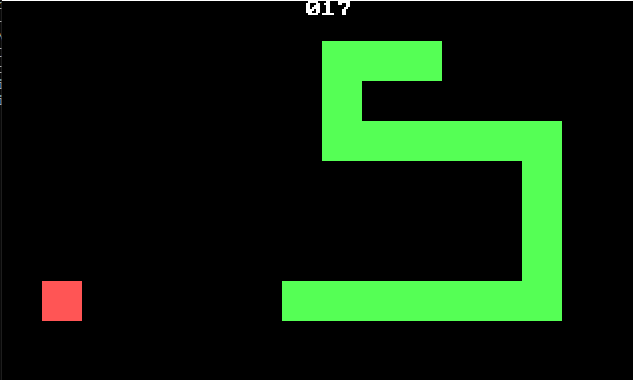
**פרוייקט באסמבלי - Snake**

****

מגיש: הראל שץ

תאריך: 30.9.2021

**תוכן עיניינים:**

**הקדמה**.................................................................................................... 2

**הסבר על המשחק**..................................................................................... 3

**השליטה במשחק**...................................................................................... 4

בעיה – Keyboard Buffer.................................................................. 4

אופטימיזציה – Switcher................................................................... 6

תרשים זרימה של תהליך קבלת הקלט................................................. 8

**גרפיקה**.................................................................................................... 9

מערכת ה-Tiles................................................................................ 9

תרשים זרימה של הפונקציה DRAW\_TILE........................................... 11

**תזוזת הנחש**........................................................................................... 12

בעיה – מיקום הזנב......................................................................... 12

אורך הנחש.................................................................................... 13

פונקציית התזוזה............................................................................ 14

בדיקת ההתנגשות........................................................................... 14

הפסד........................................................................................... 15

אכילת פרי..................................................................................... 15

תרשים זרימה של פונקציית התזוזה................................................... 16

**ניקוד המשחק**.......................................................................................... 17

אכסון הניקוד.................................................................................. 17

הצגת הניקוד.................................................................................. 18

**אקראיות**................................................................................................. 19

הורדת המספר האקראי לטווח........................................................... 20

בדיקה – האם המספר האקראי תקין.................................................. 21

תרשים זרימה של תהליך יצירת פרי חדש............................................ 22

**השהייה**.................................................................................................. 22

**מסך הפתיחה**.......................................................................................... 23

**ביבליוגרפיה**............................................................................................ 23

**הקדמה:**

בפרוייקט זה בחרתי לבצע את המשחק סנייק (Snake) בשפת אסמבלי x86 בעזרת אוסף הפקודות של מעבד 8086.

השתמשתי באסמבלר TASM (Turbo assembler) כדי לקמפל קבצי .asm ולהפוך אותם לקובץ .obj

השתמשתי בלינקר tlinker (Turbo Linker) כדי להפוך את קבצי ה- .obj לקבצי .exe

השתמשתי בDosBox- (אמולטור של מערכת ההפעלה בשם Dos) כסביבת העבודה להרצת הקובץ exe הסופי.

השתמשתי ב-TD (Turbo Debugger) בשביל הרצת קבצי exe במצב Debug (כדי שיהיה ניתן לעקוב אחרי הרגיסטרים, המשתנים, והמחסנית).

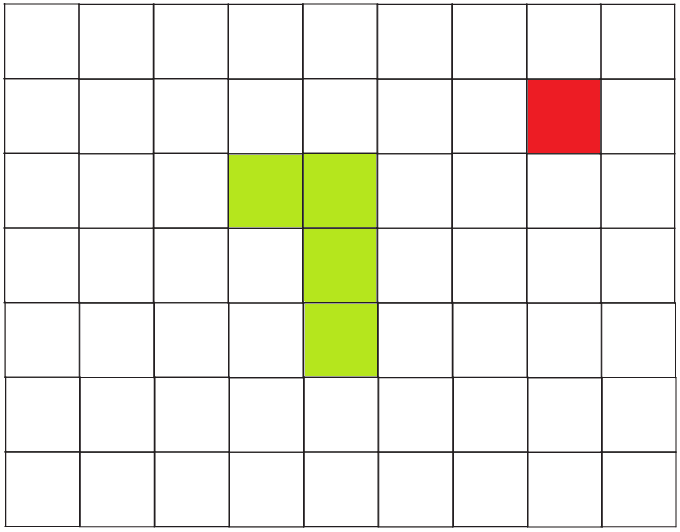
הסיבה שבחרתי במשחק סנייק היא בגלל שזהו משחק שמשלב בתוכו הרבה אלמטים שונים כמו גרפיקה, שליטה באמצעות המקלדת ואקראיות, בעוד שהמשחק נשאר פשוט בבסיסו, כך שאפשר ללמוד ממנו הרבה דברים בסיסיים שישמשו אותי גם לפרויקטים אחרים.

המשחק נוצר בתקופה שבה היו הרבה מגבלות מבחינת חומרה ולכן מפתחי המשחק היו צריכים ליצור משחק ממכר ויעיל בו זמנית, מה שגרם להם לעצב את המשחק כך שיהיה בעל מכניקות פשוטות, לדוגמה: כל פעם שהנחש זז לא צריך לצייר את כל חלקי גופו שוב מכיוון שמה שבאמת זז כל פעם הוא רק הזנב והראש.

המטרה שלי בבניית המשחק היא, כמו מטרת המפתחים המקוריים, ליצור את המשחק בצורה הכי יעילה שאני יכול.

**הסבר על המשחק:**

מסך המשחק "snake" בנוי בצורת רשת, כלומר המסך מחולק לתאים בצורת ריבוע שרק בהם יכולה להיות הגרפיקה (אי אפשר לחרוג מגבולות התאים ולצייר גרפיקה בשני תאים בו זמנית).



במשחק ישנם שני אובייקטים – הנחש והפרי.

הנחש הוא אובייקט שמזדמן במיקום קבוע (אמצא המסך) בתחילת המשחק והוא מורכב מראש וגוף:

הראש יכול לזוז על המסך ב-4 כיוונים שונים בהתאם לקלט השחקן, אבל הראש לא יכול לזוז לכיוון המנוגד לכיוון בו הוא זז ברגע מסוים, כלומר אם הנחש זז ימינה, השחקן לא יכול להזיז אותו שמאלה אלא רק למעלה או למטה.

הגוף מתנהג כמו שובל שעוקב אחרי הראש.

הפרי הוא אובייקט שמזדמן במיקום אקראי על המסך (חוץ מאשר בתוך הנחש) בתחילת המשחק וכל פעם שהנחש אוכל אותו, (הנחש אוכל את הפרי כאשר הראש שלו נמצא באותו מיקום של הפרי).

הנחש מתחיל מאורך התחלתי מסוים וכל פעם שהוא אוכל פרי, הוא מתארך והשחקן מקבל נקודה אחת.

השחקן נפסל כאשר ראש הנחש פוגע באחד מחלקי גופו או כאשר הראש יוצא מגבולות המסך.

מטרת המשחק היא להשיג כמה שיותר נקודות.

משחק זה מורכב מהרבה חלקים שונים כמו קלט המקלדת, ציור למסך, אקראיות ולכן אני אסביר על כל חלק בנפרד ובסוף אציג איך כולם מתחברים.

**השליטה במשחק:**

קבלת הקלט במשחק מתבצעת באמצעות פסיקה מספר 16h שהיא פסיקת המקלדת.

מתוך הפסיקה הזו השתמשתי בפונקציות הבאות:

ah = 1h זוהי פונקציה שמדליקה את ה-zero flag במקרה ונלחץ מקש כלשהו ושמה את קוד ה-ASCII ואת ה-BIOS scan code של המקש ב-al וב-ah בהתאמה.

בגלל שנרצה להמשיך לבצע פעולות אחרות גם אם המשתמש לא לחץ על כלום (כמו הזזת הנחש), נצטרך להשתמש בפונקציה לא חוסמת – כלומר שלא תעצור את הקוד ותחכה לקלט מהמקלדת.

זאת הסיבה לשימוש בפונקציה ah = 1h שהיא פונקציה לא חוסמת לעומת פונקציות קלט אחרות כמו ah = 0h שעוצרות את התוכנית עד שהמשתמש נותן קלט.

**בעיה – :Keyboard buffer**

ה-keyboard buffer הוא מקום קבוע כלשהו בזיכרון ה-BIOS (סגמנט 40h), גודלו הוא 32 סיביות והוא עובד בצורה הזאת:

כל פעם שהמשתמש לוחץ על מקש, אותו המקש נכנס למקום הזה בזיכרון בצורת first in first out, כלומר המקש הראשון שנכנס הוא גם הראשון שיוצא.

בעצם ניתן להבין מכך שהפונקציה ah = 1h לא מחכה ללחיצת המשתמש מכיוון שהיא פשוט לוקחת את המקש שנמצא במקום הראשון ב-buffer (כך שאם ה-buffer ריק אז התוכנית לא תיפסק).

הבעתיות מתחילה מהתנהגותה של הפונקציה הזו, שבשונה מפונקציות קלט אחרות, היא לא מוחקת את המקש האחרון שאותו קלטה מה-buffer מה שאומר שברגע שהמשתמש לחץ על המקש הראשון בתחילת המשחק, אותו המקש יישאר במקום הראשון ב-buffer לאורח כל התוכנית.

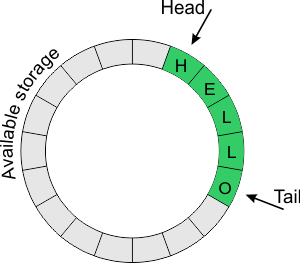
אפילו אם הפונקציה כן הייתה מוחקת את המקש האחרון מה-buffer, ישנה אפשרות שהמשתמש ילחץ על הרבה מקשים בזה אחר זה, ימלא את ה-buffer וכך הוא לא יוכל להשתמש בשום מקש אחר עד שכל המקשים שב-buffer יתקבלו כקלט ולשם כך צריך לנקות את ה-buffer כל פעם לפני בדיקתו.

מסתבר שה-buffer הזה שמור בסגמנט מספר 0040h ב-offset של:

001Ah – head - 16 bit

001Ch – tail - 16 bit

וכאשר ה-head וה-tail שווים, ה-buffer ריק, כך שניתן לגשת למקום הזה בזיכרון בצורה ישירה וכך לאפס את ה-buffer.



במקרה והתוכנה לא זיהתה שום כפתור היא תקפוץ ל-label שנקרא until\_press שהוא בעצם הלולאה האינסופית של המשחק, אם היא כן, צריך לבדוק באיזה כפתור מדובר.

כפתורי הזזת המשחק הם:

**I –** **up**

**J – left**

**K – down**

**L – right**

ויש לכן 2 סיבות:

- הפורמט הזה משומש די הרבה במשחקים שכן המבנה של המקשים האלו במקלדת דומה למבנה מקשי החיצים ולמבנה מקשי ה- w, a, s, d שהם הנפוצים ביותר במשחקים.

- במקרה האותיות האלו גם עוקבות אחד אחרי השנייה ב-ABC ולכן גם קוד ה-ASCII שלהם עוקב (תכונה שתהיה חשובה מאוחר יותר).



**אופטימיזציה - :Switcher**

במקרה רגיל שבו קוד ה- ASCIIשל מקשי המשחק היה "אקראי" היה צורך בהשוואה של הרגיסטר שמחזיק את קוד ה-ASCII של הקלט מהמקלדת (al) לכל אחד ממקשי המשחק:

cmp al, 'w'

je moveUp

cmp al, 'a'

je moveLeft

cmp al, 's'

je moveDown

cmp al, 'd'

je moveRight

אבל במקרה הזה קוד ה-ASCII של המקשים הוא בסדר עוקב ולכן ניתן להשתמש בשיטה אחרת שדומה מאוד לפקודת switch בשפות C.

ניתן ליצור מערך שנקרא switcher בגודל מספר מקשי המשחק (4) ובתוכו לשים את ערכי הכיוון שמתאימים לכיוון שעליו לחץ השחקן כך שסדר הערכים במערך הוא כסדר קוד ה-ASCII של מקשי המשחק, כלומר אם:

I - up – 69h (ASCII)

J - left – 6Ah (ASCII)

K - down – 6Bh (ASCII)

L - right – 6Ch (ASCII)

המערך יהיה:

switcher dw UpValue, LeftValue, DownValue, RightValue

כעת ניתן להחסיר מ-al את ערך קוד ה-ASCII הנמוך ביותר מבין הכפתורים (h69) ולהשתמש בו כאינדקס למערך switcher שממנו יבחר הערך המתאים, לדוגמה אם:

al = 6Ah

אז:

המשתמש לחץ על j, כלומר הוא רוצה לזוז שמאלה.

לאחר החיסור, al = 6Ah – 69h = 1h ולכן התוכנית תיקח את הערך השני במערך שהוא LeftValue.

במקרה שבו מקשי המשחק לא היו בסדר עוקב או בעלי מרווחים לא קבועים, לא היה ניתן לבצע פעולה זו מכיוון שגודל המערך היה יכול להיות גדול מאוד (תלוי במרווחי קוד ה-ASCII של המקשים) והיה מתבזבז מקום בזיכרון.

פעולה זו אמנם יעילה אך היא מעלה בעיה קטנה אחרת – מה יקרה אם המשתמש לחץ על כפתור שהוא לא אחד ממקשי המשחק?

במקרה זה האינדקס (al) יחרוג מגבולות המערך והתוכנית תנסה לקפוץ למקום לא ידוע.

כדי למנוע מקרה זה צריך לבדוק האם al נמצא בגבולות המערך לפני שימושו כאינדקס, אם הוא כן התכנית תמשיך כרגיל, אם לא התוכנית תקפוץ ל-until\_press.

בדרך כלל נצטרך לבדוק האם al גדול מ-h69 וגם האם הוא קטן מ-Ch6 בצורה הזאת:

cmp al, 69h

jl NotInRange

cmp al, 6Ch

ja NotInRange

jmp InRange

אך בגלל החיסור שביצענו מקודם הפכנו את טווח הבדיקה מ-69h-6Ch ל-0-3 ולכן ניתן להוריד השוואה אחת מכיוון שכל מספר שקטן מ-0 הוא גם גדול מ-3 בגלל שמספר שקטן מ-0 עושה underflow למספר המקסימלי לדוגמה: אם al הוא 0 ונוריד ממנו 1, ערך al יהיה FFh0 ולכן הבדיקה תתבצע כך:

cmp al, 3h

ja NotInRange

jmp InRange

לאחר בחירת הערך המתאים, נצטרך לבדוק האם ערך זה מייצג כיוון מאונך לכיוון התנועה של הנחש בזה הרגע, כאשר ערך זה שמור במשתנה HeadDirection.

HeadDirection הוא משתנה שמורכב משני בתים – הבית התחתון מייצג את כיוון תנועת הנחש בציר ה-X והבית העליון מייצג את כיוון תנועת הנחש בציר ה-Y (הכיוון החיובי בציר ה-X הוא ימינה ובציר ה-Y הוא מטה) כך שהכיוון בכל אחד מהצירים מיוצג על ידי אחד משלושת הערכים: 0 – אין תזוזה, 1 – תזוזה בכיוון החיובי, 1- – תזוזה בכיוון השלילי (באסמבלי ניתן לייצג את המספר 1- על ידי המספר המקסימאלי שבמקרה של ביית אחד הוא 0FFh מכיוון שהמספר יקבל overflow ויהפוך ל-0 אם נוסיף לו 1).

בגלל שבמשחק זה אפשר לנוע על ציר 1 כל פעם, אחד משני הערכים חייב להיות 0 תמיד כך שכאשר הכיוונים מאונכים, הבתים שערכם 0, כלומר הצירים שבהם הנחש לא זז יהיו מנוגדים ולכן אם נבצע את הפעולה AND על 2 הערכים, התוצאה תמיד תהיה 0 כאשר הכיוונים מאונכים ותמיד לא תהיה 0 כאשר הם על אותו הציר.

השתמשתי בפקודה TESTשמבצעת את הפעולה AND על 2 הערכים מבלי לשמור את התוצאה, אם דגל האפס דלוק התוכנית קופצת ל-until\_press, אם לא, נשים את ערך הכיוון החדש במשתנה HeadDirection.

**תרשים זרימה של תהליך קבלת הקלט:**

התחלה

תזוזת הנחש בהתאם לכיוונו ואיפוס ה-keyboard buffer

לא

האם ישנו מקש ב-keyboard buffer

כן

שמירת קוד ה-ASCII של המקש שנלחץ ב-al

לא

האם המקש הוא אחד ממקשי המשחק

כן

השמת ערך הכיוון ב-ax בהתאם למקש שנלחץ

לא

האם הכיוון מאונך לכיוון ראש הנחש בזה הרגע

כן

השמת ערך הכיוון במשתנה HeadDirection בהתאם למקש שנלחץ

השמת ערך הכיוון במערך הלוח במיקום ראש הנחש כפי שמוזכר בעמ' 13

**גרפיקה:**

כדי להציג את המשחק למסך, השתמשתי בפסיקה 10h שהיא פסיקת הוידאו והיא משמשת לכל מה שקשור להצגת גרפיקה למסך.

הפסיקה הזאת מאפשרת לבחור את ה-Video Mode של המסך באמצעות הפונקציה ah = 0h.

אני בחרתי ב- mode 13h (320x200, 256 colors, 1 page) בגלל שהוא מציע הרבה צבעים וגם בגלל שהוא בעל רזולוציה יחסית קטנה בהתחשב בעובדה שלא צריך הרבה פרטים במשחק זה, בנוסף רזולוציה קטנה יותר תגרום לפחות צורך ב"הגדלת" כל פיקסל מאוחר יותר.

השתמשתי גם בפונקציה ah = 0Ch שמקבלת כקלט צבע ב-al, את רכיב ה-X של מיקום הפיקסל ב-cx ואת רכיב ה-Y של מיקום הפיקסל ב-dx ומציירת פיקסל אחד למסך בצבע ובמיקום הרצוי.

**מערכת הTiles-:**

מסך המשחק בנוי בצורת טבלה של תאים ריבועיים כאשר ניתן לצייר רק בתא אחד בכל פעם.

מערכת זו היא בעצם הגדלה של כל תא בטבלה מפיקסל אחד, ל-400 פיקסלים ויש לה כמה מטרות:

1. "הגדלת כל פיקסל" במשחק כדי שיהיה נוח לראות אותו (אם כל תא היה פיקסל אחד היה קשה לראות את הנחש והפרי).

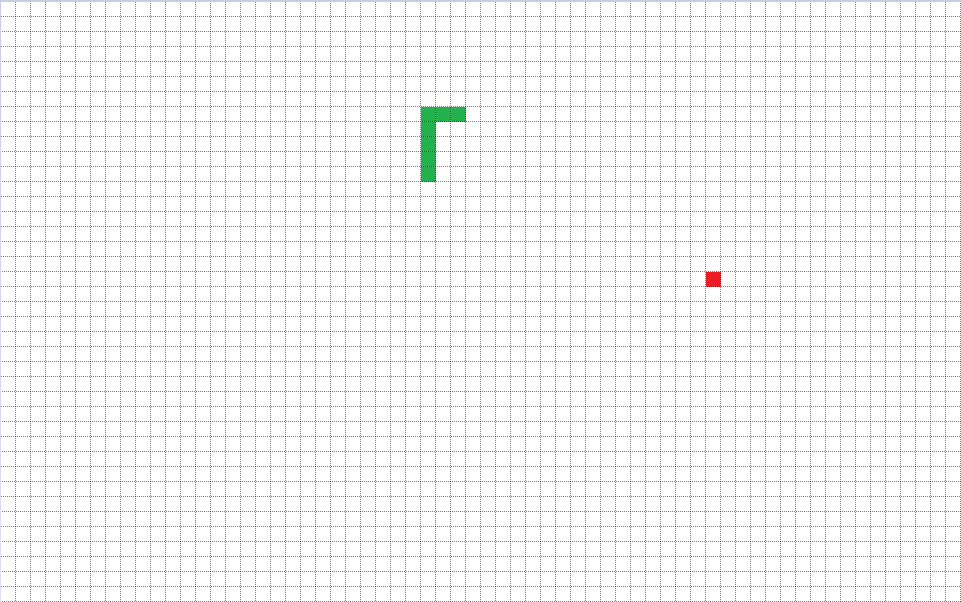
2. הקטנת שטח המשחק (אם כל תא היה פיקסל אחד היה לנחש הרבה שטח להסתובב בו).

3. צמצום משתני המיקום מ-dword ל-word.

לכל תא מוגדל כזה קראתי tile ואורכו של כל tile הוא 20 פיקסלים כלומר השטח של כל tile הוא 20\*20 = 400 פיקסלים.

הסיבה שבחרתי באורך זה היא מכיוון ש-20 מתחלק גם באורך ה-X של המסך (320) וגם באורך ה-Y של המסך (200) כך שיהיה ניתן לבנות מסך של 320x200 ממספר שלם של tiles (16x10) כדי שלא יהיו tiles שיחתכו על ידי גבולות המסך.

המסך כאשר אורך כל תא הוא פיקסל אחד:



המסך כאשר אורך כל תא הוא 20 פיקסלים:



הפונקציה DRAW\_TILE היא פונקציה שמקבלת את המיקום שבו היא צריכה לצייר ב-cx ואת הצבע ב-bl ומציירת tile אחד למסך בצורה הזאת: מתחילים מהפיקסל השמאלי והעליון ביותר, מציירים שורה אחת, יורדים שורה וחוזרים על התהליך עד לקבלת ריבוע.

המיקום מיוצג כ-word כאשר הביית העליון מייצג את רכיב ה-Y של המיקום והביית התחתון מייצג את רכיב ה-X של המיקום (ראשית הצירים היא בפינה השמאלית למעלה של המסך).

הפונקציה בהתחלה מתרגמת את המיקום ב-cx למיקום בזיכרון של הפיקסל הראשון (בפינה השמאלית למעלה) של ה-tile שבו אנו רוצים לצייר.

ניתן לעשות את החישוב הזה באמצעות הכפלה של רכיב ה-X ורכיב ה-Y באורך כל Tile (20) מכיוון שכל המסך מוגדל בקנה מידה של אורך כל Tile.

תוצאות ההמרה נשמרת ב-di וב-dx (רכיב ה-X לא נשמר ב-cx מכיוון שצריך את הרגיסטר הזה כ-iterator בלולאות ולכן כל פעם לפני קריאה לציור הפיקסל, הערך ב-DI מועבר ל-cx), הצבע מועבר מ-bl ל-al.

לאחר מכן, ישנה לולאה מקוננת שרצה TileL פעמים כל אחת כאשר בכל ריצה קוראים לפונקציית ציור הפיקסל ומגדילים את di ב-1 כדי לעבור לפיקסל הבא מימין.

כל פעם שהלולאה הפנימית נגמרת, צוירה שורה אחת של ה-Tile ולכן נגדיל את dx ב-1 כדי לרדת שורה ונחסיר TileL (20) מ-di כדי להחזיר את מיקום ה-X למיקום שממנו הוא התחיל.

לאחר ששתי הלולאות הסתיימו, צויר ריבוע שאורך צלעו 20 פיקסלים.

**תרשים זרימה של הפונקציה DRAW\_TILE:**

התחלה

המרת המיקום ב-cx ל-X ול-Y של הפיקסל הראשון בTile-

ציור פיקסל אחד במיקום ה-X וה-Y

לא

הגדלת ה-X ב-1

האם הקוד הגיע לקטע זה 20 פעמים?

כן

לא

הגדלת ה-Y ב-1

הקטנת ה-X ב-20 כדי להחזירו לשורה שממנה התחיל

האם הקוד הגיע לקטע זה 20 פעמים?

כן

סיום

**תזוזת הנחש:**

במשחק זה אין צורך לצייר את כל חלקי גוף הנחש מחדש עם כל תזוזה מכיוון ששני החלקים שזזים באמת הם ראש הנחש וזנב הנחש ולכן לכל אחד מהחלקים האלו נצטרך משתנה של מיקום ומשתנה של כיוון (כיוון התנועה של הראש לא תמיד שווה לכיוון התנועה של הזנב).

למשתנים האלו קוראים HeadPos, HeadDirection, TailPos, TailDirection, כאשר כולם בגודל word.

משתני המיקום מורכבים משני ערכים: מיקום X של ה-Tile בביית התחתון ומיקום Y של ה-Tile בביית העליון כאשר (0, 0) נמצא במיקום השמאלי למעלה של המסך וימינה הוא הכיוון החיובי של ציר ה-X, למטה הוא הכיוון החיובי של ציר ה-Y.

משתני הכיוון מורכבים גם הם משני ערכים: כיוון ה-X בביית התחתון וכיוון ה-Y בביית העליון, לדוגמה הכיוון 0001h מתאר את הכיוון ימינה כי הביית העליון הוא 0, כלומר הוא לא זז בציר ה-Y והביית התחתון הוא 1 כלומר הוא זז בכיוון החיובי של ציר ה-X, ימינה.

**בעיה – מיקום הזנב:**

אין שום בעיה לעקוב אחר ראש הנחש, שכן כיוון תזוזתו נקבע על פי קלט המשתמש אבל מה לגבי הזנב? נצטרך לעקוב אחרי מיקומו כדי "למחוק" את הזנב עם כל תזוזה.

אפשר לדעת שהמיקום ההתחלתי של הזנב הוא גם המיקום ההתחלתי של הראש מכיוון שהנחש מתחיל מ-Tile אחד אך כיצד נקבע את הכיוון שאליו זז זנב הנחש, הרי כיוונו תלוי גם בכיוון הראש וגם באורכו של הנחש, לדוגמה אם אורך הנחש הוא n והשחקן החליט לפנות ימינה, כיוון הזנב ישתנה לימין רק לאחר n תזוזות.

כדי להתמודד עם בעיה זו, יצרתי מערך שנקרא Board (לוח) שמייצג את המסך עצמו – כל תא במערך הוא בגודל ביית אחד ומייצג Tile אחד על המסך, ובגלל שאני משתמש ב-Video Mode 13h, כמות ה-Tiles בשורה אחת היא 320/20 = 16 וכמות ה-Tiles בעמודה אחת היא 200/20 = 10, כלומר המסך בנוי מ-10\*16 = 160 Tiles ולכן גם אורך המערך יהיה 160.

ערך כל תא במערך הוא מספר בין 0 ל-5 כך שכל מספר מייצג "אובייקט" אחר:

0 - מייצג חלק מגוף הנחש שפונה מעלה

1 - מייצג חלק מגוף הנחש שפונה שמאלה

2 - מייצג חלק מגוף הנחש שפונה מטה

3 - מייצג חלק מגוף הנחש שפונה ימינה

4 - מייצג חלק מגוף הנחש שלא פונה לשום כיוון (ממשיך בכיוונו הרגיל)

5 - מייצג מקום ריק

כל פעם שאנו מציירים או מוחקים חלק מהנחש, נצטרך להוסיף את הערך המתאים גם למערך הלוח.

הבעיה היחידה היא שמשתני המיקום שומרים את המיקום של X ושל Y בנפרד כמו שהוזכר מקודם ולכן נצטרך להמיר מהצורה הזאת לאינדקס במערך שמייצג את אותו ה-Tile.

הפונקציה CONVERT\_POS היא פונקציה שמתרגמת את המיקום לצורת אינדקס במערך הלוח בעזרת הנוסחה .

בעזרת המערך הזה ניתן לרשום את כל הפניות של ראש הנחש כך שהזנב ידע מתי לפנות כדי לעקוב אחרי השובל בצורה הזאת: בכל פעם שהשחקן מחליט לפנות נוצר אינדקס כאשר הערך במערך switcher באינדקס שנוצר הוא הכיוון שאליו השחקן פונה(כפי שתואר בעמ 6), את האינדקס הזה נשים במערך הלוח במיקום הראש בעת הפנייה, לדוגמה אם השחקן החליט לפנות שמאלה, האינדקס שנוצר יהיה 1 ולכן נשים את הערך 1 במיקום הראש שתורגם לאינדקס במערך הלוח באמצעות CONVERT\_POS.

הזנב בודק בכל תזוזה האם במיקום שלו שמתורגם לאינדקס במערך הלוח יש ערך בין 0 ל-3, אם לא הוא ממשיך מבלי לשנות כיוון, אם כן נשים במשתנה TailDirection את הערך במערך switcher באינדקס של הערך שקראנו ממערך הלוח.

בקצרה - ראש הנחש משאיר את כיוון פנייתו במערך הלוח עם כל פנייה כך שכאשר הזנב יגיע למקום שבו הראש פנה, הוא ידע שהוא צריך לפנות.

הסיבה שאנו שמים גם את חלקי הנחש שלא מתארים פנייה בתוך מערך הלוח (הערך 4) היא שנשתמש במערך זה גם מאוחר יותר בחלק בדיקת ההתנגשות של הנחש עם עצמו, ובחלק יצירת המיקום האקראי של הפרי.

**אורך הנחש:**

הנחש מתחיל מגודל מסוים במצב ההתחלתי אבל לא נרצה שהוא יתפוס מספר Tiles לפי גודלו, למשל אם גודלו ההתחלתי הוא 3, לא נרצה שבמצב ההתחלתי הוא יתפוס 3 Tiles.

לכן הנחש מתחיל מ-Tile אחד והוא מתארך לאורכו ההתחלתי האמיתי עם תזוזתו.

לשם כך אנו צריכים 2 ערכים שיתארו את אורך הנחש: הערך הראשון מתאר את אורך הנחש בזה הרגע והשני מתאר את אורכו המקסימלי, כלומר בתחילת המשחק הערך הראשון יהיה 1 מכיוון שבמצב ההתחלתי הנחש תופס Tile אחד והערך השני יהיה 2 מכיוון שכפי שהוזכר בעמ' 19, הנחש אוכל פרי בתחילת המשחק לפני תזוזתו, אורכו המקסימלי גדל ב-1 ולכן אם נרצה שאורכו המקסימלי של הנחש יהיה 3 בתחילת המשחק, הערך השני צריך להיות 2.

בגלל שגודל המסך הוא 160 Tiles, 2 ערכים אלו לא יכולים לעבור את המספר 160, כלומר אפשר לאחסן כל אחד מהם בביית אחד, החלטתי לאכסן אותם ברגיסטר dx כך שהביית התחתון הוא אורך הנחש בזה הרגע הביית העליון הוא האורך המקסימלי של הנחש (אכסנתי את הערכים האלו ב-dx ולא במשתנה מכיוון שהרגיסטר הזה כמעט לא משומש).

**פונקציית התזוזה:**

הפונקציה MOVE אחראית על כל מה שקשור בהזזת הנחש, היא לא מקבלת שום קלט ולא מחזירה שום פלט.

הפונקציה רצה בלולאה אינסופית שמתחילה לאחר טעינת המצב ההתחלתי של המשחק.

בהתחלה הפונקציה בודקת האם כיוון ראש הנחש הוא 0, אם לא התוכנית תמשיך אם כן היא קופצת ל-label שנקרא skip שמדלג על מחיקת הזנב מכיוון שאין שום מטרה למחיקת הזנב כאשר הנחש לא זז, שכן במצב זה מיקום הזנב הוא גם מיקום הראש.

לאחר מכן התוכנית בודקת והאם אורך הנחש כרגע שווה לאורך הנחש המקסימלי, אם כן היא קופצת ל-label שנקרא erase\_tail שבו ה-Tile שנמצא בזנב הנחש "נמחק" גם מהמסך וגם ממערך הלוח, ואם לא נעלה את גודל הנחש כרגע ב-1 ונקפוץ ל-skip כדי לדלג על המחיקה.

זאת מכיוון שכאשר הנחש הגיע לאורכו המקסימלי, נרצה להוריד את זנבו בנוסף להוספת הראש כדי שגודלו ישמר, לעומת זאת אם הנחש לא הגיע לאורכו המקסימלי, נרצה להגדיל אותו בכך שנדלג על המחיקה ונעלה את אורכו כרגע ב-1 חוץ מהמצב שבו הנחש לא זז כי לא נרצה להגדיל את אורך הנחש כאשר הוא לא זז, ולכן צריך גם לבדוק האם כיוון הראש של הנחש שווה ל-0.

אם הקוד נכנס ל-label שנקרא erase\_tail, לפני מחיקת הזנב והוספת כיוונו למיקומו, המיקום של הזנב מומר לאינדקס של מערך הלוח ובודקים האם את הערך שנמצא במערך באינדקס הזה הוא בין 0-3, אם לא הקוד ממשיך כרגיל ואם כן, משתמשים בערך הזה כאינדקס במערך switcher כדי למצוא את הכיוון שאותו מייצג הערך ושמים את הערך הזה במשתנה TailDirection.

לאחר מכן ללא שום תנאי, מוסיפים את כיוון ראש הנחש למיקומו ומציירים את ראש הנחש למסך (מציירים Tile בצבע ירוק במיקום החדש של ראש הנחש) ומוסיפים אותו למערך הלוח.

**בדיקת ההתנגשות:**

בדיקת ההתנגשות היא פעולה פשוטה מאוד הודות למערך הלוח.

כדי לבדוק אם הנחש בפגע בקיר, כלומר יוצא ממסך המשחק, צריך רק לבדוק לאחר תזוזת ראש הנחש האם הרכיב ה-X של הראש גדול מ-0Fh, אם כן השחקן נפסל ונכנס ל-label שנקרא lose.

אם לא, צריך לבדוק האם הרכיב ה-Y של הראש גדול מ-09h, אם כן השחקן נפסל ונכנס ל-label שנקרא lose.

אם לא, נשים את הערך 4 שמתאר חלק רגיל של גוף הנחש במיקום ראש הנחש במערך הלוח ונבדוק האם המספר במיקום ראש הנחש במערך הלוח לפני השמת הערך 4 הוא בין 0 ל-4, אם כן הנחש פגע בעצמו ולכן השחקן נפסל ונכנס ל-lose.

אם לא, נבדוק האם מיקום ראש הנחש שווה למיקום הפרי, אם כן הנחש אכל את הפרי ולכן נזמן פרי חדש על ידי קפיצה ל-label שנקרא new\_fruit.

אם לא הנחש לא פגע בכלום וממשיך כרגיל.

**הפסד:**

במקרה שהנחש יצא מגבולות המסך או פגע בעצמו, הקוד נכנס ל-label שנקרא lose שבו כל משתני המשחק מאופסים לערכם ההתחלתי והמחסנית מתרוקנת.

כל פעם שנוסף ערך למחסנית על ידי פקודת push ערכו של הרגיסטר SPיורד ב-2 מכיוון שהכנסנו 2 בתים ולכן כדי "לאפס" את המחסנית נוסיף ל-SP את המספר 2\*n כאשר n הוא מספר הערכים שהוכנסו למחסנית (במקרה שלנו ישנם 3 ערכים ולכן נוסיף 6 ל-SP).

לאחר מכן הקוד קופץ ל-label שנקרא setup שנמצא בתחילת הקוד.

**אכילת פרי:**

במקרה שהנחש אכל פרי, הקוד נכנס ל-label שנקרא new\_fruit שבו נוצר מיקום אקראי חדש מהמשתנה Rand, הפרי החדש מצויר במיקום הזה, האורך המקסימלי של הנחש גדל ב-1, והניקוד גדל ב-1.

**תרשים זרימה של פונקציית התזוזה:**

התחלה

איפוס כל המשתנים

הוספת ה-X של ראש הנחש ל-Rand

"ניקוי" המחסנית

קפיצה לתחילת המשחק

האם כיוון הראש שווה 0

לא

כן

סיום

שמירת מיקום הזנב ב-PrevTailPos

האם אורך הנחש כרגע שווה לאורכו המקסימלי

כן

לא

האם הערך במערך הלוח במיקום הזנב קטן מ-4

הגדלת אורך הנחש כרגע ב-1

כן

הוספת כיוון הראש למיקומו

לא

מציאת הכיוון ב-switcher לפי ערך זה והשמתו ב-TailDirection

ייצור פרי חדש

ציור Tile ירוק במיקום הראש

הגדלת האורך המקסימלי של הנחש ב-1

השמת הערך 5 במערך הלוח במיקום הזנב

כן

האם רכיב ה-X של ראש הנחש יוצא מהמסך

ציור Tile שחור במיקום הזנב

לא

הגדלת הניקוד של הנחש ב-1 (BCD)

כן

האם רכיב ה-Y של ראש הנחש יוצא מהמסך

כן

הוספת כיוון הזנב למיקומו

כן

לא

האם במערך הלוח באינדקס של מיקום ראש הנחש (לפני הפעולה הקודמת) ישנו חלק מהנחש

האם מיקום ראש הנחש שווה למיקום הפרי

לא

השמת ערך שמתאר חלק מהנחש במערך הלוח באינדקס של מיקום ראש הנחש

לא

השהיית הקוד לזמן מסוים

איפוס ה-keyboard buffer

הדפסת הניקוד ל-standard output

סיום

**ניקוד המשחק:**

בתחילת המשחק ניקוד השחקן הוא 0 ועולה ב-1 בכל פעם שהנחש מתארך.

מכיוון שמסך המשחק בנוי מ-160 Tiles והאורך ההתחלתי של הנחש הוא 3, המספר המקסימלי של הנקודות שהשחקן יכול לצבור במשחק אחד הוא 157 (כי במצב זה האורך של הנחש הוא 160 ואין לו לאן להתקדם מבלי להיפסל), כלומר הנקודות צריכות להיות מיוצגות על ידי שלושה ספרות.

**אחסון הניקוד:**

אם נרצה להציג למשתמש ניקוד קריא, נצטרך להציג את המספר בצורה עשרונית ולכן עדיף לאכסן את הניקוד בצורה עשרונית במקום כמספר רגיל, אחרת יהיה קשה להפוך את הניקוד לאותיות ה-ASCII המתאימות.

כדי לאחסן את הניקוד כמספר עשרוני, השתמשתי בשיטת אחסון שנקראת BCD.

BCD (binary coded decimal) היא שיטת אחסון של מספרים עשרוניים שיש לה 2 וריאציות מרכזיות: packed BCD, unpacked BCD:

unpacked BCD היא שיטה ששומרת כל ספרה של מספר בנפרד בביית אחד, לדוגמא:



המספר 91 מיוצג על ידי 1 בביית הראשון ו-9 בבית השני.

packed BCD היא שיטה ששומרת כל ספרה של מספר בנפרד ב-4 סיביות, לדוגמא:



המספר 91 מיוצג על ידי 1 ב-4 הסיביות הראשונות ו-9 ב-4 הסיביות האחרונות.

אמנם שמירה בצורת unpacked תופסת יותר מקום לעומת packed, אבל יותר נוח להפוך את המספר ל-ASCII כאשר כל ספרה נמצאת בביית נפרד לעומת כאשר יש 2 ספרות באותו ביית.

בחרתי להשתמש בשיטת packed BCD רק על 2 הספרות הראשונות ולהשאיר את הספרה האחרונה כספרה רגילה (ספרה זו יכולה להגיע רק עד ל-1 ולכן היא לא חייבת להיות BCD), כלומר גודל המשתנה ששומר את הניקוד הוא 16 סיביות (מכיוון ששלוש ספרות ב-packed תופסות 12 ביט) ושם המשתנה הוא ScoreNumber.

ScoreNumber מתחיל מ-0 ובכל פעם שהנחש אוכל פרי, קורה התהליך הבא:

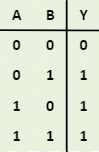
ScoreNumber עולה ב-1, הוא מתוקן להיות מסוג packed BCD במקרה שפעולת החיבור משבשת את המספר בעזרת פקודת DAA ובמקרה של overflow של הביית התחתון, מוסיפים לביית העליון 1 (פקודת DAA לא עושה זאת אוטומטית).

ערכו ההתחלתי האמיתי של ניקוד השחקן הוא בעצם 0FF99h מכיוון שכפי שהוזכר בעמ' 19, הנחש אוכל פרי ממש בתחילת המשחק עוד לפני תזוזתו ובכך מעלה את ניקודו ב-1 ולכן אנו צריכים להתחיל את משתנה זה ממספר שאם נוסיף לו 1, נקבל 0.

בגלל ששתי הספרות בביית התחתון הם מספרי BCD, המספר המקסימלי שלהם הוא 99h ובגלל שהספרה בביית העליון היא ספרה רגילה, המספר המקסימלי של הביית הזה יהיה 0FFh ולכן הערך ההתחלתי צריך להיות 0FF99h.

לאחר מכן נצטרך להמיר את המספר ל-ASCII בדרך הזאת: נמיר כל ספרה של packed BCD ל-unpacked BCD ונוסיף 30h מכיוון שערך זה מייצג את הספרה הקטנה ביותר ב-ASCII (0).

אבל בעצם אפשר לבצע את הפעולה OR על ה- unpacked BCDעם 30h במקום החיבור מכיוון שארבעת הביטים הראשונים של 30h הם 0 ולפי טבלת האמת של OR ניתן לראות שכאשר A הוא 0 התוצאה תהיה B ולכן ארבעת הביטים שדרושים כדי לייצג מספר BCD יועתקו לארבעת הביטים הראשונים של 30h, כלומר ה-BCD יתווסף אליו.



**הצגת הניקוד:**

כדי להדפיס את הטקסט למסך השתמשתי בפונקציה ah = 09h של פסיקה 21h, שמקבלת מצביע למערך של אותיות שחייב להסתיים בסימן '$' ב-dx ומדפיסה את המחרוזת למיקום ה-cursor במסך.

הסיבה שהשתמשתי בפונקציה הזאת בניגוד לפונקציות אחרות היא שהפונקציה הזאת משתמשת ב-standard output כך שניתן להדפיס מחרוזת שלמה במקום כל אות בנפרד ואפשר לחזור לאותו מיקום של תחילת הטקסט באמצעות איפוס ה-cursor למיקום התחלתי מסוים על המסך.

ניתן לקבוע את המיקום של ה-cursor על המסך עם הפונקציה ah = 02h של פסיקה int 10h, שמקבלת את מיקום ה-cursor ב-dx (מספר השורה ב-DH ומספר העמודה ב-dl).

הצגת הניקוד למסך צריכה להתבצע עם כל תזוזה של הנחש, לאחר ציור הנחש וזאת מכיוון שציור הניקוד לאחר הנחש "ידרוס" חלק מהנחש במקרה שבו הוא עובר מעליו.

**אקראיות:**

במשחק זה נרצה שהפרי שאותו הנחש צריך לאכול יהיה במיקום אקראי על המסך ולשם כך צריך לייצר מספרים אקראיים.

ישנן הרבה דרכים שבהם ניתן לייצר מספרים אקראיים, כמו לדוגמה פעולות אריתמטיות מסובכות או שימוש בשעון, אבל במקרה הזה בחרתי להשתמש בתזוזת השחקן כדי לקבוע אקראיות מכיוון שזוהי הדרך הפשוטה והאפקטיבית ביותר להשיג אקראיות במשחק מהסוג הזה, שבו ישנה תזוזה "חופשית" ואנשים שונים ינועו בצורה שונה (בעצם נעשה שימוש בעובדה שהשחקן מתנהג בצורה אקראית בהיותו בן אדם).

הסיבה לאפקטיביות של שיטה זו במשחק הספציפי הזה היא העובדה שתזוזת השחקן משפיעה על מיקום התפוח, ובתמורה מיקום התפוח משפיע על תזוזת השחקן כך שנוצרים מספרים מאוד אקראיים בלי שום דפוס, מה שיחסית מסובך להשיג מחישובים אריתמטיים.

אך בכל זאת צריך להתחיל מאיזשהו מספר אקראי עוד לפני תזוזת השחקן ולכן נצטרך להשתמש בשיטה אחרת כדי להשיג מספר אקראי רק פעם אחת (ברגע הרצת המשחק).

אם נשתמש בזמן המערכת בתור מספר אקראי, מכיוון שיעשה בו שימוש רק פעם אחת בתחילת ההרצה, הוא יהיה מספיק אקראי כך שלא צריך לבצע שום פעולה אריתמטית.

כדי להשתמש בזמן השתמשתי בפונקציה ah = 0h בפסיקה 1Ah שהיא פסיקה שכל מטרתה היא להחזיר את זמן המערכת.

הפסיקה מחזירה את מספר תקתוקי השעון שעברו מאז חצות (בערך 18.206 תקתוקים בשנייה) לרגיסטרים cx ו-dx כך ש-dl משתנה הכי מהר ו-ch משתנה הכי לאט ולכן כדי להשיג מספר יותר אקראי, צריך לקחת רגיסטר שקרוב יותר ל-dl.

את המספר האקראי הזה שמתי במשתנה שנקרא Rand שגודלו 16 סיביות שממנו נחשב אחר כך את המיקום האקראי בעזרת צמצומו לטווח והשמתו במשתנה שנקרא FruitPos ששומר את מיקום הפרי.

המשתנה FruitPos מתחיל מהערך ההתחלתי של מיקום הראש וזאת מכיוון שנרצה שבתחילת הקוד עוד לפני שהשחקן זז, הנחש יאכל פרי ובכך יצייר למסך פרי חדש, אחרת היינו צריכים לצייר את הפרי פעם נפרדת בתחילת הקוד, מה שלא כל כך יעיל.

הבעיה היחידה היא שאכילת הפרי בתחילת המשחק עוד לפני תזוזת הנחש גורמת לאורך הנחש לגדול ב-1 ולמספר הנקודות לגדול ב-1, ולכן הערך ההתחלתי של שניהם צריך להיות 1 פחות ממה שהוא אמור להיות.

לאחר מכן, המשתנה Rand ישתנה כל פעם שהנחש זז לכיוון כלשהו בצורה הזאת: נוסיף את מיקום ה-X של ראש הנחש ל-Rand, ובהנחה שתזוזת השחקן תהיה שונה בכל משחק, לא צריך שום חישובים נוספים, כלומר הקוד יעיל מאוד, ובנוסף נדיר שיהיו 2 משחקים זהים.

**הורדת המספר האקראי לטווח:**

נרצה להוריד את המשתנה Rand לטווח של מספר ה-Tiles במסך (160), כדי לייצר אינדקס למערך הלוח וזאת כדי שיהיה קל יותר לבדוק האם הפרי הזדמן בתוך הנחש מאוחר יותר.

בדרך כלל נרצה להשתמש בפעולת div כדי לחלק מספר כלשהו לטווח מסוים ולקחת את השארית (mod) כדי שהמספר יתאים לטווח אך זוהי פעולה ארוכה שעדיף להימנע מהשימוש בה.

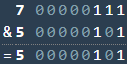
אם נשתמש בפעולת AND על 2 מספרים כלשהם 1N ו-N2, התוצאה תמיד תהיה מספר שנמצא בטווח בין 0 ל-2N (אותו מספר יהיה גם בטווח בין 0 ל-1N), אבל הבעיה היא שהמספר הזה לא חייב להיות בהכרח כל מספר בטווח 0 בין ל-2N.

כדי לראות למה, צריך להסתכל על הפעולה AND עצמה:



לפי הטבלה הזאת ניתן לראות שכאשר A הוא 1 התוצאה תהיה B וכאשר A הוא 0 התוצאה תהיה 0, כלומר פעולת AND בין 2 מספרים שומרת את הביטים של אחד מהמספרים כל עוד הביטים של השני הם 1.

זאת הסיבה שהפעולה גורמת ל-N1 להיכנס לתחום בין 0 ל-2N אבל למה התוצאה לא יכולה להיות כל מספר בין 0 ל-N2? נסתכל על מצב שבו נרצה להכניס איזשהו מספר לטווח בין 0 ל-5 (101 בבינארי):



אפשר לראות שהביט השני של חמש הוא 0 ולכן לא משנה מה יהיה הביט השני במספר העליון, הביט השני בתוצאה יהיה תמיד 0, ולכן המספר 3 לדוגמה (11 בבינארי) לא יכול להיות התוצאה אף פעם למרות שהוא בטווח בין 0 ל-5 מכיוון שהביט השני של 3 הוא 1.

כדי לצמצם את N1 לטווח בין 0 ל-N2 כך שהתוצאה יכולה להיות כל מספר בין 0 ל-N2, נצטרך שN2- יהיה מספר שכל הביטים שלו הם 1:

1 – 1

3 – 11

7 – 111

15 – 1111

המשותף לכל המספרים האלו היא הצורה *.*

*אם נרכיב ממספרים מהצורה הזאת את הטווח שלנו, נוכל לבצע את הפעולה AND על המספר האקראי עם כל אחד מהמרכיבים ולחבר את כולם ביחד לדוגמה במקרה שלנו נרצה טווח של 0-159 ולכן נמצא את המספרים מהצורה שמרכיבים את המספר 159: .*

*הבעיה היחידה היא שלא נרצה להשתמש באותו מספר אקראי לכל אחד מהמרכיבים מכיוון שאז יכול להיות מצב שמספרים מסוימים לא יוכלו להתקבל כתוצאה ולכן נשנה את המספר האקראי אחרי כל פעולת AND על ידי הוספת הביית העליון של המספר האקראי לביית התחתון.*

***בדיקה – האם המיקום האקראי תקין***

*נצטרך גם לבדוק האם במיקום האקראי שנוצר יש כבר את אחד מחלקי גוף הנחש - פשוט נשווה את הערך שנמצא במערך הלוח במקום המספר האקראי עם המספר 5.*

*אם הם שווים, המיקום האקראי נופל על מקום ריק ולכן הפרי מונח שם.*

*אם הם לא שווים, המיקום האקראי נופל על אחד מחלקי גוף הנחש ולכן נצטרך לשים את הפרי במקום אחר.*

*לא נרצה להמשיך להגריל עוד מספרים אקראיים עד שאחד מהם ייפול על מיקום ריק מכיוון שתיאורית יכול לקרות מצב שבו הפרי לא יזדמן מכיוון שמיקומו ימשיך ליפול על הנחש וגם ככל שהנחש גדל, כך יש פחות סיכוי שמיקום הפרי יתאים, לדוגמה: אם הנחש בגודל של 159, סטטיסטית נצטרך 159 איטרציות של הגרלת מספר אקראי כדי למצוא מקום ריק ולכן נגריל 3 מספרים אקראיים (נשנה את המשתנה Rand בכל איטרציה בעזרת פקודת ROL) ואם אף אחד מהמספרים לא מתאימים, הפרי מזדמן במיקום שממנו זז זנב הנחש מכיוון שזהו המיקום היחידי מכל הלוח שאפשר להיות בטוחים שהוא ריק תמיד ולכן נשים את המיקום הזה ששמור במשתנה PrevTailPos ב-cx, אם אחד מהם כן מתאים התוכנית תקפוץ ל-label שנקרא use\_random שבו האינדקס של מערך הלוח מתורגם לרכיבים של מיקום רגיל על ידי הנוסחאות: , , לאחר מכן שמים את הרכיבים האלו ב-cx כמו משתנה מיקום רגיל.*

*בסופו של דבר שני המקרים מובילים לlabel- שנקרא finally שבו נשים את הערך ב-cx במשתנה FruitPos ונצייר את הפרי למסך.*

***תרשים זרימה של תהליך יצירת פרי חדש:***

האם המיקום באינדקס שנוצר הוא בתוך הנחש

כן

לא

לא

כן

סיום

השמת cx במשתנה FruitPos וציור הפרי למסך

השמת המיקום שממנו זז זנב הנחש ב-cx

הפיכת האינדקס למיקום רגיל והשמתו ב-cx

התחלה

צמצום Rand לטווח 0-159 (אינדקס למערך הלוח)

האם התוכנית הגיעה לקטע זה 3 פעמים

"ערבוב" Rand ע"י הפקודה ROL

**השהיה:**

כדי למנוע מהנחש לזוז בלי הפסקה וכדי לשלוט במהירותו, צריך להשהות את הקוד אחרי כל תזוזה של הנחש ולכן ישנה פונקציה בקוד שאחראית על "השהיית הקוד" שנקראת DELAY.

הפונקציה משתמשת בפונקציה ah = 0h int 1Ah, כפי שהוסבר בעמ' 19.

הפונקציה מקבלת ב-SI את מספר המחזורים של טיימר המערכת שצריך לחכות (טיימר המערכת פועל בקצב של 18.2 מחזורים בשנייה).

הפונקציה בהתחלה שומרת את הזמן שבה היא התחילה ונכנסת ללולאה שבה לוקחים את הזמן ומשווים את ההפרש של הזמן כרגע והזמן ההתחלתי, לזמן שנמצא ב-SI: אם הוא קטן יותר מ-SI, לא עבר מספיק זמן והלולאה ממשיכה ואם הוא גדול או שווה ל-SI, עבר מספיק זמן והפונקציה מסתיימת (שיטה זו נקראת "Spinlock").

**מסך הפתיחה:**

מסך הפתיחה הוא מסך שעולה פעם אחת כאשר המשחק עולה, גם אם השחקן נפסל הוא לא יראה את המסך הזה שוב.

המסך מורכב מרצף של אותיות ASCII שנמצאות במשתנה TitleText (המחרוזת בקוד לא נראית כמו במסך הפתיחה מכיוון שבקוד לא ניתן להמחיש את הירידות שורה – 0Ah).

הסיבה שיש הרבה רווחים במחרוזת אחרי כל ירידת שורה היא שירידת השורה גורמת ל-cursor לחזור לתחילת השורה ולכן צריך רווחים כדי להחזיר את הטקסט לשורה הנכונה.

כדי להדפיס את האותיות האלו תחילה נצטרך לבחור את המיקום ההתחלתי על המסך שבו נרצה להתחיל להדפיס את המחרוזת ולכן נשתמש פונקציה ah = 2 של פסיקה 10h שנותנת לנו את האפשרות לבחור את מיקום ה-cursor שקובע את מיקום תחילת ההדפסה כמו שהוזכר בעמ' 18.

לאחר מכן נשתמש בפונקציה ah = 09h של פסיקה 21h שנותנת לנו להדפיס מחרוזת שנגמרת ב-'$' ל-standard output כפי שהוזכר בעמ' 18.

לאחר ההדפסה, נשתמש בפונקציה ah = 0 של פסיקה 16h שעוצרת את הקוד עד שהמשתמש לחץ על מקש כלשהו, מוציאה את המקש הנלחץ מה-keyboard buffer ושמה את קוד ה-ASCII שלו ב-al, במקרה הזה לא משנה לנו על איזה מקש השחקן לחץ, המשחק מתחיל אם יש קלט כלשהו.

**ביבליוגרפיה*:***

*טבלת הפסיקות -* <http://www.ctyme.com/intr/int.htm>

מפת נתוני ה-BIOS - <https://stanislavs.org/helppc/bios_data_area.html>

הסבר על ה-Keyboard Buffer - <http://www.fysnet.net/kbuffio.htm>