפונקציות

תיאור משתני התכנית:

שם	IO	תפקיד
X	int	מכיל את נקודת ציון ה X של העיגול (circle), כלומר הרכיב האופקי של circle
Y	int	מכיל את נקודת ציון ה Y של העיגול (circle), כלומר הרכיב האנכי של circle
Up1	enum (Direction)	הכיוון (מעלה) של ה snake במצב שחקן יחיד, ובמצב דו-שחקני – מייצג את הכיוון של ה snake העליון
Down1	enum (Direction)	הכיוון (מטה) של ה snake במצב שחקן יחיד, ובמצב דו-שחקני – מייצג את הכיוון של ה snake העליון
Right1	enum (Direction)	הכיוון (ימינה) של ה snake במצב שחקן יחיד, ובמצב דו-שחקני – מייצג את הכיוון של ה snake העליון
Left1	enum (Direction)	הכיוון (שמאלה) של ה snake במצב שחקן יחיד, ובמצב דו-שחקני – מייצג את הכיוון של ה snake העליון
Up2	enum (Direction)	הכיוון (מעלה) של ה snake במצב דו-שחקני – מייצג את הכיוון של ה snake התחתון
Down2	enum (Direction)	הכיוון (מטה) של ה snake במצב דו-שחקני – מייצג את הכיוון של ה snake התחתון
Right2	enum (Direction)	הכיוון (ימינה) של ה snake במצב דו-שחקני – מייצג את הכיוון של ה snake התחתון
Left2	enum (Direction)	הכיוון (שמאלה) של ה snake במצב דו-שחקני – מייצג את הכיוון של ה snake התחתון
Width	int	משתנה המכיל את רוחב העיגולים (circles) בפיקסלים
Height	int	משתנה המכיל את אורך העיגולים (circles) בפיקסלים

משתנה הקובע את מהירות השחקן	int	Speed
משתנה המכיל את הניקוד המצטבר של השחקן במצב של שחקן יחיד, ובמצב הדו-שחקני – מכיל את ערך הניקוד המצטבר של השחקן העליון	int	Score1
משתנה המכיל את הניקוד המצטבר של השחקן במצב הדו-שחקני – מכיל את ערך הניקוד המצטבר של השחקן התחתון	int	Score2
משתנה המכיל בתוכו את ערך הניקוד שמוסף לאחר כל אכילה	int	Points
משתנה הקובע האם המשחק הכולל נגמר או לא (כלומר שני השחקנים הפסידו)	bool	GameOver
כיוון ברירת המחדל של השחקן במצב של שחקן יחיד, ובמצב הדו-שחקני – של השחקן העליון	Direction	direction1
כיוון ברירת המחדל של השחקן התחתון במצב הדו-שחקני	Direction	direction2
snake עצמו - רשימה של circles מהם בנוי snake	List	Snake
עיגול המציין את האוכל ("התפוח", הנקודה האדומה) במצב של שחקן יחיד	Circle	food
עיגול המציין את האוכל ("התפוח", הנקודה האדומה) של השחקן העליון במצב הדו שחקני	Circle	food1
עיגול המציין את האוכל ("התפוח", הנקודה האדומה) של השחקן התחתון במצב הדו שחקני	Circle	food2
משתנה הקובע אם המשחק של השחקן במצב של שחקן יחיד נעצר או לא, כלומר אם המשחק שלו נגמר (נמצא בתוך הפעולה Die())	bool	Pause
משתנה הקובע אם המשחק של השחקן העליון במצב הדו שחקני נעצר או לא, כלומר אם המשחק שלו נגמר או לא (נמצא בתוך הפעולה Die1())	bool	Pause1
משתנה הקובע אם המשחק של השחקן התחתון במצב הדו שחקני נעצר או לא, כלומר אם המשחק שלו נגמר או לא (נמצא בתוך הפעולה Die2())	bool	Pause2
משתנה המכיל את הערך של השניות	double	Seconds

משתנה המכיל את הערך של הדקות	int	Minutes
משתנה המכיל את הערך של השעות	int	Hours
מחרוזת המכילה את ערך הזמן הכולל שעובר	string	time
משתנה הבודק אם קיימת התנגשות בגבולות מסגרת המשחק במצב של שחקן יחיד	bool	CheckColssionWithGameBorders
משתנה הבודק אם קיימת התנגשות בגבולות מסגרת המשחק של השחקן העליון המצב הדו שחקני		CheckColssionWithGameBorders1
משתנה הבודק אם קיימת התנגשות בגבולות מסגרת המשחק		CheckColssionWithGameBorders2
מחרוזת השומרת את הערך של המשתנה של השניות	string	s
מחרוזת השומרת את הערך של המשתנה של הדקות	string	m
מחרוזת השומרת את הערך של המשתנה של השעות	string	h
משתנה המכיל בתוכו את הערך המיקום האופקי המקסימלי של מסגרת המשחק (canvas)	int	MaxXPos
משתנה המכיל בתוכו את הערך המיקום האנכי המקסימלי של מסגרת המשחק (canvas)	int	MaxYpos
משתנה רנדומלי שבעזרתו נקבעים גבולות פיזור food (חלקי האוכל) במצב שחקן יחיד	Random	random
משתנה רנדומלי שבעזרתו נקבעים גבולות פיזור food (חלקי האוכל) של השחקן העליון במצב דו-שחקני	Random	random1
משתנה רנדומלי שבעזרתו נקבעים גבולות פיזור food (חלקי האוכל) של השחקן התחתון במצב דו-שחקני	Random	random2
נותן את האפשרות לצייר ולערוך דברים על מסגרת המשחק במצב השחקן היחיד	Graphics	canvas
נותן את האפשרות לצייר ולערוך דברים על מסגרת המשחק העליונה במצב הדו שחקני	Graphics	Canvas1
נותן את האפשרות לצייר ולערוך דברים על מסגרת המשחק התחתונה במצב הדו שחקני	Graphics	Canvas2

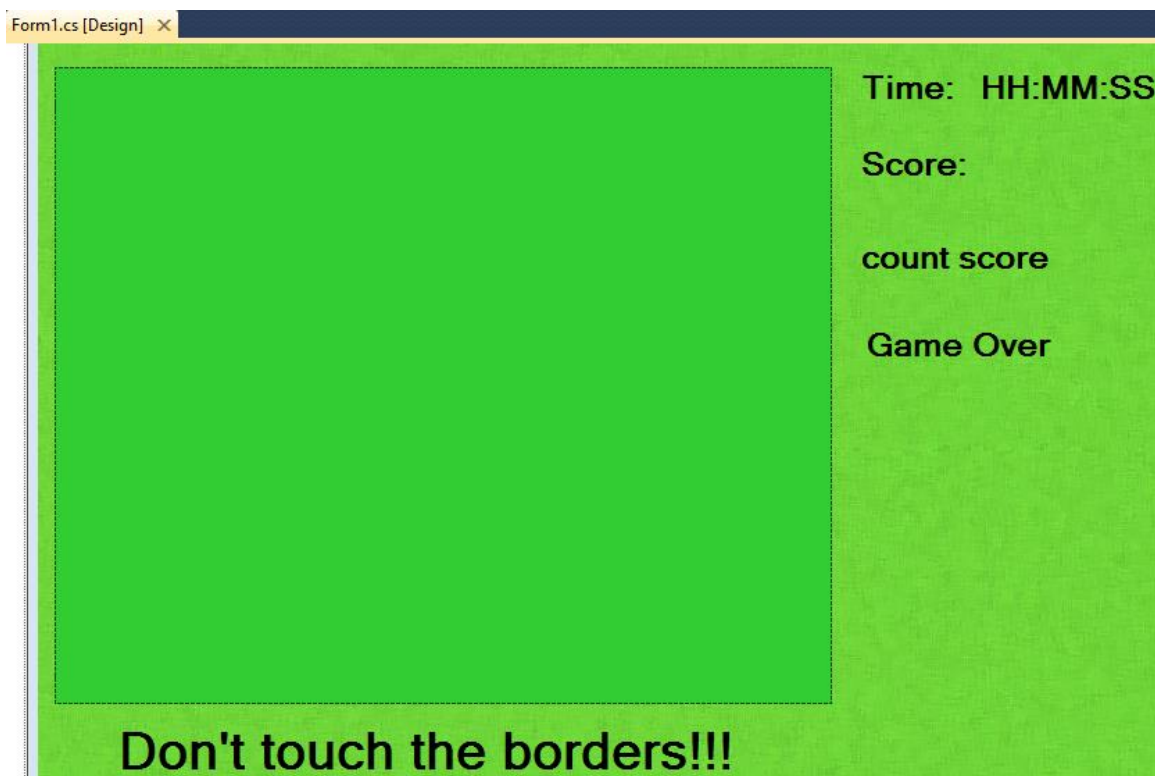
נותן לcircles של snake צבע לבחירת	Brush	snakeColor
מחרוזת המופיעה כשהשחקן מפסיד (במצב שחקן יחיד) המכילה כיתובים שונים ביניהם את הניקוד הכולל של השחקן	string	gameOver
מחרוזת המופיעה כשהשחקן העליון מפסיד (במצב דו שחקני) המכילה כיתובים שונים ביניהם את הניקוד הכולל של אותו שחקן	string	gameOver1
מחרוזת המופיעה כשהשחקן התחתון מפסיד (במצב דו שחקני) המכילה כיתובים שונים ביניהם את הניקוד הכולל של אותו שחקן	string	gameOver2
עצם מסוג עיגול המציין את עיגול האוכל (food1) במצב שחקן יחיד, נמצא בפעולה Eat ()	Circle	Foodcircle
עצם מסוג עיגול המציין את עיגול האוכל (food2) של השחקן העליון במצב הדו-שחקני נמצא בפעולה Eat1()	Circle	Foodcircle1
עצם מסוג עיגול המציין את עיגול האוכל (food2) של השחקן התחתון במצב הדו-שחקני נמצא בפעולה Eat2()	Circle	Foodcircle2
משתנה המקבל את ערך מיקום הטופס (Form)(כלומר – את שיעורי (x,y) שלו. זה מסוג Point)	var	original
משתנה רנדומלי שבו יוכנס טווח פיזור ערכים	Random	rnd
ערך קבוע, השומר בתוכו את ערך ההעתק/המרחק של התופס בזמן רעידתו	const	shake_amplitude

תיאור הפונקציות והסברים:

Form1.cs (מצב שחקן יחיד):

מרכיבי Form1.cs[Design]:

סוג	תפקיד
PictureBox	מסגרת המשחק, ניתן לצייר לערוך ולעדכן פעולות ותרשימים המתרחשים עליו
Label	התווית מאפשרת להציג בתוכה טקסטים שיוצגו למשתמש שינחו אותו ויראו לו את המתרחש
Timer	משמש להתחלת/הפעלת אירוע , וגם למדידת משך הזמן שעובר
MenuStrip	משמש להצגת תפריט שנרכיב כרצוננו למשתמש



תכונות:

רשימה של עיגולים ממנה בנוי הנחש (ה - Snake).

```
private List<Circle> Snake = new List<Circle>();
```

עיגול המציין את עיגול האוכל.

```
private Circle food = new Circle();
```

פעולות:

`private void GeneralTimer1_Tick(object sender, EventArgs e)`

טענת כניסה: אובייקט ששולח את התרחיש, אובייקט שמקבל את התרחיש.
טענת יציאה: אין
מטרה הפעולה: להציג על מסך המשחק את משך הזמן שעובר.

`private void restartToolStripMenuItem_Click(object sender, EventArgs e)`

טענת כניסה: אובייקט ששולח את התרחיש, אובייקט שמקבל את התרחיש.
טענת יציאה: אין.
מטרה הפעולה: כשלוחצים על כפתור ההפעלה מחדש ("Restart") ברצועת התפריט של מסך המשחק, המשחק מופעל מחדש.

`private void exitToolStripMenuItem_Click_1(object sender, EventArgs e)`

טענת כניסה: אובייקט ששולח את התרחיש, אובייקט שמקבל את התרחיש.
טענת יציאה: אין.
מטרה הפעולה: כשלוחצים על כפתור היציאה ("Exit") ברצועת התפריט של מסך המשחק, המשחק נסגר (יציאה מן המשחק).

`private void howToPlayToolStripMenuItem_Click(object sender, EventArgs e)`

טענת כניסה: אובייקט ששולח את התרחיש, אובייקט שמקבל את התרחיש.
טענת יציאה: אין.
מטרה הפעולה: כשלוחצים על כפתור ה"איך משחקים" ("How To Play") ברצועת התפריט של מסך המשחק, מופיע חלון המסביר איך משחקים.

`private void aboutToolStripMenuItem_Click(object sender, EventArgs e)`

טענת כניסה: אובייקט ששולח את התרחיש, אובייקט שמקבל את התרחיש.
טענת יציאה: אין.
מטרה הפעולה: כשלוחצים על כפתור ה"אודות" ("About") ברצועת התפריט של מסך המשחק, מופיע חלון המסביר על אודות המשחק.

`private void StartGame()`

טענת כניסה: אין.
טענת יציאה: אין.
מטרה הפעולה: להתחיל את המשחק.

`private void GenerateFood()`

טענת כניסה: אין.
טענת יציאה: אין.
מטרה הפעולה: לייצר חתיכות (עיגולי) אוכל ולמקם אותם באופן רנדומלי על מסגרת המשחק.

`private void UpdateScreen(object sender, EventArgs e)`

טענת כניסה: אובייקט ששולח את התרחיש, אובייקט שמקבל את התרחיש.
טענת יציאה: אין.
מטרה הפעולה: להפעיל ולעדכן את כל הפעולות והשינויים המתרחשים על מסגרת המשחק.

`private void canvas_Paint(object sender, PaintEventArgs e)`

טענת כניסה: אובייקט ששולח את התרחיש, אובייקט שמקבל את התרחיש.
טענת יציאה: אין.
מטרה הפעולה: לערוך ולקבוע את הגרפיקה על מסגרת המשחק, לדוגמה – לצייר את הנחש על מסגרת המשחק ולצייר ולמקם באופן רנדומלי את חתיכות האוכל.

`private void MovePlayer()`

טענת כניסה: אין.
טענת יציאה: אין.
מטרה הפעולה: לגרום לתזוזת הנחש וכן להפעיל פעולות שונות כגון – פעולת האכילה ולבדוק האם קיימת התנגשות בגבולות מסגרת המשחק.

`private void ColssionWithGameBorders()`

טענת כניסה: אין.

טענת יציאה: אין.

מטרה הפעולה: לבדוק האם קיימת התנגשות בגבולות מסגרת המשחק.

`private void Form1_KeyDown(object sender, KeyEventArgs e)`

טענת כניסה: אובייקט ששולח את התרחיש, אובייקט שמקבל את התרחיש.

טענת יציאה: אין.

מטרה הפעולה: זיהוי קלט (מקשי המשחק) ולאפשר את השימוש בהם ובפעולותיהם.

`private void Form1_KeyUp(object sender, KeyEventArgs e)`

טענת כניסה: אובייקט ששולח את התרחיש, אובייקט שמקבל את התרחיש.

טענת יציאה: אין.

מטרה הפעולה: זיהוי קלט (מקשי המשחק) ולאפשר את השימוש בהם ובפעולותיהם.

`private void Eat()`

טענת כניסה: אין.

טענת יציאה: אין.

מטרה הפעולה: להוסיף את עיגולי האוכל לגוף הנחש כאשר חלה התנגשות בהם ("לאכול").

`private void Die()`

טענת כניסה: אין.

טענת יציאה: אין.

מטרה הפעולה: לעצור את תזוזת הנחש, כלומר "לגרום לו למות".

`private void Shake(Form form)`

טענת כניסה: טופס (Form).

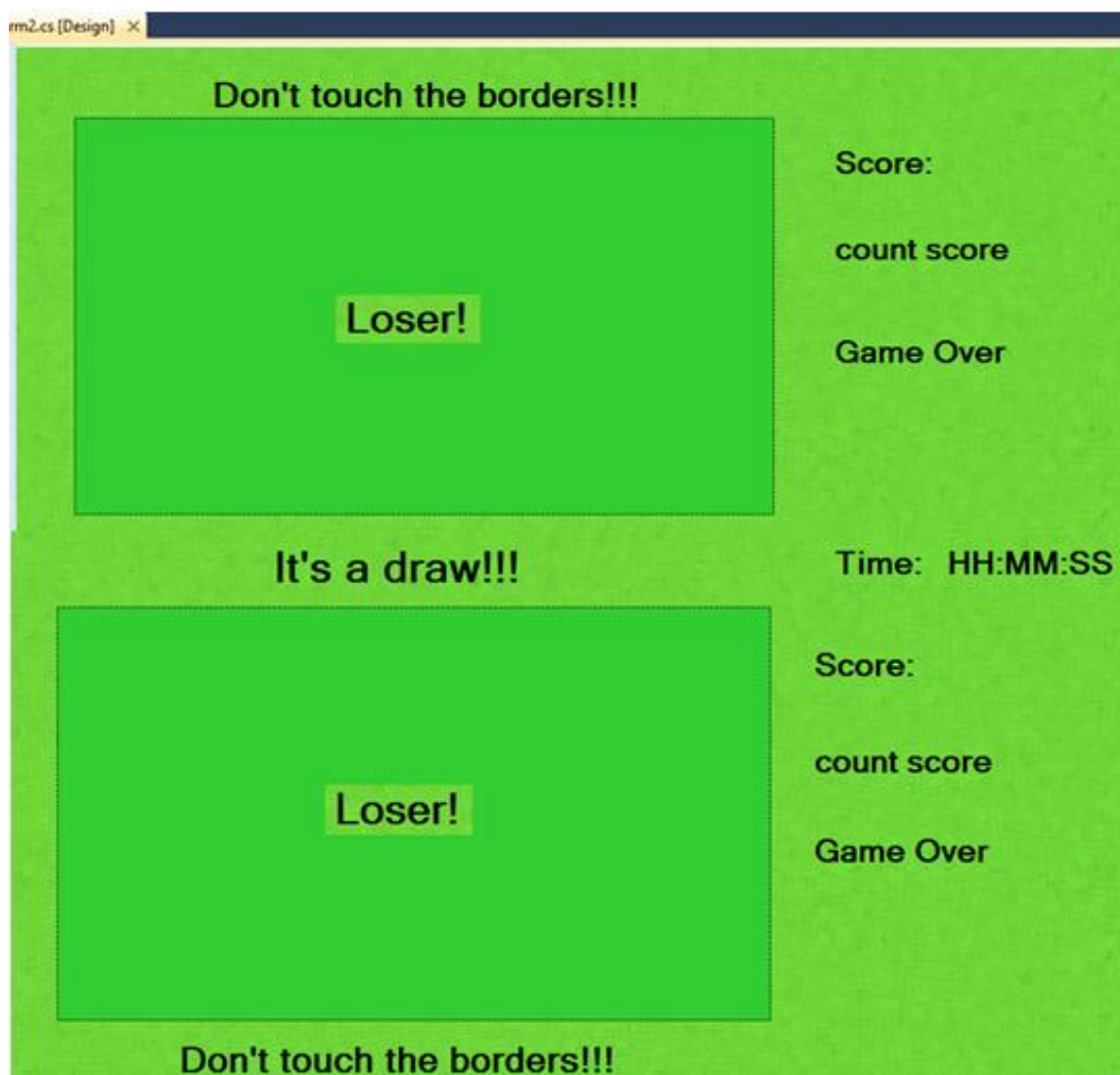
טענת יציאה: אין.

מטרה הפעולה: להרעיד את מסך המשחק (כל 30 שניות).

Form2.cs (מצב דו-שחקני):

מרכיבי Form2.cs[Design]:

סוג	תפקיד
PictureBox	מסגרת המשחק, ניתן לצייר לערוך ולעדכן פעולות ותרשימים המתרחשים עליו
Label	התווית מאפשרת להציג בתוכה טקסטים שיוצגו למשתמש שינחו אותו ויראו לו את המתרחש
Timer	משמש להתחלת/הפעלת אירוע , וגם למדידת משך הזמן שעובר
MenuStrip	משמש להצגת תפריט שנרכיב כרצוננו למשתמש



תכונות:

רשימה של עיגולים ממנה בנוי הנחש במסגרת המשחק העליונה (Snake1).

```
private List<Circle> snake1 = new List<Circle>();
```

רשימה של עיגולים ממנה בנוי הנחש במסגרת המשחק העליונה (Snake2).

```
private List<Circle> snake2 = new List<Circle>();
```

עיגול המציין את עיגול האוכל של הנחש במסגרת המשחק העליונה.

```
private Circle food1 = new Circle();
```

עיגול המציין את עיגול האוכל של הנחש במסגרת המשחק התחתונה.

```
private Circle food2 = new Circle();
```

פעולות:

```
private void GeneralTimer2_Tick(object sender, EventArgs e)
```

טענת כניסה: אובייקט ששולח את התרחיש, אובייקט שמקבל את התרחיש.

טענת יציאה: אין

מטרה הפעולה: להציג על מסך המשחק את משך הזמן שעובר.

```
private void restartToolStripMenuItem_Click(object sender, EventArgs e)
```

טענת כניסה: אובייקט ששולח את התרחיש, אובייקט שמקבל את התרחיש.

טענת יציאה: אין.

מטרה הפעולה: כשלוחצים על כפתור ההפעלה מחדש ("Restart") ברצועת התפריט של מסך המשחק, המשחק מופעל מחדש.

```
private void exitToolStripMenuItem_Click_1(object sender, EventArgs e)
```

טענת כניסה: אובייקט ששולח את התרחיש, אובייקט שמקבל את התרחיש.

טענת יציאה: אין.

מטרה הפעולה: כשלוחצים על כפתור היציאה ("Exit") ברצועת התפריט של מסך המשחק, המשחק נסגר (יציאה מן המשחק).

```
private void howToPlayToolStripMenuItem_Click(object sender, EventArgs e)
```

טענת כניסה: אובייקט ששולח את התרחיש, אובייקט שמקבל את התרחיש.
טענת יציאה: אין.
מטרה הפעולה: כשלוחצים על כפתור ה"איך משחקים" ("How To Play") ברצועת התפריט של מסך המשחק, מופיע חלון המסביר איך משחקים.

```
private void aboutToolStripMenuItem_Click(object sender, EventArgs e)
```

טענת כניסה: אובייקט ששולח את התרחיש, אובייקט שמקבל את התרחיש.
טענת יציאה: אין.
מטרה הפעולה: כשלוחצים על כפתור ה"אודות" ("About") ברצועת התפריט של מסך המשחק, מופיע חלון המסביר על אודות המשחק.

```
private void StartGame()
```

טענת כניסה: אין.
טענת יציאה: אין.
מטרה הפעולה: להתחיל את המשחק.

```
private void GenerateFood1()
```

טענת כניסה: אין.
טענת יציאה: אין.
מטרה הפעולה: לייצר חתיכות (עיגולי) אוכל ולמקם אותם באופן רנדומלי על מסגרת המשחק העליונה.

```
private void GenerateFood2()
```

טענת כניסה: אין.
טענת יציאה: אין.
מטרה הפעולה: לייצר חתיכות (עיגולי) אוכל ולמקם אותם באופן רנדומלי על מסגרת המשחק התחתונה.

`private void UpdateScreens(object sender, EventArgs e)`

טענת כניסה: אובייקט ששולח את התרחיש, אובייקט שמקבל את התרחיש.
טענת יציאה: אין.
מטרה הפעולה: להפעיל ולעדכן את כל הפעולות והשינויים המתרחשים על שתי מסגרות המשחק.

`private void canvas1_Paint(object sender, PaintEventArgs e)`

טענת כניסה: אובייקט ששולח את התרחיש, אובייקט שמקבל את התרחיש.
טענת יציאה: אין.
מטרה הפעולה: לערוך ולקבוע את הגרפיקה על מסגרת המשחק העליונה, לדוגמה – לצייר את הנחש על מסגרת המשחק העליונה ולצייר ולמקם באופן רנדומלי את חתיכות האוכל.

`private void canvas2_Paint(object sender, PaintEventArgs e)`

טענת כניסה: אובייקט ששולח את התרחיש, אובייקט שמקבל את התרחיש.
טענת יציאה: אין.
מטרה הפעולה: לערוך ולקבוע את הגרפיקה על מסגרת המשחק התחתונה, לדוגמה – לצייר את הנחש על מסגרת המשחק התחתונה ולצייר ולמקם באופן רנדומלי את חתיכות האוכל.

`private void MovePlayer1()`

טענת כניסה: אין.
טענת יציאה: אין.
מטרה הפעולה: לגרום לתזוזת הנחש הראשון (העליון) וכן להפעיל פעולות שונות כגון – פעולת האכילה ולבדוק האם קיימת התנגשות בגבולות מסגרת המשחק העליונה.

`private void MovePlayer2()`

טענת כניסה: אין.
טענת יציאה: אין.
מטרה הפעולה: לגרום לתזוזת הנחש השני (התחתון) וכן להפעיל פעולות שונות כגון – פעולת האכילה ולבדוק האם קיימת התנגשות בגבולות מסגרת המשחק התחתונה.

`private void ColssionWithGameBorders1()`

טענת כניסה: אין.

טענת יציאה: אין.

מטרה הפעולה: לבדוק האם קיימת התנגשות בגבולות מסגרת המשחק הראשונה.

`private void ColssionWithGameBorders2()`

טענת כניסה: אין.

טענת יציאה: אין.

מטרה הפעולה: לבדוק האם קיימת התנגשות בגבולות מסגרת המשחק התחתונה.

`private void Form2_KeyDown(object sender, EventArgs e)`

טענת כניסה: אובייקט ששולח את התרחיש, אובייקט שמקבל את התרחיש.

טענת יציאה: אין.

מטרה הפעולה: זיהוי קלט (מקשי המשחק) ולאפשר את השימוש בהם ובפעולותיהם.

`private void Form2_KeyUp(object sender, EventArgs e)`

טענת כניסה: אובייקט ששולח את התרחיש, אובייקט שמקבל את התרחיש.

טענת יציאה: אין.

מטרה הפעולה: זיהוי קלט (מקשי המשחק) ולאפשר את השימוש בהם ובפעולותיהם.

`private void Eat1()`

טענת כניסה: אין.

טענת יציאה: אין.

מטרה הפעולה: להוסיף את עיגולי האוכל לגוף הנחש הראשון (העליון) כאשר חלה התנגשות בהם ("לאכול").

`private void Eat2()`

טענת כניסה: אין.

טענת יציאה: אין.

מטרה הפעולה: להוסיף את עיגולי האוכל לגוף הנחש השני (התחתון) כאשר חלה התנגשות בהם ("לאכול").

`private void Die1()`

טענת כניסה: אין.

טענת יציאה: אין.

מטרה הפעולה: לעצור את תזוזת הנחש הראשון (העליון), כלומר "לגרום לו למות".

`private void Die2()`

טענת כניסה: אין.

טענת יציאה: אין.

מטרה הפעולה: לעצור את תזוזת הנחש השני (התחתון), כלומר "לגרום לו למות".

`private void Shake(Form form)`

טענת כניסה: טופס (Form).

טענת יציאה: אין.

מטרה הפעולה: להרעיד את מסך המשחק (כל 30 שניות).

:OpeningScreen.cs

:OpeningScreen.cs [Design] מרכיבי

סוג	תפקיד
TextBox	משמש להצגת/פליטת טקסט
Button	הכפתור מאפשר להציג בתוכה טקסטים שיוצגו למשתמש שינחו ויכוונו אותו

OpeningScreen.cs[Design]:



תכונות: אין.

פעולות:

`private void OpeningScreen_Load(object sender, EventArgs e)`

טענת כניסה: אובייקט ששולח את התרחיש, אובייקט שמקבל את התרחיש.

טענת יציאה: אין.

מטרה הפעולה: להשמיע את השיר המתנגן ברקע בעת הפעלת המשחק.

`private void StartGameButton_Click(object sender, EventArgs e)`

טענת כניסה: אובייקט ששולח את התרחיש, אובייקט שמקבל את התרחיש.
טענת יציאה: אין.
מטרה הפעולה: בעת לחיצה על כפתור המשחק החדש ← להפנות אותך לטופס בחירת האופציה (לבחור בין אופציית המשחק היחיד/מצב דו-שחקני/יציאה).
מטרה הפעולה: בעת לחיצה על כפתור החזרה ← לצאת מהטופס הנוכחי.

`private void ExitGameButton_Click(object sender, EventArgs e)`

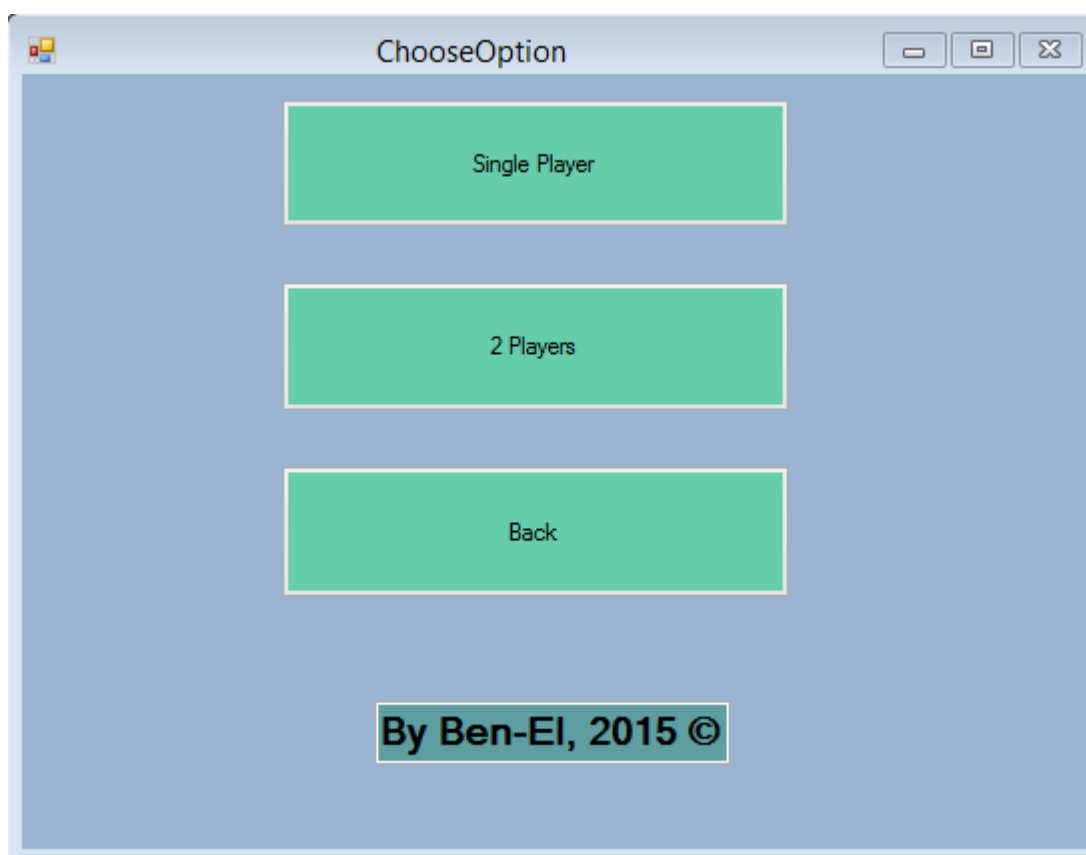
טענת כניסה: אובייקט ששולח את התרחיש, אובייקט שמקבל את התרחיש.
טענת יציאה: אין.
מטרה הפעולה: בעת לחיצה על כפתור החזרה ← לצאת מהטופס הנוכחי.

:ChooseOption.cs

:ChooseOption.cs [Design] מרכיבי

תפקיד	סוג
משמש להצגת/פליטת טקסט	TextBox
הכפתור מאפשר להציג בתוכה טקסטים שיוצגו למשתמש שינחו ויכוונו אותו	Button

ChooseOption.cs[Design]:



תכונות: אין.

פעולות:

`private void SinglePlayerButton_Click(object sender, EventArgs e)`

טענת כניסה: אובייקט ששולח את התרחיש, אובייקט שמקבל את התרחיש.

טענת יציאה: אין.

מטרה הפעולה: בעת לחיצה על כפתור מצב משחק השחקן היחיד ← להתחיל לשחק במצב זה.

`private void TwoPlayesButton_Click(object sender, EventArgs e)`

טענת כניסה: אובייקט ששולח את התרחיש, אובייקט שמקבל את התרחיש.
טענת יציאה: אין.
מטרה הפעולה: בעת לחיצה על כפתור מצב משחק דו-שחקני ← להתחיל לשחק במצב זה.

`private void BackButton_Click(object sender, EventArgs e)`

טענת כניסה: אובייקט ששולח את התרחיש, אובייקט שמקבל את התרחיש.
טענת יציאה: אין.
מטרה הפעולה: בעת לחיצה על כפתור החזרה ← יציאה מהטופס הנוכחי.

סיכום

במהלך הכנת הפרויקט למדתי רבות על תכנות ועל שפת #C. העבודה בחלקה הייתה קשה ומייגעת ממספר סיבות. ראשית, היו בעיות בקוד שלא מצאתי אותן ולקח לי זמן רב להבין מה גרם להן. בנוסף לכך, היו מספר בעיות שדרשו ממני הרבה מחשבה, השקעה ועבודה רבה על מנת לתקן. יתר על כן היו קטעים שלא ידעתי כיצד לממשם בשפת #C ובסביבת העבודה, לכן השקעתי מאמצים וזמן רב בחיפוש אחר פתרון באינטרנט שיספק אותי. לפעמים חוסר ההצלחה למצוא פתרון הוריד את המוטיבציה, אך למזלי חבריי למגמה עזרו לי רבות. בסופו של דבר, למרות הקשיים והבעיות אני מרוצה מבחירת הפרויקט שלי, משום שלמדתי הרבה דברים חדשים, הרחבתי את ידיעותיי ורכשתי מיומנויות חדשות.

תודות

ברצוני להודות למספר אנשים אשר סייעו לי בתהליך בניית הפרויקט וכתובת

הספר:

בראש ובראשונה, רוצה אני להודות למשפחתי על שתמכה בי בבניית המשחק וכתובת הספר.

תודה מיוחדת למנחה הפרויקט – שלומי אח-נין אשר סייע לי בבניית הפרויקט, ופתח בפני את האפשרות ליצור את הפרויקט במסגרת חמש יח"ל נוספות.

תודה ליובל למברוב על שנתן לי את הרעיון לתכנת את אופציית המשחק הדו שחקני (מצב המשחק לשני שחקנים).

תודה לקאי חפץ, מיכאל הראל וניר צור שעזרו לי בתהליך תכנות המשחק וכיוונו אותי לדרך הנכונה.

ביבליוגרפיה:

<https://msdn.microsoft.com/en-us/library/aa288405%28v=vs.71%29.aspx>

<http://stackoverflow.com/questions/16689741/shake-form-on-screen>

<https://msdn.microsoft.com/en-us/library/dd553229.aspx>

<https://www.youtube.com/watch?v=xuElK587jio>

<https://www.youtube.com/watch?v=i6W-aGhlq7M>

https://www.youtube.com/watch?v=pRKie_jqfBA

<https://www.youtube.com/watch?v=qOh4ooHg1UU>

<http://www.xn--7dbbqer5d.co.il/format-factory/format-factory-%D7%94%D7%9E%D7%A8%D7%AA-%D7%A7%D7%91%D7%A6%D7%99%D7%9D-%D7%9Cmp3.html>

<https://msdn.microsoft.com/en-us/library/dd30h2yb%28v=vs.110%29.aspx>

https://www.youtube.com/watch?v=mLIB60wG_AI

<https://social.msdn.microsoft.com/Forums/en-US/28c50b1e-a65e-4304-a2d4-729c2dc7f0b1/how-can-i-add-a-background-music-in-c-winform?forum=csharpgeneral>

תדפיס קוד התכנית

Form1.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Snake
{
    public partial class Form1 : Form
    {
        private List<Circle> Snake = new List<Circle>();
        private Circle food = new Circle();
        bool Pause = false;
        int minutes = 0, hours = 0;
        double seconds = -0.5;
        string time;
        //bool CheckColssionWithGameBorders = false;
        public Form1()
        {
            InitializeComponent();

            //Set settings to default
            new Settings();

            //Set game speed and start timer
            GameTimer.Interval = 800 / Settings.Speed;
            GameTimer.Tick += UpdateScreen;
            GameTimer.Start();

            //Start New game
            StartGame();

            //general timer
            GeneralTimer1.Interval = 1000;
            GeneralTimer1.Enabled = true;
            GeneralTimer1_Tick(null, null);
            GeneralTimer1.Tick += new EventHandler(GeneralTimer1_Tick);
            GeneralTimer1.Start();
        }
        private void GeneralTimer1_Tick(object sender, EventArgs e)
        {
            seconds += 0.5;
            if (seconds == 60) { seconds = 0; minutes++; }
            if (minutes == 60) { minutes = 0; hours++; }
            if (hours == 10) { this.Close(); }
            string s;
            if (seconds < 10) { s = "0" + seconds; }
            else { s = "" + seconds; }
        }
    }
}
```

```

        string m;
        if (minutes < 10) { m = "0" + minutes; }
        else { m = "" + minutes; }
        string h;
        if (hours < 10) { h = "0" + hours; }
        else { h = "" + hours; }
        time = h + ":" + m + ":" + s;
        TimeLable1.Text = time;
    }

    private void restartToolStripMenuItem_Click(object sender, EventArgs e)
    {
        StartGame();
        Pause = false;
    }

    private void exitToolStripMenuItem_Click(object sender, EventArgs e)
    {
        Application.Exit();
    }

    private void howToPlayToolStripMenuItem_Click(object sender, EventArgs
e)
    {
        MessageBox.Show("Use the arrow keys to move. 'H' to increase the
speed, 'L' to decrease the speed, 'M' to turning back to default speed.
'Space' to restart. 'Esc' to exit. Try to collect as many red dots as you
can!", "Game instructions");
    }

    private void aboutToolStripMenuItem_Click(object sender, EventArgs e)
    {
        MessageBox.Show("Made by Ben-El, 2015 @", "Snake - About");
    }

    private void StartGame()
    {
        canvas.BackColor = System.Drawing.Color.LimeGreen;
        WarningBordersLable.Text = "Don't touch the borders!!!";
        labelGameOver.Visible = false;

        //Set settings to default
        new Settings();
        GameTimer.Interval = 800 / Settings.Speed; //Set speed to the
regular speed

        //Create new player object
        Snake.Clear();
        Circle head = new Circle { X = 10, Y = 10 };
        Snake.Add(head);

        //count score

        labelScore.Text = Settings.Score1.ToString();
        GenerateFood();
    }

    //Place random food object
    private void GenerateFood()
    {

```

```

        int maxXPos = canvas.Size.Width / Settings.Width;
        int maxYPos = canvas.Size.Height / Settings.Height;
        Random random = new Random();
        food = new Circle { X = random.Next(0, maxXPos), Y = random.Next(0,
maxYPos) };
    }

    private void UpdateScreen(object sender, EventArgs e)
    {
        //Check for Game Over
        if (Settings.GameOver)
        {
            //Check if Space is pressed
            if (Input.KeyPressed(Keys.Space))
            {
                // new Settings();
                StartGame();
            }
        }
        else
        {
            //if (Input.KeyPressed(Keys.P))
            //{
            //    Thread.Sleep(100);
            //    Pause = !Pause; // pause state is changed
            //}
            if (Pause == false)
            {
                if (Input.KeyPressed(Keys.Right) && Settings.direction1 !=
Direction.Left1)
                    Settings.direction1 = Direction.Right1;
                else if (Input.KeyPressed(Keys.Left) && Settings.direction1
!= Direction.Right1)
                    Settings.direction1 = Direction.Left1;
                else if (Input.KeyPressed(Keys.Up) && Settings.direction1
!= Direction.Down1)
                    Settings.direction1 = Direction.Up1;
                else if (Input.KeyPressed(Keys.Down) && Settings.direction1
!= Direction.Up1)
                    Settings.direction1 = Direction.Down1;
                MovePlayer();
            }
            if (Input.KeyPressed(Keys.H))
            {
                GameTimer.Interval = 300 / Settings.Speed; //increases the
speed
            }
            if (Input.KeyPressed(Keys.M))
            {
                GameTimer.Interval = 800 / Settings.Speed; //regular speed
            }
            if (Input.KeyPressed(Keys.L))
            {
                GameTimer.Interval = 1500 / Settings.Speed; //decreases the
speed
            }
            if (Input.KeyPressed(Keys.Space))
            {
                StartGame();
                GameTimer.Interval = 800 / Settings.Speed;
                Pause = false;
            }
        }
    }
}

```

```

        seconds = -1;
        minutes = 0;
        hours = 0;
        GeneralTimer1.Start();
    }
    if (Settings.Score1 != 0 && Settings.Score1 % 500 == 0)
    {
        WarningBordersLable.Visible = true;
        canvas.BackColor = System.Drawing.Color.CornflowerBlue;
        ColssionWithGameBorders();
    }
    else
    {
        WarningBordersLable.Visible = false;
        canvas.BackColor = System.Drawing.Color.LimeGreen;
    }
    if (((seconds != 0 && seconds % 30 == 0) || (minutes > 0 &&
minutes % 1 == 0 && seconds == 0)) && Pause == false)
    {
        Shake(this);
    }

    if (Input.KeyPressed(Keys.Escape))
    {
        Application.Exit();
    }
}

canvas.Invalidate();
}

private void canvas_Paint(object sender, PaintEventArgs e)
{
    Graphics canvas = e.Graphics;
    if (!Settings.GameOver)
    {
        //Set colour of snake

        //Draw snake
        for (int i = 0; i < Snake.Count; i++)
        {
            Brush snakeColour;
            if (i == 0)
                snakeColour = Brushes.Black;    //Draw head
            else
                snakeColour = Brushes.DarkGreen; //Rest of body

            //Draw snake
            canvas.FillEllipse(snakeColour,
                new Rectangle(Snake[i].X * Settings.Width,
                    Snake[i].Y * Settings.Height,
                        Settings.Width, Settings.Height));

            //Draw Food
            canvas.FillEllipse(Brushes.Red,
                new Rectangle(food.X * Settings.Width,
                    food.Y * Settings.Height, Settings.Width,
Settings.Height));
        }
    }
}

```

```

    }
    if (Pause == true)
    {
        string gameOver = "Game over! \n Your final score is: " +
Settings.Score1 + " ." + "\n Press Space to try again...";
        labelGameOver.Text = gameOver;
        labelGameOver.Visible = true;
    }
}

private void MovePlayer()
{
    for (int i = Snake.Count - 1; i >= 0; i--)
    {
        //Move head
        if (i == 0)
        {
            switch (Settings.direction1)
            {
                case Direction.Right1:
                    Snake[i].X++;
                    break;
                case Direction.Left1:
                    Snake[i].X--;
                    break;
                case Direction.Up1:
                    Snake[i].Y--;
                    break;
                case Direction.Down1:
                    Snake[i].Y++;
                    break;
            }

            //Get maximum X and Y Pos
            int maxXPos = canvas.Size.Width / Settings.Width;
            int maxYPos = canvas.Size.Height / Settings.Height;

            if (Snake[i].X < 0)
                Snake[i].X = maxXPos;
            if (Snake[i].Y < 0)
                Snake[i].Y = maxYPos;
            if (Snake[i].X > maxXPos)
                Snake[i].X = 0;
            if (Snake[i].Y > maxYPos)
                Snake[i].Y = 0;

            //Detect collision with body
            for (int j = 1; j < Snake.Count; j++)
            {
                if (Snake[i].X == Snake[j].X &&
                    Snake[i].Y == Snake[j].Y)
                {
                    Die();
                }
            }
        }
    }
}

```

```

        //Detect collision with food pieces
        if (Snake[0].X == food.X && Snake[0].Y == food.Y)
        {
            Eat();
        }
    }
    else
    {
        //Move body
        Snake[i].X = Snake[i - 1].X;
        Snake[i].Y = Snake[i - 1].Y;
    }
    //if (Pause == true)
    //{
    //    Die();
    //}
}

private void ColssionWithGameBorders()
{
    int maxXPos = canvas.Size.Width / Settings.Width;
    int maxYPos = canvas.Size.Height / Settings.Height;
    bool CheckColssionWithGameBorders = false;
    for (int i = Snake.Count - 1; i >= 0; i--)
    {
        //Detect colliision with game borders
        if (Snake[i].X < 0 || Snake[i].Y < 0
            || Snake[i].X >= maxXPos || Snake[i].Y >= maxYPos)
        {
            CheckColssionWithGameBorders = true;
            if (CheckColssionWithGameBorders == true)
            {
                WarningBordersLable.Text = "I warned you...";
                Die();
            }
        }
    }
}

private void Form1_KeyDown(object sender, KeyEventArgs e)
{
    Input.ChangeState(e.KeyCode, true);
}

private void Form1_KeyUp(object sender, KeyEventArgs e)
{
    Input.ChangeState(e.KeyCode, false);
}

private void Eat()
{
    //Add circle to body
    Circle Foodcircle = new Circle
    {
        X = Snake[Snake.Count - 1].X,
        Y = Snake[Snake.Count - 1].Y
    };
    Snake.Add(Foodcircle);
}

```



```

        //Update Score
        Settings.Score1 += Settings.Points;
        labelScore.Text = Settings.Score1.ToString();

        GenerateFood();
    }

    private void Die()
    {
        Pause = true;
        GeneralTimer1.Stop();
    }

    private void Shake(Form form)
    {
        var original = form.Location;
        Random rnd = new Random();
        const int shake_amplitude = 5;
        for (int i = 0; i < 10; i++)
        {
            form.Location = new Point(original.X + rnd.Next(-
shake_amplitude, shake_amplitude),
                                original.Y + rnd.Next(-shake_amplitude, shake_amplitude));
        }
    }
}

```

Form1Designer.cs:

```
//namespace Snake
//{
//    partial class Form1
//    {
//        /// <summary>
//        /// Required designer variable.
//        /// </summary>
//        private System.ComponentModel.IContainer components = null;

//        /// <summary>
//        /// Clean up any resources being used.
//        /// </summary>
//        /// <param name="disposing">true if managed resources should be
disposed; otherwise, false.</param>
//        protected override void Dispose(bool disposing)
//        {
//            if (disposing && (components != null))
//            {
//                components.Dispose();
//            }
//            base.Dispose(disposing);
//        }

//        #region Windows Form Designer generated code

//        /// <summary>
//        /// Required method for Designer support - do not modify
//        /// the contents of this method with the code editor.
//        /// </summary>
//        private void InitializeComponent()
//        {
//            this.components = new System.ComponentModel.Container();
//            System.ComponentModel.ComponentResourceManager resources = new
System.ComponentModel.ComponentResourceManager(typeof(Form1));
//            this.canvas = new System.Windows.Forms.PictureBox();
//            this.label1 = new System.Windows.Forms.Label();
//            this.GameTimer = new System.Windows.Forms.Timer(this.components);
//            this.labelGameOver = new System.Windows.Forms.Label();
//            this.labelScore = new System.Windows.Forms.Label();
//            this.menuStrip1 = new System.Windows.Forms.MenuStrip();
//            this.fileToolStripMenuItem = new
System.Windows.Forms.ToolStripMenuItem();
//            this.exitToolStripMenuItem = new
System.Windows.Forms.ToolStripMenuItem();
//            this.helpToolStripMenuItem = new
System.Windows.Forms.ToolStripMenuItem();
//            this.howToPlayToolStripMenuItem = new
System.Windows.Forms.ToolStripMenuItem();
//            this.aboutToolStripMenuItem = new
System.Windows.Forms.ToolStripMenuItem();
//
//            ((System.ComponentModel.ISupportInitialize)(this.canvas)).BeginInit();
//            this.menuStrip1.SuspendLayout();
//            this.SuspendLayout();
//
//            //
//            // canvas
//            //
```

```

//      this.canvas.BackColor = System.Drawing.Color.LimeGreen;
//      this.canvas.Location = new System.Drawing.Point(12, 65);
//      this.canvas.Name = "canvas";
//      this.canvas.Size = new System.Drawing.Size(661, 504);
//      this.canvas.TabIndex = 0;
//      this.canvas.TabStop = false;
//      this.canvas.Paint += new
System.Windows.Forms.PaintEventHandler(this.canvas_Paint);
//      //
//      // label1
//      //
//      this.label1.AutoSize = true;
//      this.label1.BackColor = System.Drawing.Color.Transparent;
//      this.label1.Font = new System.Drawing.Font("Microsoft Sans
Serif", 24F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(177)));
//      this.label1.Location = new System.Drawing.Point(683, 107);
//      this.label1.Name = "label1";
//      this.label1.Size = new System.Drawing.Size(115, 37);
//      this.label1.TabIndex = 1;
//      this.label1.Text = "Score:";
//      //
//      // labelGameOver
//      //
//      this.labelGameOver.AutoSize = true;
//      this.labelGameOver.BackColor = System.Drawing.Color.Transparent;
//      this.labelGameOver.Font = new System.Drawing.Font("Microsoft Sans
Serif", 24F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(177)));
//      this.labelGameOver.Location = new System.Drawing.Point(683, 229);
//      this.labelGameOver.Name = "labelGameOver";
//      this.labelGameOver.Size = new System.Drawing.Size(190, 37);
//      this.labelGameOver.TabIndex = 3;
//      this.labelGameOver.Text = "Game Over";
//      //
//      // labelScore
//      //
//      this.labelScore.AutoSize = true;
//      this.labelScore.BackColor = System.Drawing.Color.Transparent;
//      this.labelScore.Font = new System.Drawing.Font("Microsoft Sans
Serif", 24F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(177)));
//      this.labelScore.Location = new System.Drawing.Point(683, 166);
//      this.labelScore.Name = "labelScore";
//      this.labelScore.Size = new System.Drawing.Size(194, 37);
//      this.labelScore.TabIndex = 2;
//      this.labelScore.Text = "count score";
//      //
//      // menuStrip1
//      //
//      this.menuStrip1.Items.AddRange(new
System.Windows.Forms.ToolStripItem[] {
//      this.fileToolStripMenuItem,
//      this.helpToolStripMenuItem});
//      this.menuStrip1.Location = new System.Drawing.Point(0, 0);
//      this.menuStrip1.Name = "menuStrip1";
//      this.menuStrip1.Size = new System.Drawing.Size(979, 24);
//      this.menuStrip1.TabIndex = 4;
//      this.menuStrip1.Text = "menuStrip1";
//      //
//      // fileToolStripMenuItem
//      //

```

```

//          this.fileToolStripMenuItem.DropDownItems.AddRange(new
System.Windows.Forms.ToolStripItem[] {
//          this.exitToolStripMenuItem});
//          this.fileToolStripMenuItem.Name = "fileToolStripMenuItem";
//          this.fileToolStripMenuItem.Size = new System.Drawing.Size(37,
20);
//          this.fileToolStripMenuItem.Text = "File";
//          //
//          // exitToolStripMenuItem
//          //
//          this.exitToolStripMenuItem.Name = "exitToolStripMenuItem";
//          this.exitToolStripMenuItem.Size = new System.Drawing.Size(92,
22);
//          this.exitToolStripMenuItem.Text = "Exit";
//          this.exitToolStripMenuItem.Click += new
System.EventHandler(this.exitToolStripMenuItem_Click);
//          //
//          // helpToolStripMenuItem
//          //
//          this.helpToolStripMenuItem.DropDownItems.AddRange(new
System.Windows.Forms.ToolStripItem[] {
//          this.howToPlayToolStripMenuItem,
//          this.aboutToolStripMenuItem});
//          this.helpToolStripMenuItem.Name = "helpToolStripMenuItem";
//          this.helpToolStripMenuItem.Size = new System.Drawing.Size(44,
20);
//          this.helpToolStripMenuItem.Text = "Help";
//          //
//          // howToPlayToolStripMenuItem
//          //
//          this.howToPlayToolStripMenuItem.Name =
"howToPlayToolStripMenuItem";
//          this.howToPlayToolStripMenuItem.Size = new
System.Drawing.Size(138, 22);
//          this.howToPlayToolStripMenuItem.Text = "How to play";
//          this.howToPlayToolStripMenuItem.Click += new
System.EventHandler(this.howToPlayToolStripMenuItem_Click);
//          //
//          // aboutToolStripMenuItem
//          //
//          this.aboutToolStripMenuItem.Name = "aboutToolStripMenuItem";
//          this.aboutToolStripMenuItem.Size = new System.Drawing.Size(138,
22);
//          this.aboutToolStripMenuItem.Text = "About";
//          this.aboutToolStripMenuItem.Click += new
System.EventHandler(this.aboutToolStripMenuItem_Click);
//          //
//          // Form1
//          //
//          this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
//          this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
//          this.BackColor = System.Drawing.Color.LimeGreen;
//          this.BackgroundImage =
((System.Drawing.Image)(resources.GetObject("$this.BackgroundImage")));
//          this.ClientSize = new System.Drawing.Size(979, 581);
//          this.Controls.Add(this.labelGameOver);
//          this.Controls.Add(this.labelScore);
//          this.Controls.Add(this.label1);
//          this.Controls.Add(this.canvas);
//          this.Controls.Add(this.menuStrip1);
//          this.MainMenuStrip = this.menuStrip1;
//          this.Name = "Form1";

```

```

//          this.Text = "Snake";
//          this.KeyUp += new
System.Windows.Forms.KeyEventHandler(this.Form1_KeyUp);
//          this.KeyDown += new
System.Windows.Forms.KeyEventHandler(this.Form1_KeyDown);
//
((System.ComponentModel.ISupportInitialize)(this.canvas)).EndInit();
//          this.menuStrip1.ResumeLayout(false);
//          this.menuStrip1.PerformLayout();
//          this.ResumeLayout(false);
//          this.PerformLayout();

//      }

//      #endregion

//      private System.Windows.Forms.PictureBox canvas;
//      private System.Windows.Forms.Label label1;
//      private System.Windows.Forms.Timer GameTimer;
//      private System.Windows.Forms.Label labelGameOver;
//      private System.Windows.Forms.Label labelScore;
//      private System.Windows.Forms.MenuStrip menuStrip1;
//      private System.Windows.Forms.ToolStripMenuItem fileToolStripMenuItem;
//      private System.Windows.Forms.ToolStripMenuItem exitToolStripMenuItem;
//      private System.Windows.Forms.ToolStripMenuItem helpToolStripMenuItem;
//      private System.Windows.Forms.ToolStripMenuItem
howToPlayToolStripMenuItem;
//      private System.Windows.Forms.ToolStripMenuItem
aboutToolStripMenuItem;
//      }
//  }
//  }

```

```

namespace Snake
{
    partial class Form1
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be
disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>

```

```

private void InitializeComponent()
{
    this.components = new System.ComponentModel.Container();
    System.ComponentModel.ComponentResourceManager resources = new
System.ComponentModel.ComponentResourceManager(typeof(Form1));
    this.canvas = new System.Windows.Forms.PictureBox();
    this.label1 = new System.Windows.Forms.Label();
    this.GameTimer = new System.Windows.Forms.Timer(this.components);
    this.labelGameOver = new System.Windows.Forms.Label();
    this.labelScore = new System.Windows.Forms.Label();
    this.menuStrip1 = new System.Windows.Forms.MenuStrip();
    this.fileToolStripMenuItem = new
System.Windows.Forms.ToolStripItem();
    this.exitToolStripMenuItem = new
System.Windows.Forms.ToolStripItem();
    this.restartToolStripMenuItem = new
System.Windows.Forms.ToolStripItem();
    this.helpToolStripMenuItem = new
System.Windows.Forms.ToolStripItem();
    this.howToPlayToolStripMenuItem = new
System.Windows.Forms.ToolStripItem();
    this.aboutToolStripMenuItem = new
System.Windows.Forms.ToolStripItem();
    this.GeneralTimer1 = new
System.Windows.Forms.Timer(this.components);
    this.label2 = new System.Windows.Forms.Label();
    this.TimeLabel1 = new System.Windows.Forms.Label();
    this.WarningBordersLable = new System.Windows.Forms.Label();

    ((System.ComponentModel.ISupportInitialize)(this.canvas)).BeginInit();
    this.menuStrip1.SuspendLayout();
    this.SuspendLayout();
    //
    // canvas
    //
    this.canvas.BackColor = System.Drawing.Color.LimeGreen;
    this.canvas.Location = new System.Drawing.Point(12, 44);
    this.canvas.Name = "canvas";
    this.canvas.Size = new System.Drawing.Size(560, 458);
    this.canvas.TabIndex = 0;
    this.canvas.TabStop = false;
    this.canvas.Paint += new
System.Windows.Forms.PaintEventHandler(this.canvas_Paint);
    //
    // label1
    //
    this.label1.AutoSize = true;
    this.label1.BackColor = System.Drawing.Color.Transparent;
    this.label1.Font = new System.Drawing.Font("Microsoft Sans Serif",
18F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(177)));
    this.label1.Location = new System.Drawing.Point(588, 99);
    this.label1.Name = "label1";
    this.label1.Size = new System.Drawing.Size(89, 29);
    this.label1.TabIndex = 1;
    this.label1.Text = "Score:";
    //
    // labelGameOver
    //
    this.labelGameOver.AutoSize = true;
    this.labelGameOver.BackColor = System.Drawing.Color.Transparent;

```

```

        this.labelGameOver.Font = new System.Drawing.Font("Microsoft Sans
Serif", 18F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(177)));
        this.labelGameOver.Location = new System.Drawing.Point(592, 229);
        this.labelGameOver.Name = "labelGameOver";
        this.labelGameOver.Size = new System.Drawing.Size(145, 29);
        this.labelGameOver.TabIndex = 3;
        this.labelGameOver.Text = "Game Over";
        //
        // labelScore
        //
        this.labelScore.AutoSize = true;
        this.labelScore.BackColor = System.Drawing.Color.Transparent;
        this.labelScore.Font = new System.Drawing.Font("Microsoft Sans
Serif", 18F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(177)));
        this.labelScore.Location = new System.Drawing.Point(588, 166);
        this.labelScore.Name = "labelScore";
        this.labelScore.Size = new System.Drawing.Size(148, 29);
        this.labelScore.TabIndex = 2;
        this.labelScore.Text = "count score";
        //
        // menuStrip1
        //
        this.menuStrip1.Items.AddRange(new
System.Windows.Forms.ToolStripItem[] {
        this.fileToolStripMenuItem,
        this.helpToolStripMenuItem});
        this.menuStrip1.Location = new System.Drawing.Point(0, 0);
        this.menuStrip1.Name = "menuStrip1";
        this.menuStrip1.Size = new System.Drawing.Size(1276, 24);
        this.menuStrip1.TabIndex = 4;
        this.menuStrip1.Text = "menuStrip1";
        //
        // fileToolStripMenuItem
        //
        this.fileToolStripMenuItem.DropDownItems.AddRange(new
System.Windows.Forms.ToolStripItem[] {
        this.exitToolStripMenuItem,
        this.restartToolStripMenuItem});
        this.fileToolStripMenuItem.Name = "fileToolStripMenuItem";
        this.fileToolStripMenuItem.Size = new System.Drawing.Size(37, 20);
        this.fileToolStripMenuItem.Text = "File";
        //
        // exitToolStripMenuItem
        //
        this.exitToolStripMenuItem.Name = "exitToolStripMenuItem";
        this.exitToolStripMenuItem.Size = new System.Drawing.Size(110, 22);
        this.exitToolStripMenuItem.Text = "Exit";
        this.exitToolStripMenuItem.Click += new
System.EventHandler(this.exitToolStripMenuItem_Click);
        //
        // restartToolStripMenuItem
        //
        this.restartToolStripMenuItem.Name = "restartToolStripMenuItem";
        this.restartToolStripMenuItem.Size = new System.Drawing.Size(110,
22);
        this.restartToolStripMenuItem.Text = "Restart";
        this.restartToolStripMenuItem.Click += new
System.EventHandler(this.restartToolStripMenuItem_Click);
        //
        // helpToolStripMenuItem

```

```

        //
        this.helpToolStripMenuItem.DropDownItems.AddRange(new
System.Windows.Forms.ToolStripItem[] {
        this.howToPlayToolStripMenuItem,
        this.aboutToolStripMenuItem});
        this.helpToolStripMenuItem.Name = "helpToolStripMenuItem";
        this.helpToolStripMenuItem.Size = new System.Drawing.Size(44, 20);
        this.helpToolStripMenuItem.Text = "Help";
        //
        // howToPlayToolStripMenuItem
        //
        this.howToPlayToolStripMenuItem.Name =
"howToPlayToolStripMenuItem";
        this.howToPlayToolStripMenuItem.Size = new System.Drawing.Size(138,
22);
        this.howToPlayToolStripMenuItem.Text = "How to play";
        this.howToPlayToolStripMenuItem.Click += new
System.EventHandler(this.howToPlayToolStripMenuItem_Click);
        //
        // aboutToolStripMenuItem
        //
        this.aboutToolStripMenuItem.Name = "aboutToolStripMenuItem";
        this.aboutToolStripMenuItem.Size = new System.Drawing.Size(138,
22);
        this.aboutToolStripMenuItem.Text = "About";
        this.aboutToolStripMenuItem.Click += new
System.EventHandler(this.aboutToolStripMenuItem_Click);
        //
        // GeneralTimer1
        //
        this.GeneralTimer1.Tick += new
System.EventHandler(this.GeneralTimer1_Tick);
        //
        // label2
        //
        this.label2.AutoSize = true;
        this.label2.BackColor = System.Drawing.Color.Transparent;
        this.label2.Font = new System.Drawing.Font("Microsoft Sans Serif",
18F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(177)));
        this.label2.Location = new System.Drawing.Point(588, 44);
        this.label2.Name = "label2";
        this.label2.Size = new System.Drawing.Size(80, 29);
        this.label2.TabIndex = 6;
        this.label2.Text = "Time:";
        //
        // TimeLabel1
        //
        this.TimeLabel1.AutoSize = true;
        this.TimeLabel1.BackColor = System.Drawing.Color.Transparent;
        this.TimeLabel1.Font = new System.Drawing.Font("Microsoft Sans
Serif", 18F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(177)));
        this.TimeLabel1.Location = new System.Drawing.Point(674, 44);
        this.TimeLabel1.Name = "TimeLabel1";
        this.TimeLabel1.Size = new System.Drawing.Size(139, 29);
        this.TimeLabel1.TabIndex = 7;
        this.TimeLabel1.Text = "HH:MM:SS";
        //
        // WarningBordersLable
        //
        this.WarningBordersLable.AutoSize = true;

```



```

        this.WarningBordersLable.BackColor =
System.Drawing.Color.Transparent;
        this.WarningBordersLable.Font = new System.Drawing.Font("Microsoft
Sans Serif", 27.75F, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((byte)(177)));
        this.WarningBordersLable.Location = new System.Drawing.Point(51,
514);
        this.WarningBordersLable.Name = "WarningBordersLable";
        this.WarningBordersLable.Size = new System.Drawing.Size(461, 42);
        this.WarningBordersLable.TabIndex = 8;
        this.WarningBordersLable.Text = "Don\'t touch the borders!!!";
        //
        // Form1
        //
        this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
        this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
        this.BackColor = System.Drawing.Color.LimeGreen;
        this.BackgroundImage =
((System.Drawing.Image)(resources.GetObject("$this.BackgroundImage")));
        this.ClientSize = new System.Drawing.Size(1276, 741);
        this.Controls.Add(this.WarningBordersLable);
        this.Controls.Add(this.TimeLable1);
        this.Controls.Add(this.label12);
        this.Controls.Add(this.labelGameOver);
        this.Controls.Add(this.labelScore);
        this.Controls.Add(this.label1);
        this.Controls.Add(this.canvas);
        this.Controls.Add(this.menuStrip1);
        this.MainMenuStrip = this.menuStrip1;
        this.Name = "Form1";
        this.Text = "Snake";
        this.KeyDown += new
System.Windows.Forms.KeyEventHandler(this.Form1_KeyDown);
        this.KeyUp += new
System.Windows.Forms.KeyEventHandler(this.Form1_KeyUp);

        ((System.ComponentModel.ISupportInitialize)(this.canvas)).EndInit();
        this.menuStrip1.ResumeLayout(false);
        this.menuStrip1.PerformLayout();
        this.ResumeLayout(false);
        this.PerformLayout();

    }

    #endregion

    private System.Windows.Forms.PictureBox canvas;
    private System.Windows.Forms.Label label1;
    private System.Windows.Forms.Timer GameTimer;
    private System.Windows.Forms.Label labelGameOver;
    private System.Windows.Forms.Label labelScore;
    private System.Windows.Forms.MenuStrip menuStrip1;
    private System.Windows.Forms.ToolStripMenuItem fileToolStripMenuItem;
    private System.Windows.Forms.ToolStripMenuItem exitToolStripMenuItem;
    private System.Windows.Forms.ToolStripMenuItem helpToolStripMenuItem;
    private System.Windows.Forms.ToolStripMenuItem
howToPlayToolStripMenuItem;
    private System.Windows.Forms.ToolStripMenuItem aboutToolStripMenuItem;
    private System.Windows.Forms.ToolStripMenuItem
restartToolStripMenuItem;
    private System.Windows.Forms.Timer GeneralTimer1;
    private System.Windows.Forms.Label label12;

```

```
        private System.Windows.Forms.Label TimeLable1;  
        private System.Windows.Forms.Label WarningBordersLable;  
    }  
}
```

Form2.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Snake
{
    public partial class Form2 : Form
    {
        private List<Circle> snake1 = new List<Circle>();
        private List<Circle> snake2 = new List<Circle>();
        private Circle food1 = new Circle();
        private Circle food2 = new Circle();
        int score1 = 0;
        int score2 = 0;
        bool Pause1 = false;
        bool Pause2 = false;
        int minutes = 0, hours = 0;
        double seconds = -0.5;
        string time, gameOver1, gameOver2;
        bool CheckColssionWithGameBorders1 = false;
        bool CheckColssionWithGameBorders2 = false;

        public Form2()
        {
            InitializeComponent();

            //Set settings to default
            new Settings();

            //Set game speed and start timer
            GameTimer.Interval = 800 / Settings.Speed;
            GameTimer.Tick += UpdateScreens;
            GameTimer.Start();

            //Start New game
            StartGame();

            //general timer
            GeneralTimer2.Interval = 1000;
            GeneralTimer2.Enabled = true;
            GeneralTimer2_Tick(null, null);
            GeneralTimer2.Tick += new EventHandler(GeneralTimer2_Tick);
            GeneralTimer2.Start();
        }

        private void GeneralTimer2_Tick(object sender, EventArgs e)
        {
            seconds += 0.5;
            if (seconds == 60) { seconds = 0; minutes++; }
            if (minutes == 60) { minutes = 0; hours++; }
            if (hours == 10) { this.Close(); }
        }
    }
}
```

```

        string s;
        if (seconds < 10) { s = "0" + seconds; }
        else { s = "" + seconds; }
        string m;
        if (minutes < 10) { m = "0" + minutes; }
        else { m = "" + minutes; }
        string h;
        if (hours < 10) { h = "0" + hours; }
        else { h = "" + hours; }
        time = h + ":" + m + ":" + s;
        TimeLable2.Text = time;
    }
    private void restartToolStripMenuItem_Click(object sender, EventArgs e)
    {
        StartGame();
        Pause1 = false;
        Pause2 = false;
        TheDrawLable.Visible = false;
        score1 = 0;
        score2 = 0;
    }

    private void exitToolStripMenuItem_Click_1(object sender, EventArgs e)
    {
        Application.Exit();
    }

    private void howToPlToolStripMenuItem_Click(object sender, EventArgs e)
    {
        MessageBox.Show("Use the arrow keys to move the first snake, and  
the keys: 'A','S','W','D' to move the second snake. 'H' to increase the speed,  
'L' to decrease the speed, 'M' to turning back to default speed. 'Space' to  
restart. 'Esc' to exit. The player who has the highest score - wins.", "Game  
instructions");
    }

    private void aboutToolStripMenuItem_Click_1(object sender, EventArgs e)
    {
        MessageBox.Show("Made by Ben-El, 2015 @", "Snake - About");
    }

    private void StartGame()
    {
        WarningBordersLable1.Text = "Don't touch the borders!!!";
        WarningBordersLable2.Text = "Don't touch the borders!!!";
        labelGameOver1.Visible = false;
        labelGameOver2.Visible = false;
        LoserLable1.Visible = false;
        LoserLable2.Visible = false;
        //Set settings to default
        new Settings();
        GameTimer.Interval = 800 / Settings.Speed; //Set speed to the
regular speed

        //Create new player object
        snake1.Clear();
        Circle head1 = new Circle { X = 10, Y = 10 };
        snake1.Add(head1);

        snake2.Clear();
        Circle head2 = new Circle { X = 10, Y = 10 };
        snake2.Add(head2);
    }

```

```

        //count score

        labelScore1.Text = Settings.Score1.ToString();
        labelScore2.Text = Settings.Score1.ToString();
        GenerateFood1();
        GenerateFood2();

    }

    //Place random food object
    private void GenerateFood1()
    {
        int maxXPos = canvas1.Size.Width / Settings.Width;
        int maxYPos = canvas1.Size.Height / Settings.Height;
        Random random1 = new Random();
        food1 = new Circle { X = random1.Next(0, maxXPos), Y =
random1.Next(0, maxYPos) };
    }

    private void GenerateFood2()
    {
        int maxXPos = canvas2.Size.Width / Settings.Width;
        int maxYPos = canvas2.Size.Height / Settings.Height;
        Random random2 = new Random();
        food2 = new Circle { X = random2.Next(0, maxXPos), Y =
random2.Next(0, maxYPos) };
    }

    private void UpdateScreens(object sender, EventArgs e)
    {
        //Check for Game Over
        if (Settings.GameOver == true)
        {
            //Check if Space is pressed
            if (Input.KeyPressed(Keys.Space))
            {
                //new Settings();
                StartGame();
            }
        }
        else
        {
            //if (Input.KeyPressed(Keys.P))
            //{
            //    Thread.Sleep(100);
            //    Pause = !Pause; // pause state is changed
            //}

            if (Pause1 == false)
            {
                if (Input.KeyPressed(Keys.Right) && Settings.direction1 !=
Direction.Left1)
                    Settings.direction1 = Direction.Right1;
                else if (Input.KeyPressed(Keys.Left) && Settings.direction1
!= Direction.Right1)
                    Settings.direction1 = Direction.Left1;
                else if (Input.KeyPressed(Keys.Up) && Settings.direction1
!= Direction.Down1)
                    Settings.direction1 = Direction.Up1;
            }
        }
    }

```

```

        else if (Input.KeyPressed(Keys.Down) && Settings.direction1
!= Direction.Up1)
            Settings.direction1 = Direction.Down1;
            MovePlayer1();
        }
        if (Pause2 == false)
        {
            if (Input.KeyPressed(Keys.D) && Settings.direction2 !=
Direction.Left2)
                Settings.direction2 = Direction.Right2;
            else if (Input.KeyPressed(Keys.A) && Settings.direction2 !=
Direction.Right2)
                Settings.direction2 = Direction.Left2;
            else if (Input.KeyPressed(Keys.W) && Settings.direction2 !=
Direction.Down2)
                Settings.direction2 = Direction.Up2;
            else if (Input.KeyPressed(Keys.S) && Settings.direction2 !=
Direction.Up2)
                Settings.direction2 = Direction.Down2;
            MovePlayer2();
        }
    }
    if (Input.KeyPressed(Keys.H))
    {
        GameTimer.Interval = 300 / Settings.Speed; //increases the speed
    }
    if (Input.KeyPressed(Keys.M))
    {
        GameTimer.Interval = 800 / Settings.Speed; //regular speed
    }
    if (Input.KeyPressed(Keys.L))
    {
        GameTimer.Interval = 1500 / Settings.Speed; //decreases the
speed
    }
    if (Input.KeyPressed(Keys.Space))
    {
        StartGame();
        Pause1 = false;
        Pause2 = false;
        TheDrawLable.Visible = false;
        score1 = 0;
        score2 = 0;
        seconds = -1;
        minutes = 0;
        hours = 0;
        GeneralTimer2.Start();
    }
    if (Settings.Score1 != 0 && Settings.Score1 % 500 == 0)
    {
        WarningBordersLable1.Visible = true;
        canvas1.BackColor = System.Drawing.Color.CornflowerBlue;
        ColssionWithGameBorders1();
    }
    else
    {
        WarningBordersLable1.Visible = false;
        canvas1.BackColor = System.Drawing.Color.LimeGreen;
    }
    if (Settings.Score2 != 0 && Settings.Score2 % 500 == 0)

```

```

    {
        WarningBordersLable2.Visible = true;
        canvas2.BackColor = System.Drawing.Color.CornflowerBlue;
        ColssionWithGameBorders2();
    }
    else
    {
        WarningBordersLable2.Visible = false;
        canvas2.BackColor = System.Drawing.Color.LimeGreen;
    }
    if (((seconds != 0 && seconds % 30 == 0) || (minutes > 0 && minutes
% 1 == 0 && seconds == 0)) && (Pause1 == false && Pause2 == false))
    {
        Shake(this);
    }

    if (Input.KeyPressed(Keys.Escape))
    {
        Application.Exit();
    }

    canvas1.Invalidate();
    canvas2.Invalidate();
}

private void canvas1_Paint(object sender, PaintEventArgs e)
{
    Graphics canvas1 = e.Graphics;

    if (!Settings.GameOver)
    {
        //Set colour of snake

        //Draw snake
        for (int i = 0; i < snake1.Count; i++)
        {
            Brush snakeColour;
            if (i == 0)
                snakeColour = Brushes.Black;        //Draw head
            else
                snakeColour = Brushes.DarkGreen;    //Rest of body

            //Draw snake
            canvas1.FillEllipse(snakeColour,
                new Rectangle(snake1[i].X * Settings.Width,
                    snake1[i].Y * Settings.Height,
                        Settings.Width, Settings.Height));

            //Draw Food
            canvas1.FillEllipse(Brushes.Red,
                new Rectangle(food1.X * Settings.Width,
                    food1.Y * Settings.Height, Settings.Width,
Settings.Height));
        }
    }
}

```

```

private void canvas2_Paint(object sender, PaintEventArgs e)
{
    Graphics canvas2 = e.Graphics;

    if (!Settings.GameOver)
    {
        //Set colour of snake

        //Draw snake
        for (int k = 0; k < snake2.Count; k++)
        {
            Brush snakeColour;
            if (k == 0)
                snakeColour = Brushes.Black;    //Draw head
            else
                snakeColour = Brushes.DarkGreen; //Rest of body

            //Draw snake
            canvas2.FillEllipse(snakeColour,
                new Rectangle(snake2[k].X * Settings.Width,
                    snake2[k].Y * Settings.Height,
                    Settings.Width, Settings.Height));

            //Draw Food
            canvas2.FillEllipse(Brushes.Red,
                new Rectangle(food2.X * Settings.Width,
                    food2.Y * Settings.Height, Settings.Width,
Settings.Height));
        }
    }
}

private void MovePlayer1()
{
    for (int i = snake1.Count - 1; i >= 0; i--)
    {
        //Move head
        if (i == 0)
        {
            switch (Settings.direction1)
            {
                case Direction.Right1:
                    snake1[i].X++;
                    break;
                case Direction.Left1:
                    snake1[i].X--;
                    break;
                case Direction.Up1:
                    snake1[i].Y--;
                    break;
                case Direction.Down1:
                    snake1[i].Y++;
                    break;
            }

            //Get maximum X and Y Pos

```



```

int maxXPos = canvas1.Size.Width / Settings.Width;
int maxYPos = canvas1.Size.Height / Settings.Height;

if (snake1[i].X < 0)
    snake1[i].X = maxXPos;
if (snake1[i].Y < 0)
    snake1[i].Y = maxYPos;
if (snake1[i].X > maxXPos)
    snake1[i].X = 0;
if (snake1[i].Y > maxYPos)
    snake1[i].Y = 0;

//Detect collision with body
for (int j = 1; j < snake1.Count; j++)
{
    if (snake1[i].X == snake1[j].X &&
        snake1[i].Y == snake1[j].Y)
    {
        Die1();
    }
    if (Pause2 == true && Pause1 == true)
    {
        GeneralTimer2.Stop();
    }
}

//Detect collision with food pieces
if (snake1[0].X == food1.X && snake1[0].Y == food1.Y)
{
    Eat1();
}
}

else
{
    //Move body
    snake1[i].X = snake1[i - 1].X;
    snake1[i].Y = snake1[i - 1].Y;
}
}
}

private void MovePlayer2()
{
    for (int k = snake2.Count - 1; k >= 0; k--)
    {
        //Move head
        if (k == 0)
        {
            switch (Settings.direction2)
            {
                case Direction.Right2:
                    snake2[k].X++;
                    break;
                case Direction.Left2:
                    snake2[k].X--;
                    break;
                case Direction.Up2:
                    snake2[k].Y--;
                    break;
            }
        }
    }
}

```

```

        case Direction.Down2:
            snake2[k].Y++;
            break;
    }

    //Get maximum X and Y Pos
    int maxCPos = canvas2.Size.Width / Settings.Width;
    int maxDPos = canvas2.Size.Height / Settings.Height;

    if (snake2[k].X < 0)
        snake2[k].X = maxCPos;
    if (snake2[k].Y < 0)
        snake2[k].Y = maxDPos;
    if (snake2[k].X > maxCPos)
        snake2[k].X = 0;
    if (snake2[k].Y > maxDPos)
        snake2[k].Y = 0;

    //Detect collision with body
    for (int m = 1; m < snake2.Count; m++)
    {
        if (snake2[k].X == snake2[m].X &&
            snake2[k].Y == snake2[m].Y)
        {
            Die2();
        }
        if (Pause2 == true && Pause1 == true)
        {
            GeneralTimer2.Stop();
        }
    }

    //Detect collision with food pieces
    if (snake2[0].X == food2.X && snake2[0].Y == food2.Y)
    {
        Eat2();
    }
}

else
{
    //Move body
    snake2[k].X = snake2[k - 1].X;
    snake2[k].Y = snake2[k - 1].Y;
}
}

private void ColssionWithGameBorders1()
{
    int max1XPos = canvas1.Size.Width / Settings.Width;
    int max1YPos = canvas1.Size.Height / Settings.Height;
    for (int i = snake1.Count - 1; i >= 0; i--)
    {
        //Detect collision with game borders

        if (snake1[i].X < 0 || snake1[i].Y < 0
            || snake1[i].X >= max1XPos || snake1[i].Y >= max1YPos)
    }
}

```

```

        {
            CheckColssionWithGameBorders1 = true;
            if (CheckColssionWithGameBorders1 == true)
            {
                WarningBordersLable1.Text = "I warned you...";
                Die1();
            }
        }
    }

private void ColssionWithGameBorders2()
{
    int max2XPos = canvas2.Size.Width / Settings.Width;
    int max2YPos = canvas2.Size.Height / Settings.Height;
    for (int i = snake2.Count - 1; i >= 0; i--)
    {
        //Detect collission with game borders

        if (snake2[i].X < 0 || snake2[i].Y < 0
            || snake2[i].X >= max2XPos || snake2[i].Y >= max2YPos)
        {
            CheckColssionWithGameBorders2 = true;
            if (CheckColssionWithGameBorders2 == true)
            {
                WarningBordersLable2.Text = "I warned you...";
                Die2();
            }
        }
    }
}

private void Form2_KeyDown(object sender, KeyEventArgs e)
{
    Input.ChangeState(e.KeyCode, true);
}

private void Form2_KeyUp(object sender, KeyEventArgs e)
{
    Input.ChangeState(e.KeyCode, false);
}

private void Eat1()
{
    CheckForLoser();
    //Add circle to body
    Circle Foodcircle1 = new Circle
    {
        X = snake1[snake1.Count - 1].X,
        Y = snake1[snake1.Count - 1].Y
    };
    snake1.Add(Foodcircle1);

    //Update Score
    Settings.Score1 += Settings.Points;
    score1 = Settings.Score1;
    labelScore1.Text = Settings.Score1.ToString();
    GenerateFood1();
}

```

```

private void Eat2()
{
    CheckForLoser();
    //Add circle to body
    Circle Foodcircle2 = new Circle
    {
        X = snake2[snake2.Count - 1].X,
        Y = snake2[snake2.Count - 1].Y
    };
    snake2.Add(Foodcircle2);

    //Update Score
    Settings.Score2 += Settings.Points;
    score2 = Settings.Score2;
    labelScore2.Text = Settings.Score2.ToString();
    GenerateFood2();
}

private void CheckForLoser()
{
    int CheckLoser = Math.Min(score1, score2);
    if (Pause1 && !Pause2)
    {
        LoserLable1.Visible = false;
        LoserLable2.Visible = false;
        if (CheckLoser == score1)
        {
            LoserLable1.Visible = true;
        }
        gameOver1 = "The game is over for you...! \n Your final score
is: "
        + Settings.Score1 + " ." + "\n Wait until your opponent is
done...";
        labelGameOver1.Text = gameOver1;
        labelGameOver1.Visible = true;
    }
    if (!Pause1 && Pause2)
    {
        LoserLable1.Visible = false;
        LoserLable2.Visible = false;
        if (CheckLoser == score2)
        {
            LoserLable2.Visible = true;
        }
        gameOver2 = "The game is over for you...! \n Your final score
is: "
        + Settings.Score2 + " ." + "\n Wait until your opponent is
done...";
        labelGameOver2.Text = gameOver2;
        labelGameOver2.Visible = true;
    }
    if (Pause1 && Pause2)
    {
        GeneralTimer2.Stop();
        LoserLable1.Visible = false;
        LoserLable2.Visible = false;
        if (CheckLoser == score2)
        {
            LoserLable2.Visible = true;
        }
        if (CheckLoser == score1)
        {

```

```

        LoserLable1.Visible = true;
    }
    if (CheckLoser == score1 && CheckLoser == score2)
    {
        TheDrawLable.Visible = true;
        LoserLable1.Visible = false;
        LoserLable2.Visible = false;
    }
    gameOver1 = "Game over! \n Your final score is: " +
Settings.Score1 + " ." + "\n Press Space to try again...";
    labelGameOver1.Text = gameOver1;
    labelGameOver1.Visible = true;

    gameOver2 = "Game over! \n Your final score is: " +
Settings.Score2 + " ." + "\n Press Space to try again...";
    labelGameOver2.Text = gameOver2;
    labelGameOver2.Visible = true;
    }
}

private void Die1()
{
    Pause1 = true;
    CheckForLoser();
}
private void Die2()
{
    Pause2 = true;
    CheckForLoser();
}

private void Shake(Form form)
{
    var original = form.Location;
    Random rnd = new Random();
    const int shake_amplitude = 5;
    for (int i = 0; i < 10; i++)
    {
        form.Location = new Point(original.X + rnd.Next(-
shake_amplitude, shake_amplitude), original.Y + rnd.Next(-shake_amplitude,
shake_amplitude));
    }
}
}
}
}

```

-

Form2Designer.cs:

```
namespace Snake
{
    partial class Form2
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be
disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            System.ComponentModel.ComponentResourceManager resources = new
System.ComponentModel.ComponentResourceManager(typeof(Form2));
            this.menuStrip2 = new System.Windows.Forms.MenuStrip();
            this.fileToolStripMenuItem = new
System.Windows.Forms.ToolStripItem();
            this.exitToolStripMenuItem = new
System.Windows.Forms.ToolStripItem();
            this.restartToolStripMenuItem = new
System.Windows.Forms.ToolStripItem();
            this.helpToolStripMenuItem = new
System.Windows.Forms.ToolStripItem();
            this.howToPlToolStripMenuItem = new
System.Windows.Forms.ToolStripItem();
            this.aboutToolStripMenuItem = new
System.Windows.Forms.ToolStripItem();
            this.GameTimer = new System.Windows.Forms.Timer(this.components);
            this.canvas1 = new System.Windows.Forms.PictureBox();
            this.ScoreLabel1 = new System.Windows.Forms.Label();
            this.labelScore1 = new System.Windows.Forms.Label();
            this.labelGameOver1 = new System.Windows.Forms.Label();
            this.canvas2 = new System.Windows.Forms.PictureBox();
            this.ScoreLabel2 = new System.Windows.Forms.Label();
            this.labelScore2 = new System.Windows.Forms.Label();
        }
    }
}
```

```

        this.labelGameOver2 = new System.Windows.Forms.Label();
        this.LoserLable1 = new System.Windows.Forms.Label();
        this.LoserLable2 = new System.Windows.Forms.Label();
        this.TimeLable2 = new System.Windows.Forms.Label();
        this.lable2 = new System.Windows.Forms.Label();
        this.GeneralTimer2 = new
System.Windows.Forms.Timer(this.components);
        this.TheDrawLable = new System.Windows.Forms.Label();
        this.WarningBordersLable1 = new System.Windows.Forms.Label();
        this.WarningBordersLable2 = new System.Windows.Forms.Label();
        this.menuStrip2.SuspendLayout();

        ((System.ComponentModel.ISupportInitialize)(this.canvas1)).BeginInit();

        ((System.ComponentModel.ISupportInitialize)(this.canvas2)).BeginInit();
        this.SuspendLayout();
        //
        // menuStrip2
        //
        this.menuStrip2.Items.AddRange(new
System.Windows.Forms.ToolStripItem[] {
        this.fileToolStripMenuItem,
        this.helpToolStripMenuItem});
        this.menuStrip2.Location = new System.Drawing.Point(0, 0);
        this.menuStrip2.Name = "menuStrip2";
        this.menuStrip2.Size = new System.Drawing.Size(1276, 24);
        this.menuStrip2.TabIndex = 0;
        this.menuStrip2.Text = "menuStrip2";
        //
        // fileToolStripMenuItem
        //
        this.fileToolStripMenuItem.DropDownItems.AddRange(new
System.Windows.Forms.ToolStripItem[] {
        this.exitToolStripMenuItem,
        this.restartToolStripMenuItem});
        this.fileToolStripMenuItem.Name = "fileToolStripMenuItem";
        this.fileToolStripMenuItem.Size = new System.Drawing.Size(37, 20);
        this.fileToolStripMenuItem.Text = "File";
        //
        // exitToolStripMenuItem
        //
        this.exitToolStripMenuItem.Name = "exitToolStripMenuItem";
        this.exitToolStripMenuItem.Size = new System.Drawing.Size(152, 22);
        this.exitToolStripMenuItem.Text = "Exit";
        this.exitToolStripMenuItem.Click += new
System.EventHandler(this.exitToolStripMenuItem_Click_1);
        //
        // restartToolStripMenuItem
        //
        this.restartToolStripMenuItem.Name = "restartToolStripMenuItem";
        this.restartToolStripMenuItem.Size = new System.Drawing.Size(152,
22);
        this.restartToolStripMenuItem.Text = "Restart";
        this.restartToolStripMenuItem.Click += new
System.EventHandler(this.restartToolStripMenuItem_Click);
        //
        // helpToolStripMenuItem
        //
        this.helpToolStripMenuItem.DropDownItems.AddRange(new
System.Windows.Forms.ToolStripItem[] {
        this.howToPlToolStripMenuItem,
        this.aboutToolStripMenuItem});

```

```

        this.helpToolStripMenuItem.Name = "helpToolStripMenuItem";
        this.helpToolStripMenuItem.Size = new System.Drawing.Size(44, 20);
        this.helpToolStripMenuItem.Text = "Help";
        //
        // howToPlToolStripMenuItem
        //
        this.howToPlToolStripMenuItem.Name = "howToPlToolStripMenuItem";
        this.howToPlToolStripMenuItem.Size = new System.Drawing.Size(138,
22);
        this.howToPlToolStripMenuItem.Text = "How to play";
        this.howToPlToolStripMenuItem.Click += new
System.EventHandler(this.howToPlToolStripMenuItem_Click);
        //
        // aboutToolStripMenuItem
        //
        this.aboutToolStripMenuItem.Name = "aboutToolStripMenuItem";
        this.aboutToolStripMenuItem.Size = new System.Drawing.Size(138,
22);
        this.aboutToolStripMenuItem.Text = "About";
        this.aboutToolStripMenuItem.Click += new
System.EventHandler(this.aboutToolStripMenuItem_Click_1);
        //
        // canvas1
        //
        this.canvas1.BackColor = System.Drawing.Color.LimeGreen;
        this.canvas1.Location = new System.Drawing.Point(45, 78);
        this.canvas1.Name = "canvas1";
        this.canvas1.Size = new System.Drawing.Size(543, 305);
        this.canvas1.TabIndex = 1;
        this.canvas1.TabStop = false;
        this.canvas1.Paint += new
System.Windows.Forms.PaintEventHandler(this.canvas1_Paint);
        //
        // ScoreLable1
        //
        this.ScoreLable1.AutoSize = true;
        this.ScoreLable1.BackColor = System.Drawing.Color.Transparent;
        this.ScoreLable1.Font = new System.Drawing.Font("Microsoft Sans
Serif", 18F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(177)));
        this.ScoreLable1.Location = new System.Drawing.Point(630, 98);
        this.ScoreLable1.Name = "ScoreLable1";
        this.ScoreLable1.Size = new System.Drawing.Size(89, 29);
        this.ScoreLable1.TabIndex = 2;
        this.ScoreLable1.Text = "Score:";
        //
        // labelScore1
        //
        this.labelScore1.AutoSize = true;
        this.labelScore1.BackColor = System.Drawing.Color.Transparent;
        this.labelScore1.Font = new System.Drawing.Font("Microsoft Sans
Serif", 18F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(177)));
        this.labelScore1.Location = new System.Drawing.Point(630, 164);
        this.labelScore1.Name = "labelScore1";
        this.labelScore1.Size = new System.Drawing.Size(148, 29);
        this.labelScore1.TabIndex = 3;
        this.labelScore1.Text = "count score";
        //
        // labelGameOver1
        //
        this.labelGameOver1.AutoSize = true;

```



```

        this.labelGameOver1.BackColor = System.Drawing.Color.Transparent;
        this.labelGameOver1.Font = new System.Drawing.Font("Microsoft Sans
Serif", 18F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(177)));
        this.labelGameOver1.Location = new System.Drawing.Point(630, 243);
        this.labelGameOver1.Name = "labelGameOver1";
        this.labelGameOver1.Size = new System.Drawing.Size(145, 29);
        this.labelGameOver1.TabIndex = 4;
        this.labelGameOver1.Text = "Game Over";
        //
        // canvas2
        //
        this.canvas2.BackColor = System.Drawing.Color.LimeGreen;
        this.canvas2.Location = new System.Drawing.Point(44, 445);
        this.canvas2.Name = "canvas2";
        this.canvas2.Size = new System.Drawing.Size(543, 305);
        this.canvas2.TabIndex = 7;
        this.canvas2.TabStop = false;
        this.canvas2.Paint += new
System.Windows.Forms.PaintEventHandler(this.canvas2_Paint);
        //
        // ScoreLable2
        //
        this.ScoreLable2.AutoSize = true;
        this.ScoreLable2.BackColor = System.Drawing.Color.Transparent;
        this.ScoreLable2.Font = new System.Drawing.Font("Microsoft Sans
Serif", 18F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(177)));
        this.ScoreLable2.Location = new System.Drawing.Point(615, 473);
        this.ScoreLable2.Name = "ScoreLable2";
        this.ScoreLable2.Size = new System.Drawing.Size(89, 29);
        this.ScoreLable2.TabIndex = 8;
        this.ScoreLable2.Text = "Score:";
        //
        // labelScore2
        //
        this.labelScore2.AutoSize = true;
        this.labelScore2.BackColor = System.Drawing.Color.Transparent;
        this.labelScore2.Font = new System.Drawing.Font("Microsoft Sans
Serif", 18F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(177)));
        this.labelScore2.Location = new System.Drawing.Point(615, 544);
        this.labelScore2.Name = "labelScore2";
        this.labelScore2.Size = new System.Drawing.Size(148, 29);
        this.labelScore2.TabIndex = 9;
        this.labelScore2.Text = "count score";
        //
        // labelGameOver2
        //
        this.labelGameOver2.AutoSize = true;
        this.labelGameOver2.BackColor = System.Drawing.Color.Transparent;
        this.labelGameOver2.Font = new System.Drawing.Font("Microsoft Sans
Serif", 18F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(177)));
        this.labelGameOver2.Location = new System.Drawing.Point(615, 610);
        this.labelGameOver2.Name = "labelGameOver2";
        this.labelGameOver2.Size = new System.Drawing.Size(145, 29);
        this.labelGameOver2.TabIndex = 10;
        this.labelGameOver2.Text = "Game Over";
        //
        // LoserLable1
        //

```

```

        this.LoserLable1.AutoSize = true;
        this.LoserLable1.BackColor = System.Drawing.Color.Transparent;
        this.LoserLable1.Font = new System.Drawing.Font("Microsoft Sans
Serif", 24F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(177)));
        this.LoserLable1.Location = new System.Drawing.Point(248, 214);
        this.LoserLable1.Name = "LoserLable1";
        this.LoserLable1.Size = new System.Drawing.Size(112, 37);
        this.LoserLable1.TabIndex = 12;
        this.LoserLable1.Text = "Loser!";
        this.LoserLable1.Visible = false;
        //
        // LoserLable2
        //
        this.LoserLable2.AutoSize = true;
        this.LoserLable2.BackColor = System.Drawing.Color.Transparent;
        this.LoserLable2.Font = new System.Drawing.Font("Microsoft Sans
Serif", 24F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(177)));
        this.LoserLable2.Location = new System.Drawing.Point(248, 576);
        this.LoserLable2.Name = "LoserLable2";
        this.LoserLable2.Size = new System.Drawing.Size(112, 37);
        this.LoserLable2.TabIndex = 13;
        this.LoserLable2.Text = "Loser!";
        this.LoserLable2.Visible = false;
        //
        // TimeLable2
        //
        this.TimeLable2.AutoSize = true;
        this.TimeLable2.BackColor = System.Drawing.Color.Transparent;
        this.TimeLable2.Font = new System.Drawing.Font("Microsoft Sans
Serif", 18F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(177)));
        this.TimeLable2.Location = new System.Drawing.Point(716, 398);
        this.TimeLable2.Name = "TimeLable2";
        this.TimeLable2.Size = new System.Drawing.Size(139, 29);
        this.TimeLable2.TabIndex = 15;
        this.TimeLable2.Text = "HH:MM:SS";
        //
        // lable2
        //
        this.lable2.AutoSize = true;
        this.lable2.BackColor = System.Drawing.Color.Transparent;
        this.lable2.Font = new System.Drawing.Font("Microsoft Sans Serif",
18F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(177)));
        this.lable2.Location = new System.Drawing.Point(630, 398);
        this.lable2.Name = "lable2";
        this.lable2.Size = new System.Drawing.Size(80, 29);
        this.lable2.TabIndex = 14;
        this.lable2.Text = "Time:";
        //
        // GeneralTimer2
        //
        this.GeneralTimer2.Tick += new
System.EventHandler(this.GeneralTimer2_Tick);
        //
        // TheDrawLable
        //
        this.TheDrawLable.AutoSize = true;
        this.TheDrawLable.BackColor = System.Drawing.Color.Transparent;

```

```

        this.TheDrawLable.Font = new System.Drawing.Font("Microsoft Sans
Serif", 24F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(177)));
        this.TheDrawLable.Location = new System.Drawing.Point(202, 398);
        this.TheDrawLable.Name = "TheDrawLable";
        this.TheDrawLable.Size = new System.Drawing.Size(204, 37);
        this.TheDrawLable.TabIndex = 16;
        this.TheDrawLable.Text = "It\'s a draw!!!";
        this.TheDrawLable.Visible = false;
        //
        // WarningBordersLable1
        //
        this.WarningBordersLable1.AutoSize = true;
        this.WarningBordersLable1.BackColor =
System.Drawing.Color.Transparent;
        this.WarningBordersLable1.Font = new System.Drawing.Font("Microsoft
Sans Serif", 20.25F, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((byte)(177)));
        this.WarningBordersLable1.Location = new System.Drawing.Point(146,
44);
        this.WarningBordersLable1.Name = "WarningBordersLable1";
        this.WarningBordersLable1.Size = new System.Drawing.Size(345, 31);
        this.WarningBordersLable1.TabIndex = 17;
        this.WarningBordersLable1.Text = "Don\'t touch the borders!!!";
        //
        // WarningBordersLable2
        //
        this.WarningBordersLable2.AutoSize = true;
        this.WarningBordersLable2.BackColor =
System.Drawing.Color.Transparent;
        this.WarningBordersLable2.Font = new System.Drawing.Font("Microsoft
Sans Serif", 20.25F, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((byte)(177)));
        this.WarningBordersLable2.Location = new System.Drawing.Point(131,
762);
        this.WarningBordersLable2.Name = "WarningBordersLable2";
        this.WarningBordersLable2.Size = new System.Drawing.Size(345, 31);
        this.WarningBordersLable2.TabIndex = 18;
        this.WarningBordersLable2.Text = "Don\'t touch the borders!!!";
        //
        // Form2
        //
        this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
        this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
        this.BackgroundImage =
((System.Drawing.Image)(resources.GetObject("$this.BackgroundImage")));
        this.ClientSize = new System.Drawing.Size(1276, 928);
        this.Controls.Add(this.WarningBordersLable2);
        this.Controls.Add(this.WarningBordersLable1);
        this.Controls.Add(this.TheDrawLable);
        this.Controls.Add(this.TimeLable2);
        this.Controls.Add(this.lable2);
        this.Controls.Add(this.LoserLable2);
        this.Controls.Add(this.LoserLable1);
        this.Controls.Add(this.labelGameOver2);
        this.Controls.Add(this.labelScore2);
        this.Controls.Add(this.ScoreLable2);
        this.Controls.Add(this.canvas2);
        this.Controls.Add(this.labelGameOver1);
        this.Controls.Add(this.labelScore1);
        this.Controls.Add(this.ScoreLable1);
        this.Controls.Add(this.canvas1);

```

```

        this.Controls.Add(this.menuStrip2);
        this.MainMenuStrip = this.menuStrip2;
        this.Name = "Form2";
        this.Text = "Snake";
        this.KeyDown += new
System.Windows.Forms.KeyEventHandler(this.Form2_KeyDown);
        this.KeyUp += new
System.Windows.Forms.KeyEventHandler(this.Form2_KeyUp);
        this.menuStrip2.ResumeLayout(false);
        this.menuStrip2.PerformLayout();

((System.ComponentModel.ISupportInitialize)(this.canvas1)).EndInit();

((System.ComponentModel.ISupportInitialize)(this.canvas2)).EndInit();
        this.ResumeLayout(false);
        this.PerformLayout();

    }

    #endregion

    private System.Windows.Forms.MenuStrip menuStrip2;
    private System.Windows.Forms.Timer GameTimer;
    private System.Windows.Forms.PictureBox canvas1;
    private System.Windows.Forms.ToolStripMenuItem fileToolStripMenuItem;
    private System.Windows.Forms.ToolStripMenuItem exitToolStripMenuItem;
    private System.Windows.Forms.ToolStripMenuItem helpToolStripMenuItem;
    private System.Windows.Forms.ToolStripMenuItem
howToPlToolStripMenuItem;
    private System.Windows.Forms.ToolStripMenuItem aboutToolStripMenuItem;
    private System.Windows.Forms.Label ScoreLabel1;
    private System.Windows.Forms.Label labelScore1;
    private System.Windows.Forms.Label labelGameOver1;
    private System.Windows.Forms.PictureBox canvas2;
    private System.Windows.Forms.Label ScoreLabel2;
    private System.Windows.Forms.Label labelScore2;
    private System.Windows.Forms.Label labelGameOver2;
    private System.Windows.Forms.Label LoserLabel1;
    private System.Windows.Forms.Label LoserLabel2;
    private System.Windows.Forms.ToolStripMenuItem
restartToolStripMenuItem;
    private System.Windows.Forms.Label TimeLabel2;
    private System.Windows.Forms.Label labl2;
    private System.Windows.Forms.Timer GeneralTimer2;
    private System.Windows.Forms.Label TheDrawLabel;
    private System.Windows.Forms.Label WarningBordersLabel1;
    private System.Windows.Forms.Label WarningBordersLabel2;
    }
}

```

Circle.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Snake
{
    class Circle
    {
        public int X { get; set; }
        public int Y { get; set; }

        public Circle1()
        {
            X = 0;
            Y = 0;
        }
    }
}
```

Input.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Collections;
using System.Windows.Forms;

namespace Snake
{
    internal class Input
    {
        //Load list of available Keyboard buttons
        private static Hashtable keyTable = new Hashtable();

        //Perform a check to see if a particular button is pressed.
        public static bool KeyPressed(Keys key)
        {
            if (keyTable[key] == null)
            {
                return false;
            }

            return (bool)keyTable[key];
        }

        //Detect if a keyboard button is pressed
        public static void ChangeState(Keys key, bool state)
        {
            keyTable[key] = state;
        }
    }
}
```

Settings.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Snake
{
    public enum Direction
    {
        Up1,
        Down1,
        Left1,
        Right1,
        Up2,
        Down2,
        Left2,
        Right2
    };

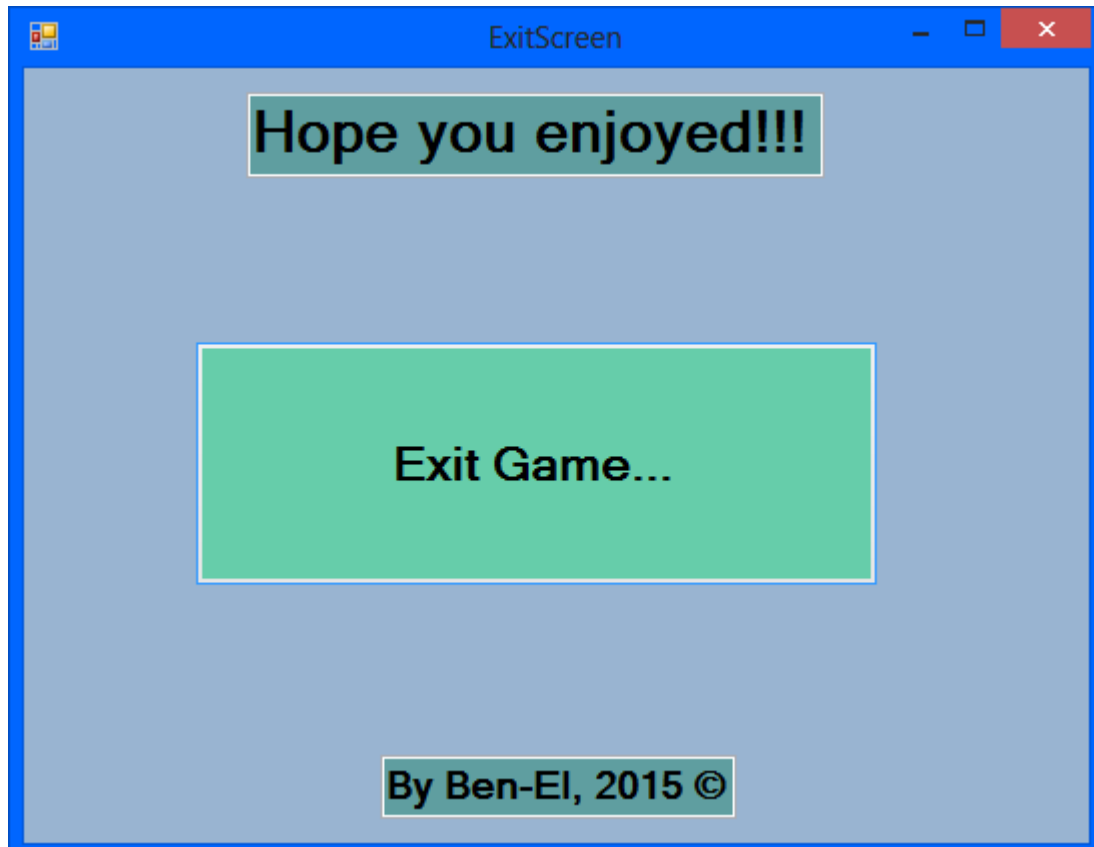
    public class Settings
    {
        public static int Width { get; set; }
        public static int Height { get; set; }
        public static int Speed { get; set; }
        public static int Score1 { get; set; }
        public static int Score2 { get; set; }
        public static int Points { get; set; }
        public static bool GameOver { get; set; }
        public static Direction direction1 { get; set; }
        public static Direction direction2 { get; set; }

        public Settings() // constructor
        {
            Width = 17;
            Height = 17;
            Speed = 17;
            Score1 = 0;
            Score2 = 0;
            Points = 100;
            GameOver = false;
            direction1 = Direction.Right1;
            direction2 = Direction.Right2;
        }
    }
}
```

Program.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

namespace Snake
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new OpeningScreen());
        }
    }
}
```



עבודות גדולות אינן מבוצעות על ידי כוח, אלא על ידי התמדה (סמואל ג'ונסון).