פרויקט גמר בארגון המחשב ושפת סף – הערכה חלופית מדעי המחשב – שאלון 899381

מגיש: עידן ברקין.

שם הפרויקט: Snake .

מורה: שמואל מצא.

שנה: 2022 תשפ"ב.

בית ספר: הנדסאים.

קובץ ראשי: idan_barkin_snake.asm .

הפרויקט

תיאור כללי:

המשחק SNAKE הוא משחק לשחקן יחיד בו השחקן שולט בנחש אשר זז על גבי הלוח. מטרת השחקן היא לצבור כמה שיותר נקודות, על ידי אכילת התפוחים (נקודות אדומות), זאת בזמן שהוא נמנע מהתנגשות עם צידי הלוח או עם גוף הנחש, דבר אשר יוביל להפסד. כאשר ה SNAKE מת ניתנת לשחקן אפשרות לשחק מחדש, והמשחק זוכר את הניקוד המקסימלי אליו השחקן הגיע במשחקים הקודמים.

כאשר הSANKE פוגע בקצה של הלוח, או בעצמו, הוא מת. כאשר הוא מת הוא למעשה מפסיד במשחק הספציפי ששוחק וניתנת לו האפשרות לנסות שוב.

כמו כן השחקן מקבל ניקוד על כל תפוח שהוא אוכל. מטרת המשחק היא להגיע לניקוד הגבוה ביותר שהשחקן ישאף להגיע אליו, ועליו להימנע מהפסד.

תיאור מסכים:

המשתמש מתחיל בתפריט הראשי, משם הוא יכול לקבל את הוראות המשחק, להתחיל משחק חדש, או לסגור את התוכנית.

כאשר השחקן מתחיל משחק חדש הוא עובר למסך אחר בו מודפסים: הלוח (grid) אשר בתוכו: הנחש (בירוק), התפוח (באדום), והאזורים הריקים (בכחול).

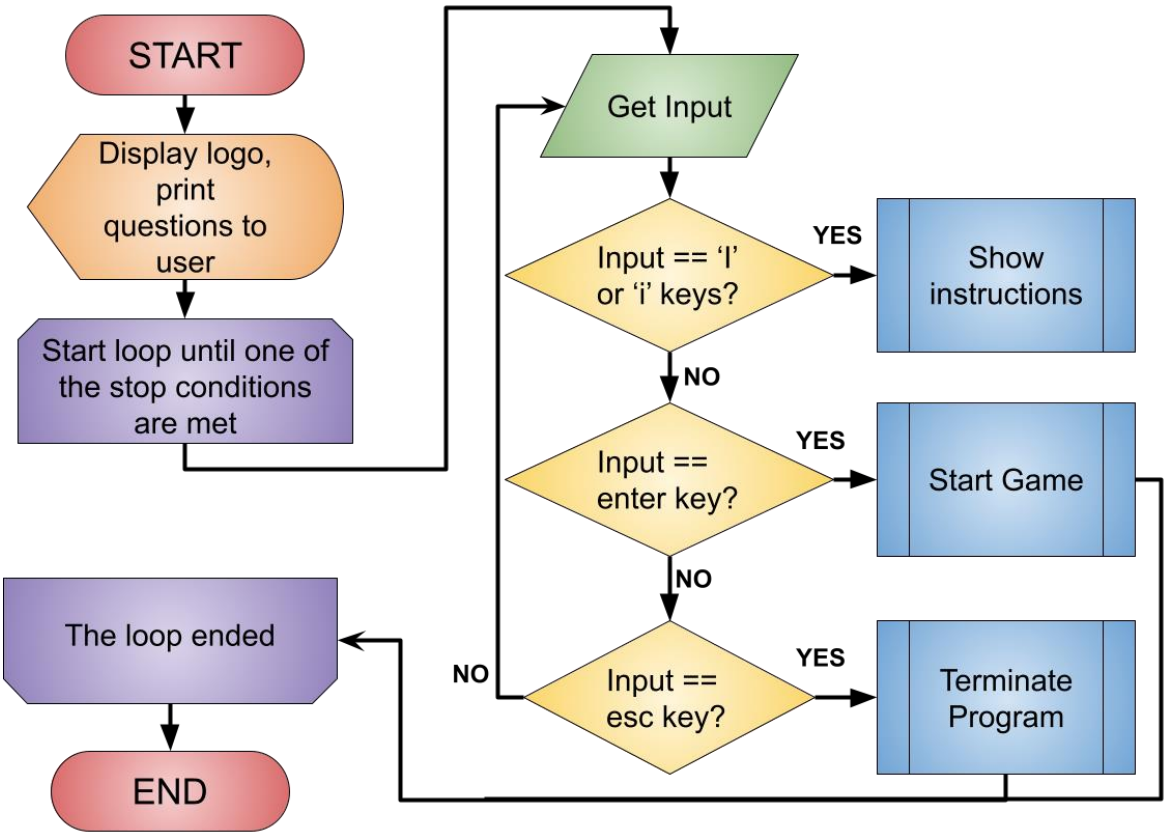
מודפס למסך גם :SCORE ו- :MAX SCORE . ולאחר מכן יודפס מתחתם הניקוד המתאים.

במסך זה מתרחש מהלך המשחק.

על מנת לשחק על השחקן להקיש על מקשי החיצים שעל גבי המקלדת. תזוזת הנחש תיקבע בהתאם למקש הנלחץ (חץ לכיוון ימין: הנחש יזוז ימינה), זאת בתנאי שתזוזה זו אפשרית (לדוגמה אם הנחש זז לכיוון מעלה, ויילחץ למטה על ידי השחקן, כמובן שתזוזה זו אינה אפשרית).

כאשר השחקן מפסיד מופיע לו מסך הפסד (GAME OVER) ולאחר שהוא לוחץ על כפתור כלשהו במקלדת הוא יוחזר למסך הראשי.

<u>תרשים זרימה לתפריט הראשי:</u>



{תיאור חלקי הפרויקט (והפרוצדורות)}

פרויקט ה SNAKE הוא פרויקט ברמת מורכבות כזו שצריך להבין ראשית את כל החלקים מהם הוא מורכב. בחלק זה אעבור על כל מרכיבי הפרויקט, ואסביר אותם לחוד ולאחר מכן גם אסביר כיצד כל החלקים מתחברים יחד על מנת ליצור את הפרויקט הסופי.

המערך הראשי

ראשית לפני שאסביר על כל חלקי הפרויקט רציתי להסביר על הבסיס שעליו מושתת כל הפרויקט, זהו המערך axis_arr .

מערך זה מוגדר כך:

```
axis_arr                    dw ARR_LENGTH dup(0)
```

כל מיקום (Index) במערך זה מייצג מיקום על גבי הלוח שיודפס למשתמש, וכל ערך (Value) הנמצא בו מייצג אחד משלוש (ארבע אפילו) אפשרויות. האפשרויות הן: 0 המייצג ריבוע אפור (ריק) כלומר, ערך חסר משמעות אשר פנוי להיווצרות תפוח / מעבר של הנחש דרכו.

1- מייצג את התפוח, כלומר איפה שמופיע 1- במערך, יופיע תפוח אדום על המסך. לבדיקת האם במיקום במערך יש ערך 1- תהיה חשיבות כאשר נרצה לבדוק האם התרחשה אכילת תפוח.

מספרים מ-1 עד snake_length-1 מייצגים את גוף הנחש. 1 מייצג את זנב הנחש וכך מעלה. במיקומים בהם יש ערכים כאלה יופיע ריבוע ירוק בהיר.

לבסוף המספר snake_length מייצג את ראש הנחש. משום שמיקום זה הוא קריטי ביותר (חשוב לכמעט כל פרוצדורה) שמרתי אותו במשתנה בו אני אשתמש מאוחר יותר. במיקום זה יופיע על המסך ריבוע ירוק כהה שידגיש שמדובר בראש הנחש.

מציאת המיקום של ערך הלוח על ידי המערך

משום שהגדרתי רק מערך אחד, חד-ממדדי, הרי שאי אפשר להשתמש במיקומים שלו כדי לקבוע מיקום על המסך, שכן המסך הוא דו-ממדי. או שאולי, כן אפשר?

על מנת לאפשר שימוש במערך אחד חד-מימדי, פשוט ויעיל יצרתי פרוצדורה בשם CellToAxis . פרוצדורה זו מקבלת מיקום במערך החד-ממדי, וממירה אותו לפוזיציות x ו y על ידי מתמטיקה שיצרתי בעצמי, וכך ניתן יהיה להדפיס את מה שצריך במיקום הנוכחי (פסיקת ההדפסה ללוח דורשת נקודת x ו-y).

המתמטיקה שיצרתי עובדת כך:

$$Y = \frac{Cell}{GetLength}$$

$$X = Cell - (Y \bullet GetLength)$$

בעזרת המתמטיקה הזו אני יכול להמיר את המיקום שבמערך לפוזיציה שניתן להגדיר להדפסה.

במילים אחרות $Cell = (X, Y)$.

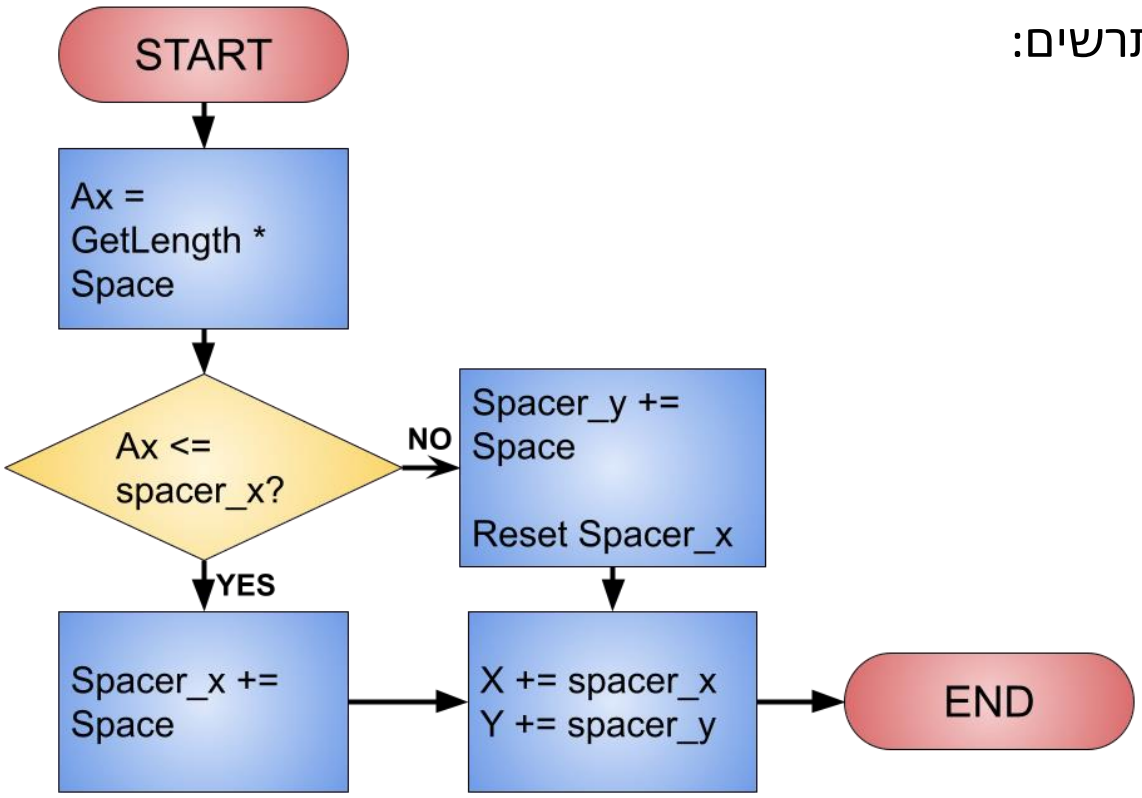
ריווח הריבועים

בתחילה הלוח שלי היה נטול רווחים, אך קיבלתי תלונות מאנשים שניסו את המשחק שמעט קשה לשחק כך. על כן, החלטתי שעליי לרווח כל ריבוע על מנת לשפר את החוויה האסתטית של המשתמש.

פרוצדורה זו גם משתמשת במתמטיקה. למעשה ראשית בדקתי האם המיקום של הריבוע הנוכחי הוא באותה שורה כמו קודמיו או לא. אם כן, אז צריך רק להוסיף לערך המרווח של הx כגודל הרווח שהוגדר. אם לא, יש להוסיף לערך המרווח של הy כגודל הרווח שהוגדר, ולאפס את המרווח של x.

לאחר מכן הוספתי את המרווחים לערכי המיקום x,y .

בתרשים:



יצירת הלוח

כאמור יצירת הלוח מתבססת על המצב במערך הראשי.

על מנת לעבור על ערכי ומיקומי המערך דחפתי (Push) לפונקציה את ההיסט (Offset) של המערך הראשי על מנת לגשת לערכיו בשיטת "Pass By Reference", וגם יצרתי משתנה לוקאלי בשם COUNTER שיעבור על המיקומים (Indexes).

לאחר מכן התחלתי לולאה אשר תעבור על כל ערכי המערך (256 פעמים).

הלולאה ממירה את המיקום במערך, לפוזיציית x ו y על ידי קריאה לפרוצדורה CellToAxis. לאחר מכן היא מכפילה את המיקומים ב 8 (כגודל כל ריבוע), ומוסיפה למיקומים ערך קבוע שימרכז את הלוח. לאחר מכן הפונקציה קוראת לפרוצדורה SpaceSquare שתיצור רווח בגודל קבוע בין כל ריבוע.

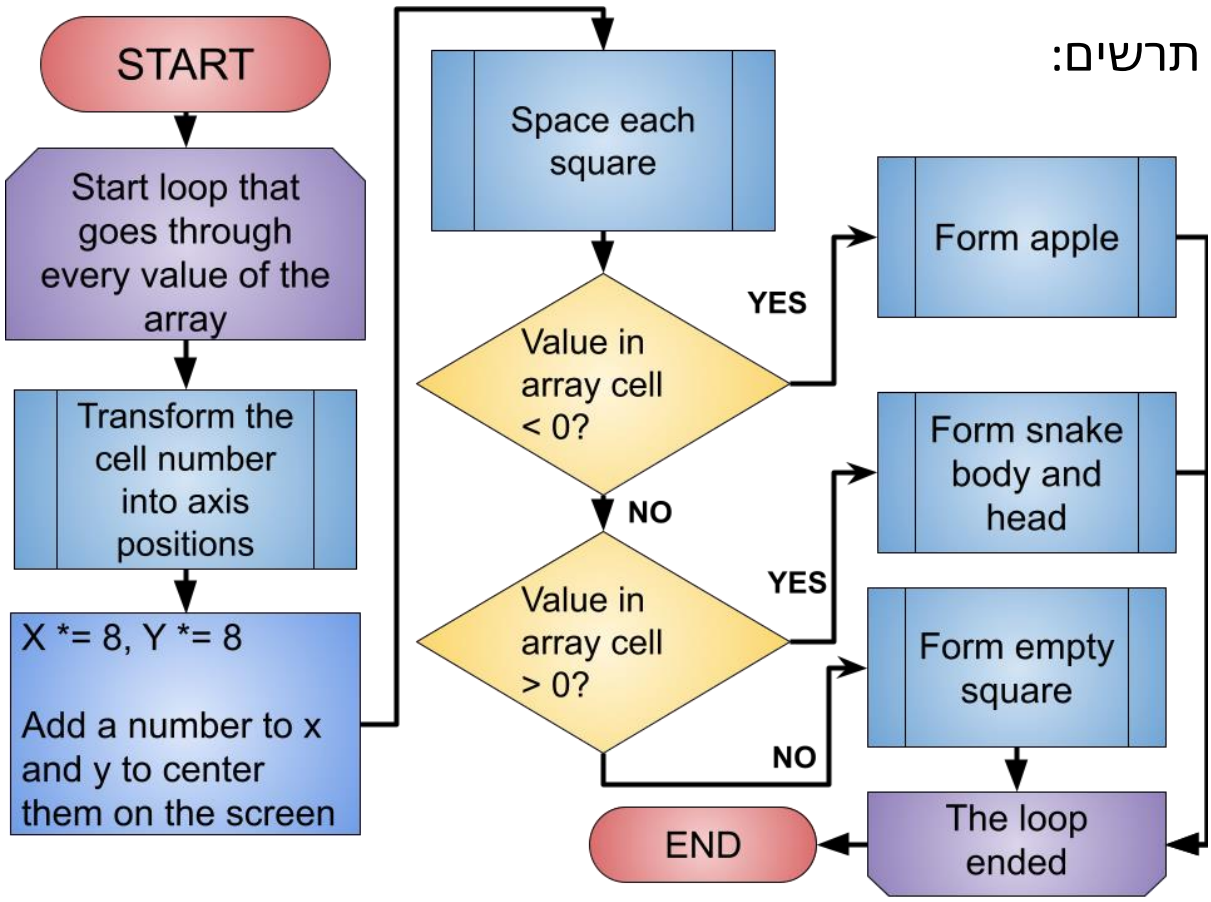
לאחר מכן בדקתי האם הריבוע שעומד להיווצר הוא מסוג תפוח, נחש או ריק. עשיתי זאת על ידי השוואת הערך במיקום הנוכחי ל0, אם הערך קטן מ0 מדובר בתפוח, אם הערך גדול מ0 מדובר בנחש ואם שווה לו – מדובר במשבצת ריקה. אם מדובר בריק או בתפוח הפרוצדורה פשוט תדפיס את הריבוע שלהם בצבע המתאים למסך.

אך אם מדובר בנחש, יש צורך בפעולות נוספות כגון: בדיקה האם תא הנחש הנוכחי הוא הראש, אם כן יש לקבוע את הבוליאני של מציאת הראש לTRUE, לאחר מכן קראתי

לפרוצדורה שתטפל בהדפסת הנחש, אם מדובר בגוף הנחש הדפס את הריבוע בצבע ירוק בהיר ואם מדובר בראש הנחש (אני בודק זאת על ידי הבוליאני שהגדרתי), אז הדפס ריבוע ירוק כהה.

בסוף כל לולאה אני מקדם את המשתנה המקומי ואת הרגיסטר di בו אני משתמש על מנת לבדוק את ערכי המערך בשיטת "Pass By Reference".

תרשים:



(הדפסת הבלוקים)

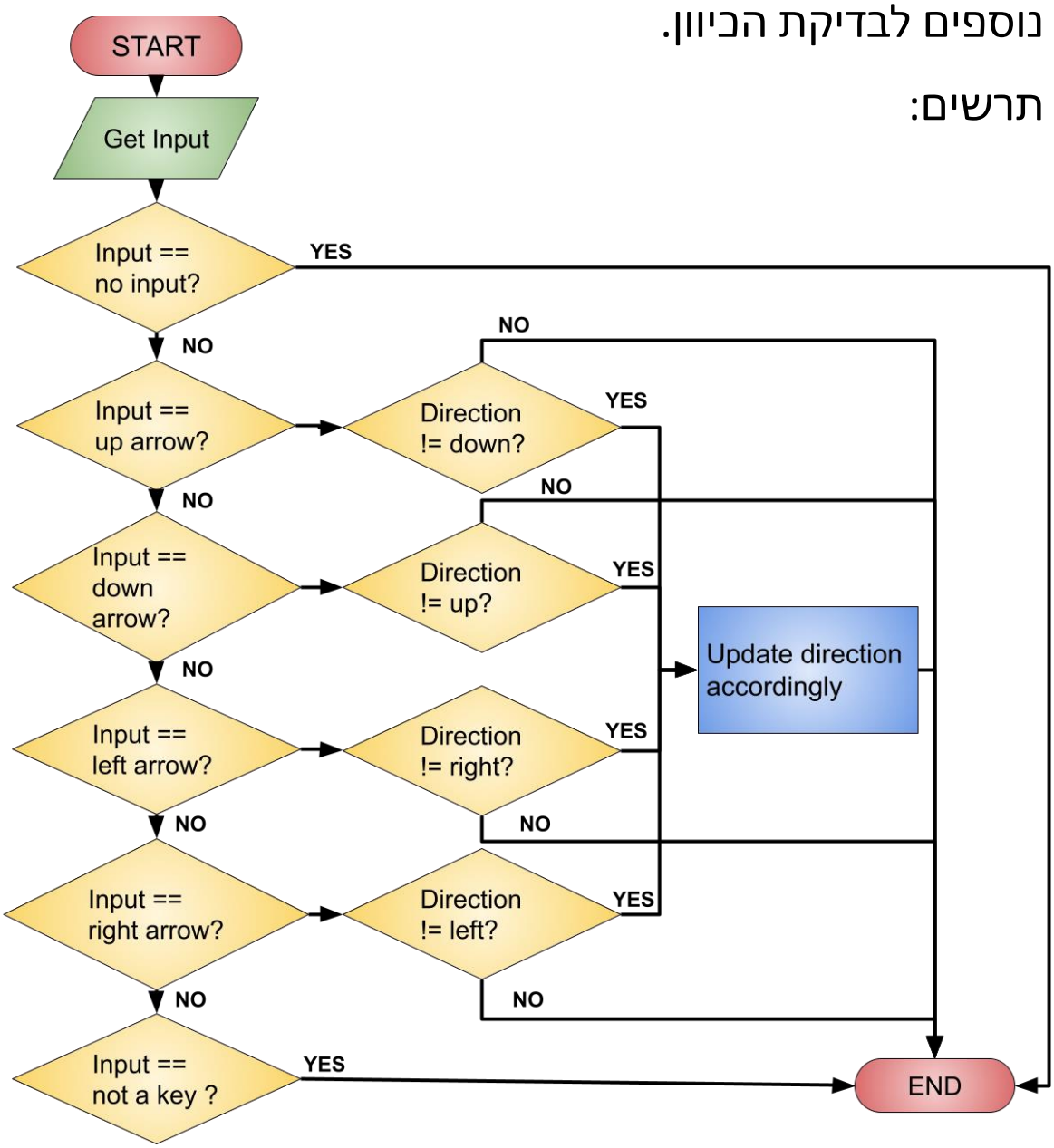
על מנת שאוכל ליצור גרפיקה בקלות יצרתי פרוצדורה גנרית שתוכל להדפיס לי ריבועים בכל גודל שאבחר, בכל מיקום אפשרי על גבי המסך. הפרוצדורה מקבלת גודל של צלע, את הצבע הרצוי, ערך x וערך y ומדפיסה למסך את הריבוע המבוקש

במיקום המבוקש וכך עוזרת ליצור גרפיקה מבלי להסתבך יותר מדי.

בדיקת עדכון בכיוון התנועה

על מנת שהנחש יוכל לזוז ברחבי הלוח בארבעת הכיוונים השונים, על המשתמש להכניס קלט של אחד מארבעת החיצים שעל גבי המקלדת. אלא, שלא בהכרח הכיוון שיוכנס הינו חוקי (לדוגמה, אם הנחש הולך ימינה, מן הסתם שקלט לכיוון שמאל הוא אינו חוקי). על מנת לטפל במקרים כאלו הוספתי תנאים נוספים לבדיקת הכיוון.

תרשים:



הסבר קצר על החלק הבא:

לאחר הפרוצדורות שהוצגו, מגיעה הפרוצדורה הכללית של המשחק CheckAndChangeArrValues, זו שתשנה את ערכי המערך ועל ידי כך תיצור תזוזה, זו שתבדוק האם נאכל תפוח ותגדיל את הנחש והניקוד וזו שגם תקבע האם הנחש הפסיד. למעשה פרוצדורה זו מלבד פעולותיה קוראת לתת-פרוצדורות בתוכה.

אעשה תרשים מפורט לפרוצדורה זו וקריאותיה לאחר שאסביר על כל חלקי הפרוצדורה.

<u>חלקי פרוצדורה זו הם:</u>

תזוזת הנחש על גבי המסך

על מנת להזיז את הנחש על גבי הלוח באופן האופייני למשחק הסנייק יש לשנות מספר דברים.

ראשית כל יש לברר לאיזה כיוון הנחש נע. לאחר בדיקת הכיוון עלי לשנות את פוזיציית ראש הנחש, איך אני עושה זאת? מתמטיקה כמובן. (:

הסבר על המתמטיקה:

Going UP:

$$Head_{cell} - (GetLength * 2)$$

Going DOWN:

$$Head_{cell} + (GetLength * 2)$$

Going LEFT:

$$Head_{cell} - 2$$

Going RIGHT:

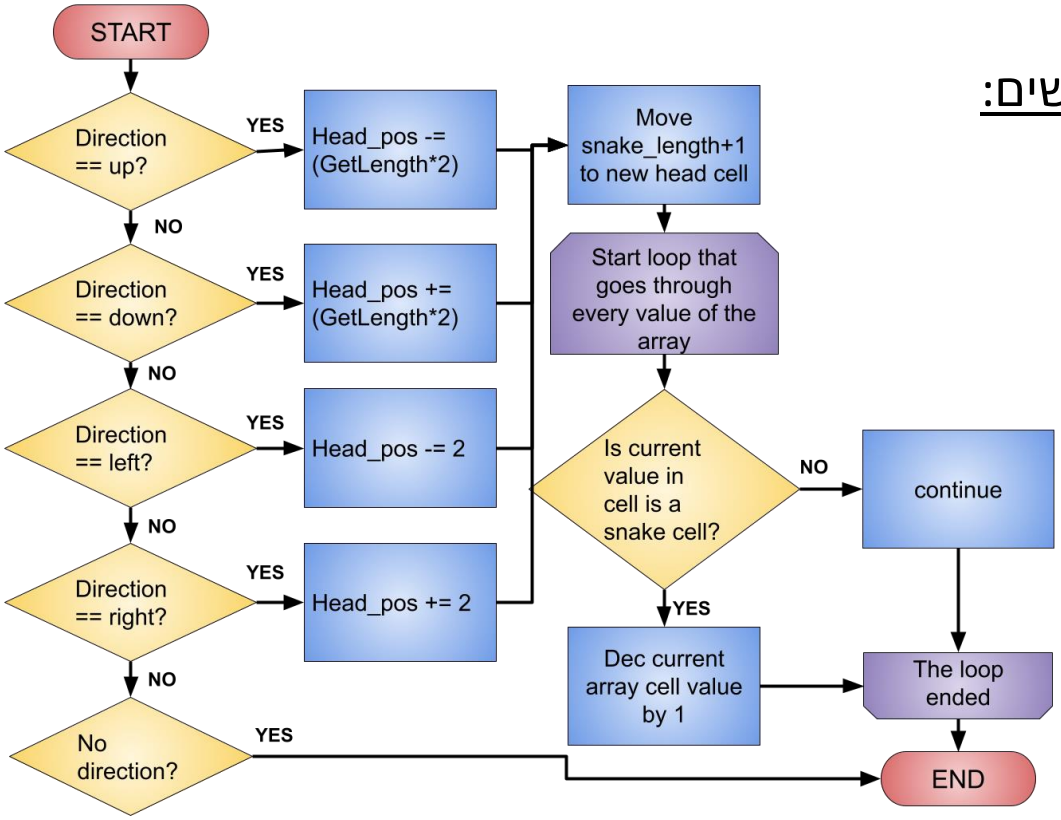
$$Head_{cell} + 2$$

אך כמובן שרק זה בלבד אינו מספיק לתזוזה, יש צורך בפעולות נוספות.

לאחר החישוב של מיקום הראש הבא, אני מעביר למיקום זה במערך ערך בגודל **snake_length+1** הסיבה לפלוס אחד תהיה ברורה יותר בהמשך.

לאחר מכן (בהנחה שהנחש לא מת או אכל תפוח) אני מתחיל בלולאה שתרוץ על כל ערכי המערך אשר תפחית ב1 את כל תאי הנחש שבמערך, כך למעשה תיווצר התזוזה – מדוע? משום שהזנב מיוצג על ידי הערך 1, ועל כן הוא ייהפך ל0 ובהרצה הבאה יימחק (יהפוך לריק), כמו כן המיקום בו שמתי את **snake_length+1** ייהפך לראש הנחש כפי שרציתי, משום שהוא יהיה בגודל של snake_length ומשום שכל שאר ערכי הנחש יוזזו בהתאם, למעשה תיווצר התזוזה.

תרשים:



בדיקת מקרי ההפסד

בSNAKE יש 2 דרכים להפסיד. דרך 1: הנחש מתנגש באחד מן הקירות. דרך 2: הנחש מתנגש בעצמו.

בדקתי את שני מקרים אלה בנפרד. אתחיל בהסבר מקרה ההפסד של הנחש מתנגש בעצמו.

על מנת לבדוק האם הנחש התנגש בעצמו בדקתי האם בפוזיציית הראש החדשה, נמצא ערך הגדול מ1, אם בפוזיציה החדשה נמצא ערך הגדול מ1 הרי שהנחש התנגש באחד מחלקי גוף הנחש, לא משנה איזה, הנחש מת. הסיבה שאני בודק אם הערך גדול מ1, ולא אם הוא גדול מ0, היא משום שאני יודע שהערך 1 (הזנב) יימחק בתזוזה הבאה, כלומר המיקום שלו יהיה פנוי לראש.

על מנת לבדוק אם הנחש התנגש בקירות יש צורך בכמה בדיקות שונות, ראשית כל בדקתי את הכיוון אליו הולך הנחש ולאחר מכן בדקתי האם הוא נמצא בקצה כך שאם הוא יזוז בכיוון אליו הוא ינוע הוא יתנגש בקיר (לדוגמה אם הנחש נע ימינה והוא על הקצה הימני, הוא מת). בדיקת הכיוון היא פשוטה, אך הבדיקה שבודקת האם מיקום הראש הוא באחד מהקצוות דורשת מעט מתמטיקה:

Top Border:

$$Head_{cell} < (GetLength * 2)$$

Bottom Border:

$$Head_{cell} \geq [(ArrLength - GetLength) * 2]$$

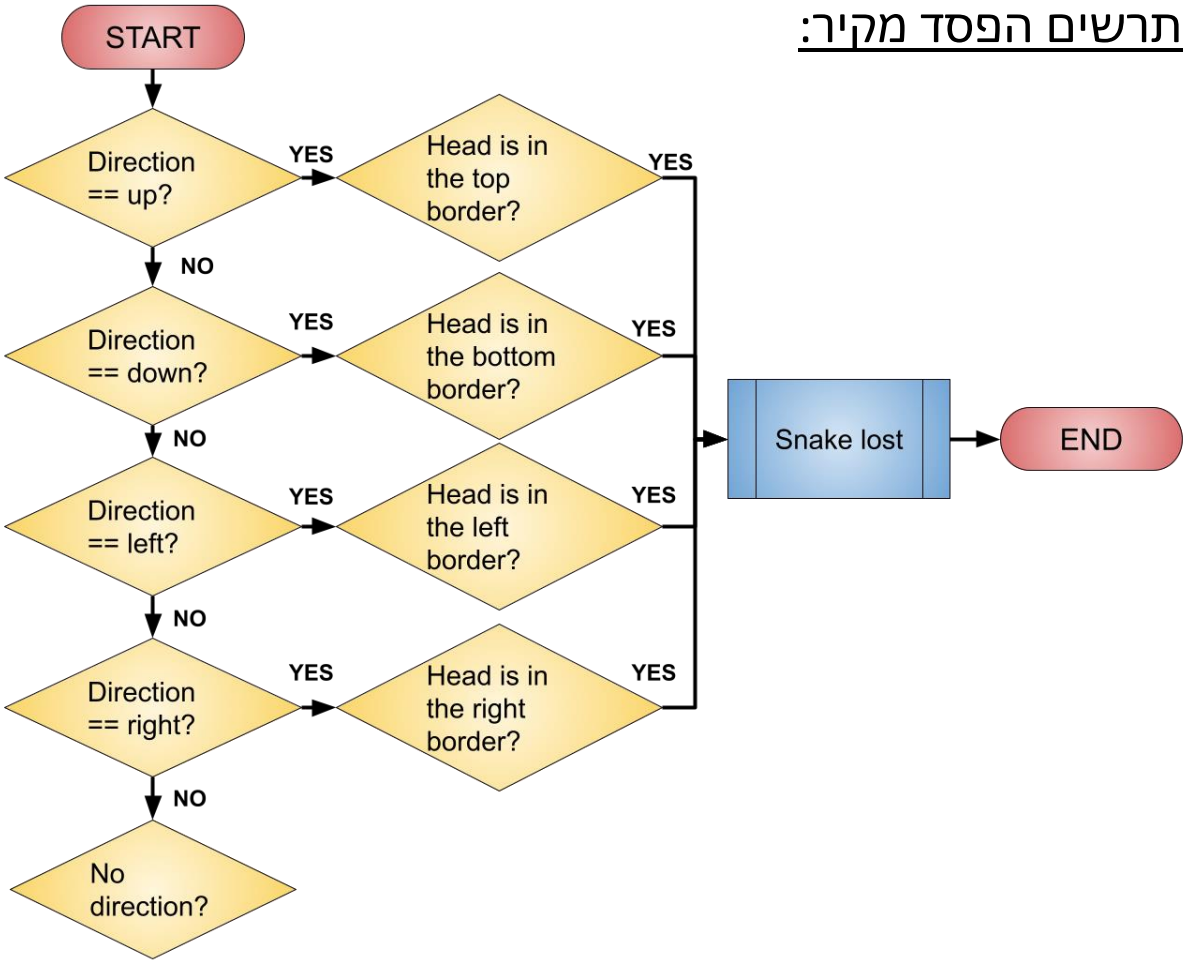
Left Border:

$$Head_{cell} \% (GetLength * 2) == 0$$

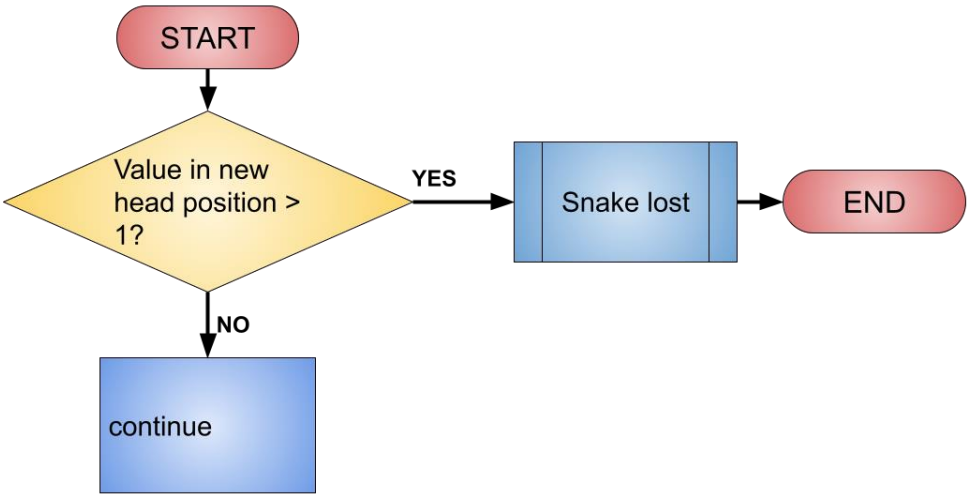
Right Border:

$$(Head_{cell} + 2) \% (GetLength * 2) == 0$$

<u>תרשים הפסד מקיר:</u>



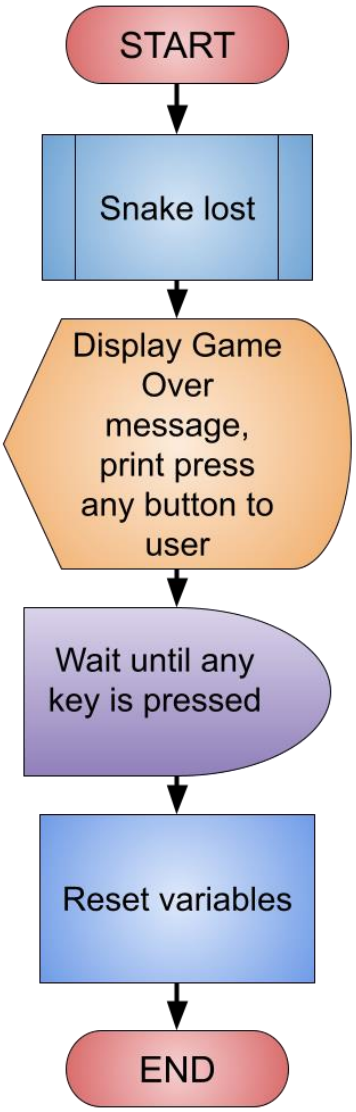
תרשים הפסד מהתנגשות בעצמו:



אתחול המשחק לאחר הפסד

לאחר שהמשתמש הפסיד במשחק הוא רואה מולו מסך GAME OVER בו הוא מתבקש ללחוץ על כל מקש על מנת לחזור למסך הבית. מה שהפרוצדורה עושה בעצם, היא מאתחלת את כל המשתנים לערכיהם המקוריים, ומדליקה משתנה בוליאני שיצביע על כך שצריך לעשות אתחול. כך התוכנה חוזרת ממש לתחילת הקוד, ולאחר אתחול המשתנים וקוראת מחדש למסך הבית ולמשחק כולו. כלומר, המשחק מאותחל מחדש לאחר ההפסד.

תרשים:



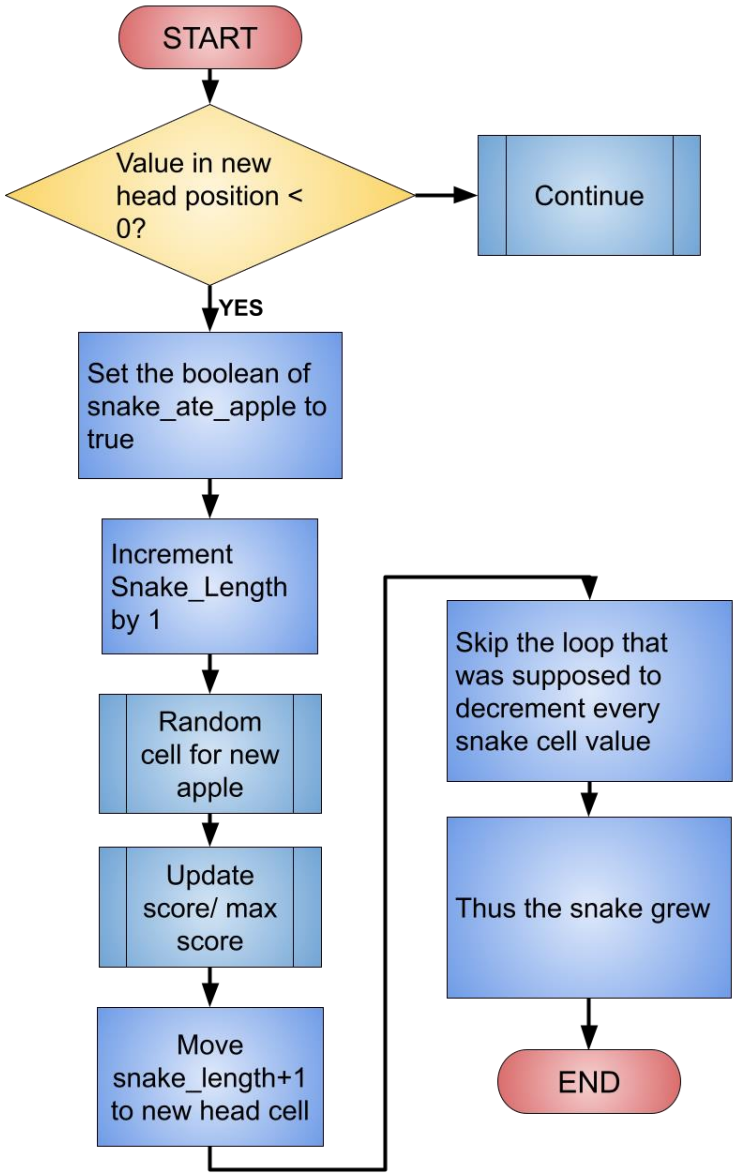
גדילת הנחש על ידי אכילת התפוח

ראשית יש לבדוק האם הנחש אכל את התפוח. על מנת לעשות זאת אני עושה בדיקה דומה לזו של המוות של הנחש על ידי התנגשות בעצמו. למעשה אני משווה את הערך במיקום הראש החדש ל0 ואם הוא קטן מ0 הרי שמדובר בתפוח, ועל כן הראש התנגש בתפוח, ולמעשה "אכל" אותו.

לאחר בדיקה זו יש להגדיל את אורך הנחש. על מנת לעשות זאת כל מה שאני צריך לעשות הוא קודם כל להגדיל את משתנה snake_length באחד, וגם להימנע מכניסה ללולאה שמחסירה באחד את כל ערכי הסנייק, כך למעשה משום שאני מכניס

לראש ערך של `snake_length+1` ונמנע מהחיסור באחד לאחר מכן אני בעצם מגדיל את הנחש וקובע את הערך במיקום הראש החדש לאורך הנחש החדש כפי שצריך. וכך יש לנו גדילה של הנחש!

תרשים:



מערכת Random שתקבע מיקום חדש לתפוח

לאחר שנאכל התפוח עלי היה למצוא דרך ליצור תפוח חדש במיקום רנדומלי על גבי המסך. על כן, היה עלי ליצור מערכת Random.

על מנת לעשות זאת השתמשתי בפסיקה Int 1Ah, 0. פסיקה זו מכניסה ל cx:dx את הtime ticks שעברו מאז חצות הלילה (זמן אמת). הtime ticks מתעדכנים כל שניה בערך 18.206 פעמים.

אם נסתכל על הערך בdx נראה שהוא זה שיתעדכן תמיד משום שהוא מקבל את הlower של הtime ticks ועל כן ניתן להשתמש בערך שמתקבל בו כמספר רנדומלי. אך זה אינו מספיק משום שאני רוצה שהערך הרנדומלי יהיה בתחום הלוח, ועל כן אני נעזר בפעולה מודולו שתגרום לכך שהערך הרנדומלי שיובא בסופו של דבר יהיה בין 0 ל ARR_LENGTH-1 .

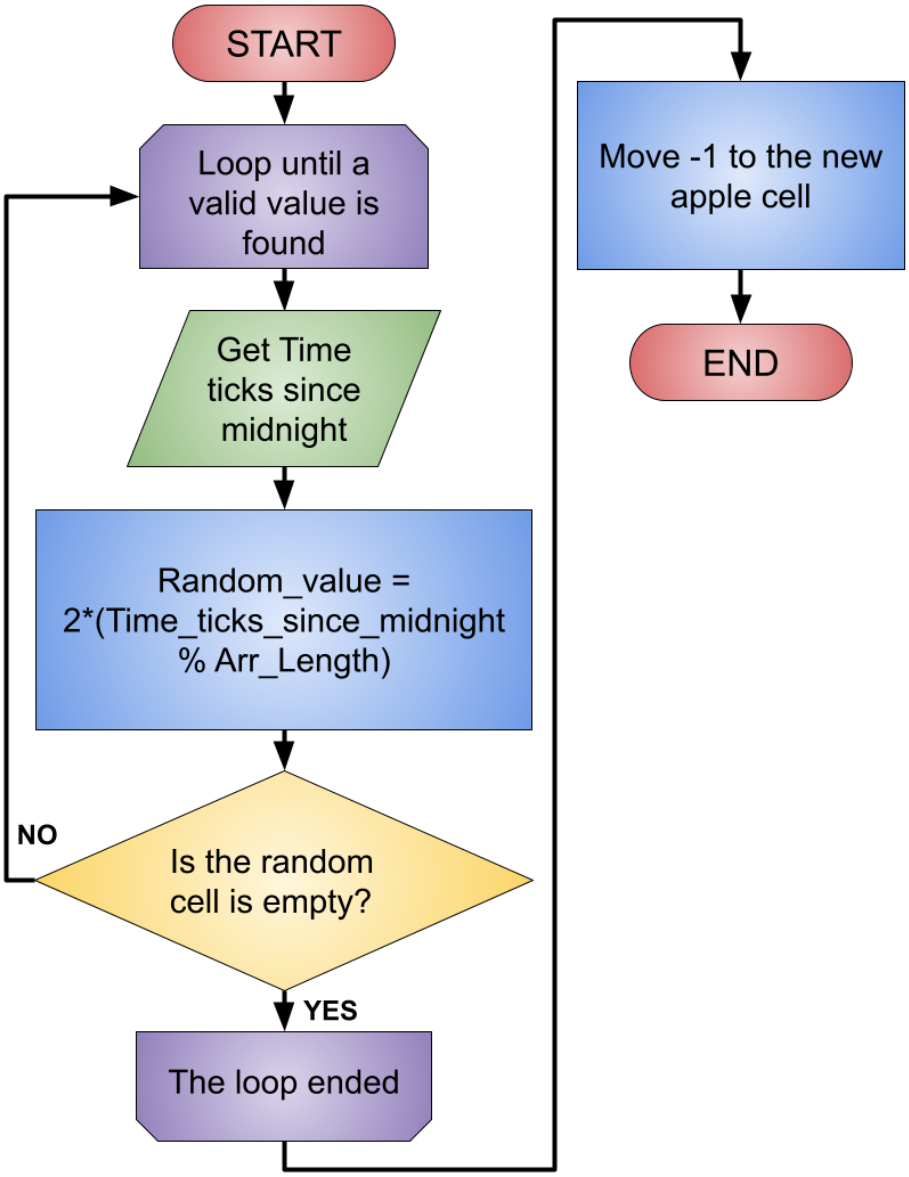
מתמטית זה נראה כך:

Get Random Value From 0 ⟷ (ARR_LENGTH - 1)

$$Time\_Ticks\_In\_Dx \% Arr\_Length = [Random\ Value\ From\ 0 \Leftrightarrow (Arr\_Length - 1)]$$

אך ייתכן והערך הרנדומלי שיצא אינו חוקי, הרי שהתפוח אינו יכול להיווצר במיקום בו נמצא נחש, ועל כן עשיתי בדיקה שתבדוק אם התפוח נוצר במקום שתפוס על ידי הנחש ואם כן יוגרל מחדש מספר רנדומלי.

תרשים:



פרוצדורת המרת כל ערך מספרי חיובי מintegerל string

לפני שאסביר על מערכת הניקוד אני מוכרח להסביר על הליבה של הדפסת הניקוד.

כידוע, על מנת להדפיס למסך מספרים או אותיות יש להדפיס את הערך שלהם בטבלת ה ASCII. לדוגמה אם אני רוצה להדפיס למסך את הספרה 4, עלי להדפיס למסך את ערך הASCII של 4, שהוא 34h או 54d.

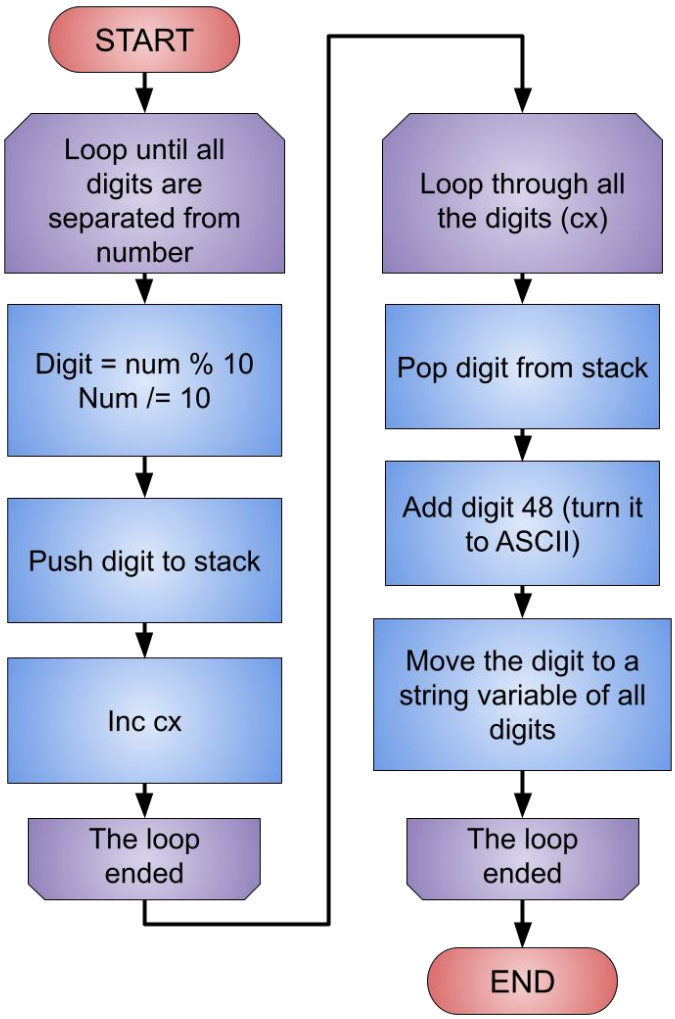
ועל כן יצרתי פרוצדורה שעושה בדיוק כך, אבל לא מטפלת רק במספרים חד ספרתיים אלה בכל רמה ספרתית. איך עשיתי זאת?

יצרתי לולאה שתרוץ עד אשר יתאפס הערך המספרי. כל הרצה אני עושה פעולת מודולו 10 למספר אותו אני רוצה להדפיס וגם מחלק אותו ב10 כל פעם, ולמעשה כך אני מפריד את הספרות שלו אחת אחת עד אשר הפרדתי את כולן. כאשר אני מפריד ספרה אני דוחף (Push) אותה למחסנית (Stack), הסיבה לכך תהיה ברורה יותר בהמשך. כמו כן אני גם מגדיל את הערך של cx באחד כל פעם, גם הסיבה לכך תהיה ברורה יותר בהמשך.

לאחר שדחפתי את כל הספרות למחסנית, אני מעוניין להוציא אותן משם. שאלה שוודאי תישאל, מדוע דחפתי אותם למחסנית מלכתחילה? הסיבה לכך היא שאני בעצם מנצל את הדרך שבא המחסנית עובדת לטובתי. כידוע המחסנית עובדת כך שהערך הראשון שנכנס – יצא ממנה אחרון, הערך האחרון שנכנס – יצא ממנה ראשון. משום שהספרות יצאו מהמספר בסדר עולה (קודם ספרת האחדות, אז עשרות, אז מאות וכו') אם אדפיס אותן מיידית הרי שלא ישוקף המספר האמיתי (לדוגמה במקום 198 יודפס 891). על כן הכנסתי אותם לStack וברגע שאני אוציא אותן משם הן יצאו בדיוק בסדר שאני רוצה להדפיס אותן, בדיוק בגלל הדרך שבא המחסנית עובדת כפי שתיארתי קודם לכן.

וכך, בלולאה שתרוץ cx פעמים (הסיבה שהגדלנו אותו באחד כל פעם קודם לכן), אני מוציא ספרה ספרה מהמחסנית, מוסיף לכל ספרה 48, שהוספה של מספר זה ישנה אותם לערך הASCII שלהם ומכניס אותם לString שיודפס לאחר מכן. וכך בעזרת פרוצדורה זו אפשר להמיר כל מספר למחרוזת.

תרשים:



מערכת הניקוד והניקוד המקסימלי

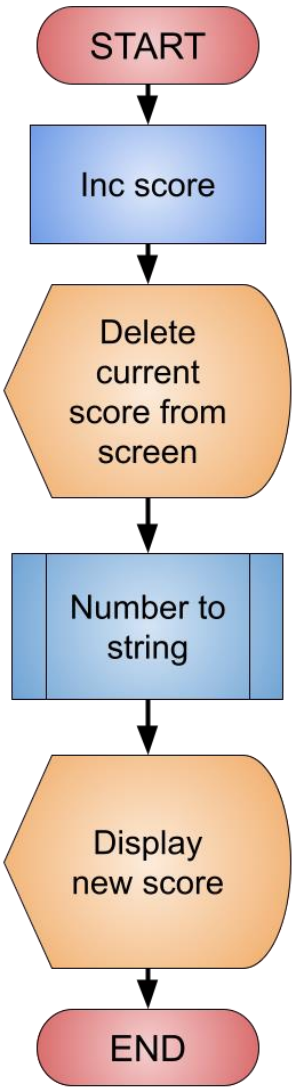
על מנת שיהיה לשחקן מעניין וגם שתהיה לו דרך לעקוב אחר כמות התפוחים שאכל יצרתי שתי פרוצדורות אשר חולקות שתיהן פרוצדורה גנרית. אסביר מיד למה הכוונה.

ראשית יש פרוצדורה שתפקידה להגדיל את הניקוד המקסימלי ב1 לאחר אכילת תפוח. פרוצדורה זו לאחר מכן מוחקת את הערך שהופיע עד אז בתור הניקוד, וקוראת לפרוצדורה שממירה מספר למחרוזת, ולאחר מכן מדפיסה את הניקוד.

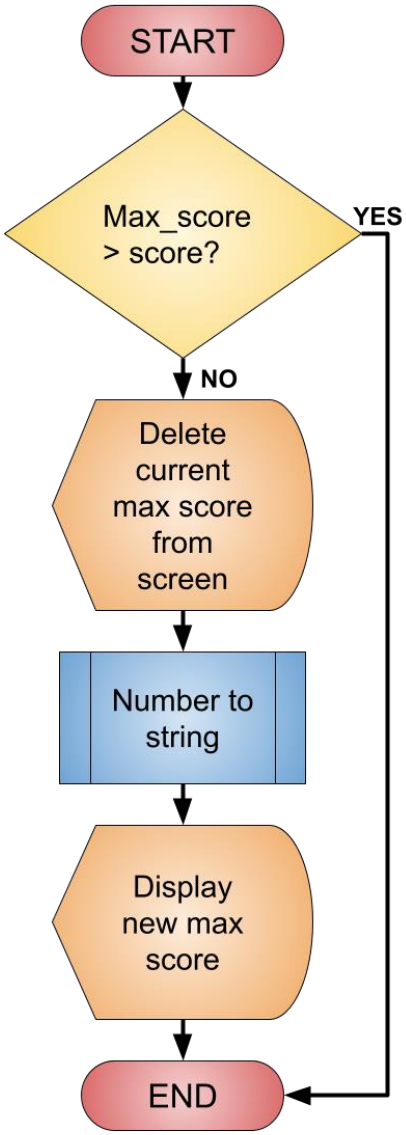
בנוגע לניקוד המקסימלי אני פשוט בודק כל פעם שמתעדכן הניקוד, האם הניקוד החדש גדול יותר מהניקוד המקסימלי ואם כן אז אני מגדיר את הניקוד המקסימלי כך שיהיה שווה לניקוד

הנוכחי, לאחר מכן אני מוחק את הערך שהופיע על המסך עד אז ומדפיס את הניקוד המקסימלי החדש (שהומר למחרוזת על ידי הפעולה שממירה מספר למחרוזת).

ניקוד:



ניקוד מקסימלי:



מערכת delay

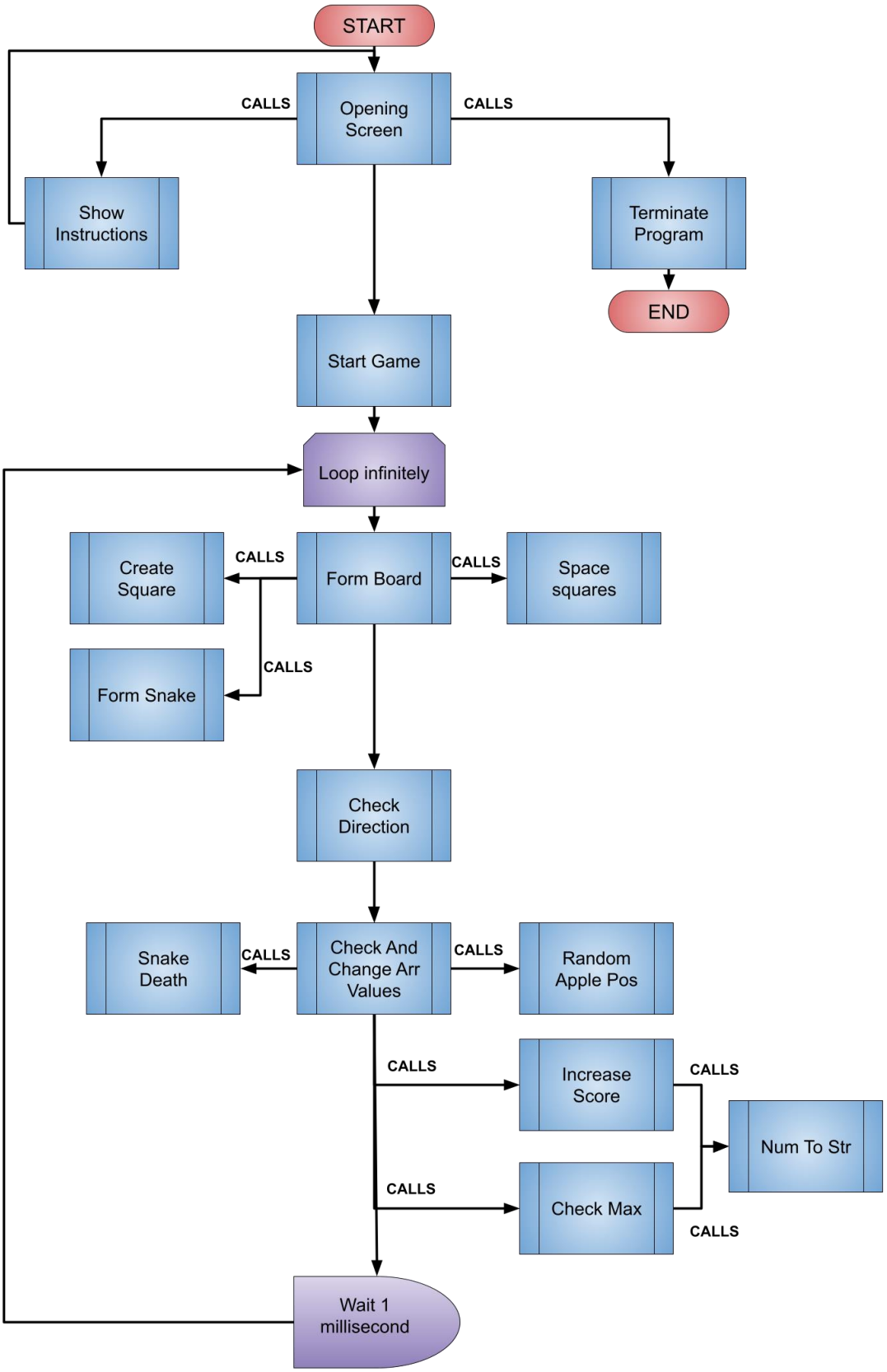
משום שהמשחק רץ מהר מאוד, המהירות הגנרית למעשה בלתי אפשרי לשחק בה משחק הגיוני (הנחש זז כל כך מהר שהוא יתנגש ישר באיזה קיר לפני שתינתן בכלל הזדמנות לשחקן להגיב).

לכן השתמשתי בפסיקה Int 15h, 86h אשר מקבלת כקלט ב cx:dx את מספר המיקרו-שניות שעליה לחכות. אני הגדרתי את cx להיות 01h ואת dx להיות 086A0, כלומר 100,000 מיקרו-שניות או במילים אחרות – מילישנייה אחד.

כך מהירות המשחק היא אפשרית למשתמש לשחק בה.

תרשים וטבלת פרוצדורות

תרשים זרימה – סדר קריאות לפרוצדורות:



טבלת פרוצדורת בקובץ הראשי

שם	קלט	פלט	מה מבצעת
MainScreen	באפר מקלדת	לוגו של המשחק, שאלות לשחקן	נותנת לשחקן אפשרות לקבל הוראות משחק, להתחיל לשחק או לסגור את התוכנית
DrawBigText	אורך ורוחב הקטע שיודפס כמו גם המיקום שלו על המסך וההיסט (offset) של הקטע	כל צורה שתיכנס אליו תודפס בצבעים ללוח	מדפיסה צורות (הנמצאות במערך דו-ממדי) בצבעים הכתובים במערך ללוח
FormBoard	ההיסט של המערך הראשי (מערך הלוח)	יודפס לוח של המשחק למסך	מדפיסה לוח חדש למסך ועל ידי כך מאתחלת את הלוח עם השינויים שנוספו במהלך הקוד

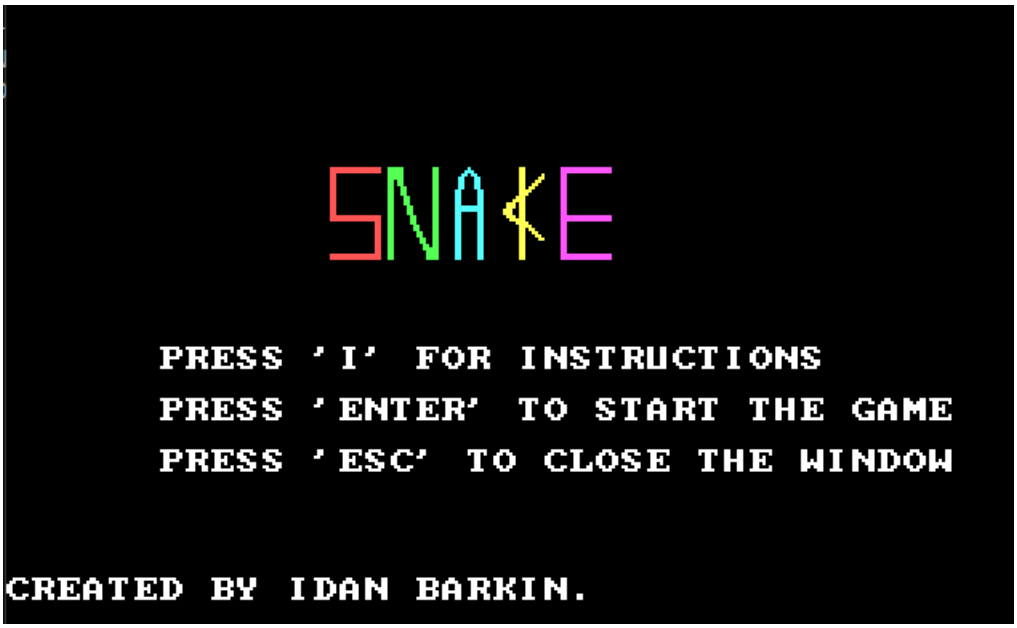
CellToAxis	האינדקס של תא במערך	אין	ממירה אינדקס של תא (חד-ממדי) לערכי (x,y) [דו-ממדי] שיהיה אפשר להשתמש בהם להדפסה
FormSnake	האינדקס של תא נחש, פוזיציית ה-x וה-y של ריבוע נחש	ריבוע גוף נחש או ראש נחש מודפס למסך	בודקת האם תא הנחש הוא מסוג גוף או ראש ומדפיסה ריבוע בהתאם
SpaceSquare	אין	אין	מוסיפה לכל ריבוע ריווח כך שבין כל ריבוע יהיה רווח קבוע כפי שמוגדר
CheckDirection	באפר מקלדת	אין	מעדכנת את משתנה הכיוון לכיוון החדש שהוכנס (בתנאי שהוא חוקי)

CheckAndChange ArrValues	הכיוון הנוכחי, פוזיציית תא הראש, ההיסט של מערך הלוח	אין	הפרוצדורה מעדכנת את המיקום של הנחש ועל ידי כך יוצרת תזוזה, אך היא גם בודקות האם הנחש מת והאם הנחש אכל תפוח ואם כן אז מגדילה את הנחש
SnakeDeath	באפר מקלדת	מסך Game_Over	מדפיסה מסך מוות, מחכה עד שהמשתמש יגיב ואז מאתחלת את כל המשתנים בהכנה לאתחול של המשחק ואפשרות לשחק מחדש
RandomApplePos	אין	אין	מגרילה ערך רנדומלי חוקי למיקום הבא של התפוח

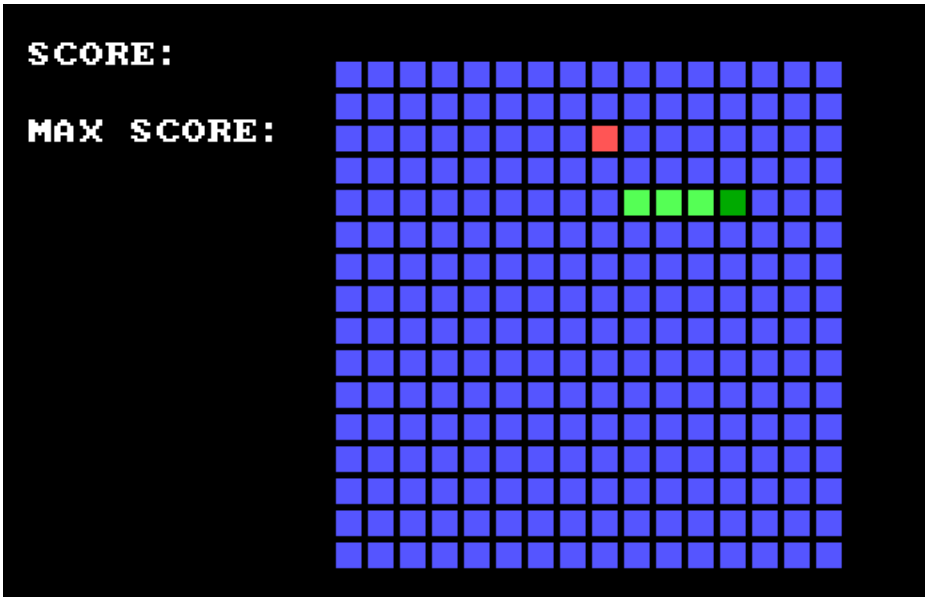
IncreaseScore	אין	מדפיסה את הניקוד הנוכחי	מגדילה את הניקוד הנוכחי ב1 ומדפיסה אותו לאחר מכן
CheckMax	הניקוד הנוכחי	מדפיסה את הניקוד המקסימלי	בודקת האם המשתמש הגיע למקסימום חדש ואם כן מעדכנת את המשתנה ומדפיסה אותו
NumToStr	ההיסט של סטרינג כלשהו, הערך המספרי אותו רוצים להמיר	אין	לוקחת ערך מספרי בכל גודל וממירה אותו למחרוזת הניתנת להדפסה
CreateSquare	גודל הצלע, הצבע הנבחר, פוזיציות x,y	ריבוע בגודל, בצבע ובמיקום המבוקש	מדפיסה כל ריבוע אפשרי (כל הגדלים, הצבעים והמיקומים האפשריים)

{ צילומי מסך }

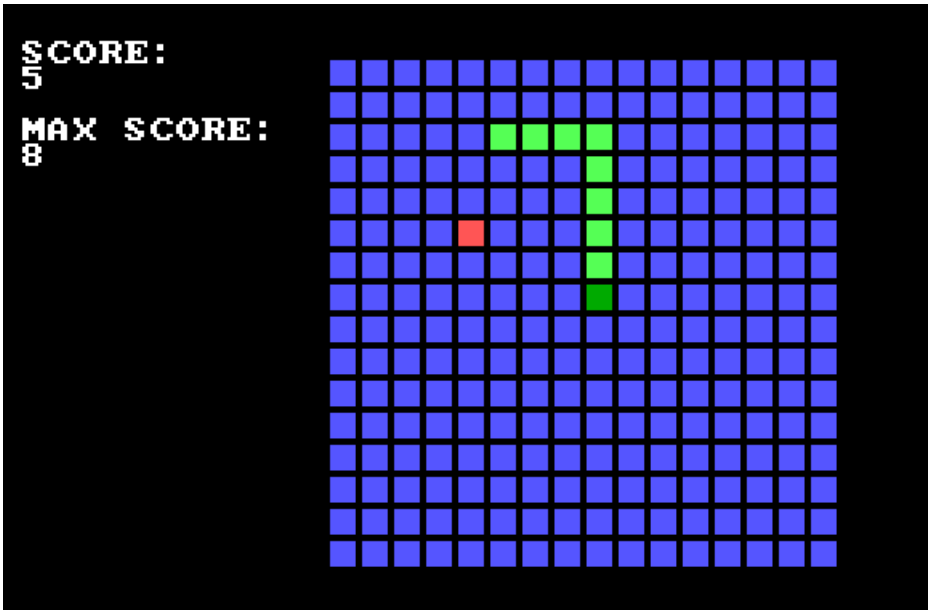
מסך פתיחה:



התחלת המשחק:



תיעוד מהמשחק:



מסך המוות:

{ קטע קוד מובחר }

לקח לי לא מעט זמן לחשוב איזה פרוצדורה לשים פה, התלבטתי בין הקוד שממיר כל ערך מספרי למחרוזת הראויה להדפסה לבין הקוד היעיל שלי לבדיקה ושינוי ערכי המערך. בסופו של דבר החלטתי לבחור בקוד לבדיקה ושינוי המערך:

```
    ; change snake position on axis board, according to the direction
    ; this procedure accesses the array using "pass by refernce" method
    ; this procedure also gets the direction and head cell
    ; this procedure changes the snake position in the array, which will
affect the screen in the next iteration, thus create a movment
    push offset axis_arr
    push head_cell
    push direction
    call CheckAndChangeArrValues

……
……
……


    REDIRECT                    EQU bp + 4
    HEAD_CELL_POS               EQU bp + 6
    BOARD_ARR_OFFSET            EQU bp + 8
    COUNT                       EQU bp - 2
    proc CheckAndChangeArrValues
        push bp
        mov bp, sp
        sub sp, 2
        push ax
        push bx
        push di
        push cx
```

```
        ; reset the boolean
        mov snake_ate_apple, FALSE

        ; set the new head position according to the direction

        ; check the direction in which the snake is going
        cmp [REDIRECT], GOING_UP
        je facing_up

        cmp [REDIRECT], GOING_DOWN
        je facing_down

        cmp [REDIRECT], GOING_LEFT
        je facing_left

        cmp [REDIRECT], GOING_RIGHT
        je facing_right

        jmp finished_calculating_head_pos

    facing_up:
        ; death check: top border
        mov ax, GET_LENGTH
        shl ax, 1
        mov cx, [HEAD_CELL_POS]
        cmp cx, ax
        jb death

        ;;;;;;;;; snake is alive, change head position

        ; if it is going up, than the new head position is: current_pos -
(GET_LENGTH * 2) (times 2 is because it is a word size)
        mov ax, GET_LENGTH
        shl ax, 1
        sub [HEAD_CELL_POS], ax
        jmp finished_calculating_head_pos

    facing_down:
        ; death check: bottom border
        mov ax, ARR_LENGTH
        sub ax, GET_LENGTH
        shl ax, 1
        mov cx, [HEAD_CELL_POS]
```

```
        cmp cx, ax
        jae death

        ;;;;;;;;; snake is alive, change head position

        ; if it is going down, than the new head position is: current_pos +
(GET_LENGTH * 2) (times 2 is because it is a word size)
        mov ax, GET_LENGTH
        shl ax, 1
        add [HEAD_CELL_POS], ax
        jmp finished_calculating_head_pos

    facing_left:
        ; death check: left border
        mov ax, [HEAD_CELL_POS]
        mov cx, GET_LENGTH
        shl cx, 1
        xor dx, dx
        div cx
        cmp dx, 0 ; if modulu 16 is zero (no division reminder)
        je death

        ;;;;;;;;; snake is alive, change head position

        ; if it is going left, than the new head position is: current_pos - 2
(minus 2 is because it is a word size)
        sub [HEAD_CELL_POS], 2
        jmp finished_calculating_head_pos

    facing_right:
        ; death check: right border
        mov ax, [HEAD_CELL_POS]
        add ax, 2 ; if head_pos + 1 is divisble by 16 its in the right border
        mov cx, GET_LENGTH
        shl cx, 1
        xor dx, dx
        div cx
        cmp dx, 0 ; if modulu 16 is zero (no division reminder)
        je death

        ;;;;;;;;; snake is alive, change head position
```

```
        ; if it is going left, than the new head position is: current_pos + 2
(plus because it is a word size)
        add [HEAD_CELL_POS], 2
        jmp finished_calculating_head_pos

    finished_calculating_head_pos:
        ; update the new snake head position
        mov ax, snake_length
        ; this will be decremented very soon
        inc ax

        mov bx, [HEAD_CELL_POS]

        ; check if snake ate apple
        cmp axis_arr[bx], 0
        jl snake_eaten_apple

        ; check snake death by eating himself
        cmp axis_arr[bx], 1
        jg snake_ate_himself

        jmp snake_didnt_eat

    snake_ate_himself:
        ; call death message prepare for a reset of the game
        call SnakeDeath
        mov game_needs_to_reset, TRUE
        jmp not_ready_update

    snake_eaten_apple:
        ; prepare for snake growth
        mov snake_ate_apple, TRUE
        inc snake_length
        ; set a new apple position
        call RandomApplePos

        ; increase the score
        call IncreaseScore
        ; check if needs to increase the max score
        push current_score
        call CheckMax
```

```
    snake_didnt_eat:
        ; mov the head value (snake_length+1) to the new head position
        mov axis_arr[bx], ax

        ...........
        ;;;;;;;;;;

        ; decrement every snake value. this will delete the tail and cause a
movement.
        ; that is also the reason for the increment of the snake head value
beforehand, as it will get decremented now

        ; make a growing by not decrementing the snake values
        cmp snake_ate_apple, TRUE
        je not_ready_update

        mov di, [BOARD_ARR_OFFSET] ; di will access the array values
using "pass by refernce" method
        mov cx, ARR_LENGTH ; cx will help loop until the last value of the
array
        mov ax, 00000000h
        mov [COUNT], ax ; count is a local variable that will keep track of
the indexes

    change_snake_pos_loop:

        ; check if found snake body
        cmp [di], 0
        jg its_snake_body

        jmp not_snake


    its_snake_body:
        ; if found snake body, decrement its value
        dec [di]

    not_snake:
        ; keep looking
        add di, 2
        inc [COUNTER]

        loop change_snake_pos_loop
```

```
    ; reset the booleans
    mov formed_snake, FALSE
    mov count_snake, 0
    jmp not_ready_update

death:
    call SnakeDeath
    mov game_needs_to_reset, TRUE

not_ready_update:
    pop cx
    pop di
    pop bx
    pop ax
    add sp, 2
    pop bp
    ret 6
endp CheckAndChangeArrValues
```

קשיים ובעיות

זה פרויקט התכנות הגדול ביותר שאי פעם עשיתי, וכראוי לכך היו מספר בעיות שהייתי צריך לתקן כגון:

מרבית הבעיות שהיו לי קרו כי שכחתי להכפיל ערכים ב2, ואז בגלל שהמערך הראשי שלי בגודל word זה יצר כל מיני בעיות, לרוב גיליתי באגים מסוג זה לאחר דיבאג קצרצר או שזיהיתי ישר.

בעיות שלקחו קצת יותר זמן:

בעיה שכאשר הנחש אוכל תפוח לאחר Random הוא מת:

הבעיה הייתה, שייתכן וערך התפוח יצא במיקום הlower של תא במערך, דבר שיוביל לבאגים כפי שאכן קרה.

על מנת לגלות את בעיה זו פתחתי את הדיבאגר ועברתי על מה שהתרחש שם, ולאחר שעקבתי אחרי השינויים בזיכרון בסגמנט ds , גיליתי שהבעיה הייתה חוסר ההכפלה ב2.

בעיה שהנחש לא מת בקצוות הימנית והשמאלית כאשר הוא מתנגש בהם ולפעמים סתם מת:

הייתי משוכנע שהשיטה שלי לגלות את האם הראש בקצוות עובדת, אך לאחר שעשיתי תהליך דיבאג קצר התברר לי שבכלל הבדיקה לא בודקת את הקצוות הימניים והשמאליים אלא כמעט כל ריבוע ולכן גם הנחש מת סתם ככה. לאחר שגיליתי שהבעיה בשיטה הגיתי שיטה חדשה שתבדוק האם התא מתחלק ב16 בלי שארית ואז הוא בצד שמאל או שהתא פלוס אחד מתחלק ב16 בלי שארית ואז הוא בצד ימין. וכך תיקנתי את הבאג.

בעיות בריווח:

אם כמה שהקוד של הרווח הוא יעיל בסופו של דבר להגיע לכך דרש הרבה מחשבה. בתחילה פשוט הוספתי את ערך הריווח לכל ריבוע, אך זה פשוט הזיז את כל הריבועים מעט הצידה ולא יצר רווח.

לאחר מכן ניסיתי שיטה בה אני מכפיל כל פעם את הערך של צלע לוח במספר הspacer_x הנוכחי. זה אומנם יצר רווחים, אך מוזרים ביותר ולא עקביים בכלל, מה גם שלא קידם את spacer_y מה שגרם לי להבין שזה מדלג על החלק של זה וכנראה שם הבעיה. לאחר דיבוג רציני הבנתי שכן צריך להכפיל במשהו אבל לא בspacer_x אלא בערך קבוע, שהוא ה SPACE, ואכן הריווח עבד לאחר מכן.

אלא, שגיליתי שאם אני מגדיל את הריווח נוצרות בעיות. לאחר דיבאג גיליתי שפשוט העברתי באיפוס למשתנים spacer_x spacer_y את הערך עצמו ולא את SPACE. לאחר שתיקנתי זאת הריווח עבד מושלם.

מילות סיכום

כפי שאמרתי קודם לכן, זהו פרויקט התכנות הגדול ביותר שעשיתי בחיים שלי. למעשה הזמן שלקח לי להתחיל את הפרויקט לקח יותר זמן מלעבוד עליו משום שבהתחלה נתקעתי עם כל מיני מחשבות של כיצד אעשה כך וכיצד אעשה משהו אחר וזה מאוד הרתיע אותו מלהתחיל לעבוד. אך, ברגע שהתחלתי באופן רציני לעבוד ולחשוב על לוגיקות הפרויקט התקדם יפה.

הפרויקט תרם לי בהתמודדות עם לחץ ועם דד-ליין, הבנתי שאין לי זמן לבזבז על להילחץ ושעלי פשוט לתכנת את הקוד על מנת שאצליח לספק את הפרויקט הטוב ביותר שאוכל. בסופו של דבר זו הייתה חוויה מאוד מהנה ומעשירה.

כמו כן, הידע שלי לאחר הפרויקט בתכנות ובאסמבלי בפרט כמו גם החשיבה הלוגית שלי השתפרו מאוד במהלך הפרויקט ואני רואה זאת כנקודת ציון בין אותו תלמיד שרק התחיל כיתה י' לתלמיד בסוף כיתה י' שהצליח לכתוב פרויקט מכובד מאוד.

{ מקורות חיצוניים: }

Use the modulo (%) operator to generate random numbers within a specific range. | Sololearn: Learn to code for FREE!

Flow Chart Symbols

Assembly: REP MOVS mechanism - Stack Overflow

Intel Instruction Set - REP

int 1a,0

int 15,86

int 10,2

int 19

cold boot

מבנה המחשב ושפת אסמבלי

התמונה הזו שמצאתי בגלריה של גוגל עזרה לי להבין כיצד להגדיר ערכים במערך: