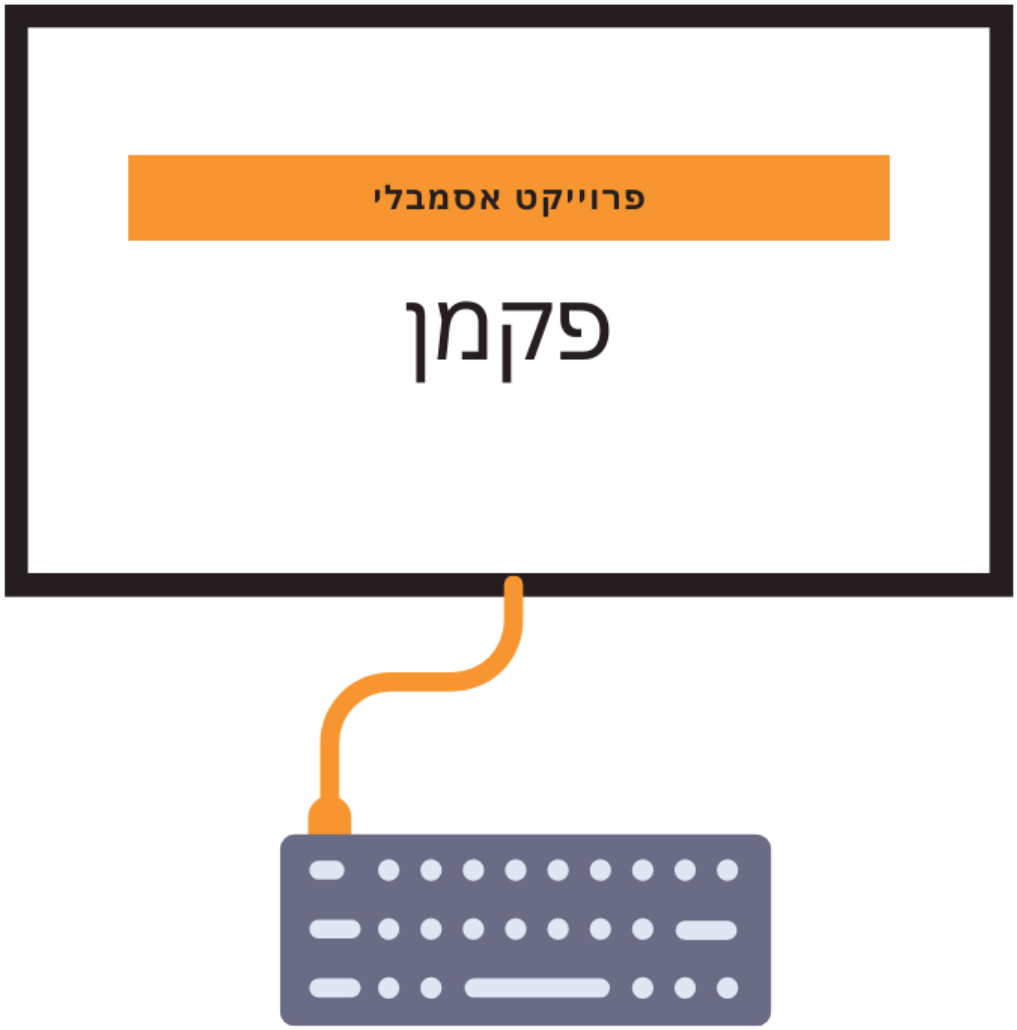


שם: רוני ניסן
ת"ז:327938510
מורה: אופיר שביט
כיתה: י-1
בית הספר: איש שלום

תוכן עניינים

מבוא

שם העבודה: Pacman

שם הקובץ: Main.asm

קבצים נלווים:

- GameEqu.asm
- GameData.asm
- GameProc.asm
- OepnEqu.asm
- OpenData.asm
- OepnProc.asm
- Main.asm
- 1.bmp
- 2.bmp
- 3.bmp
- 4.bmp
- Game.bmp
- GameOver.bmp
- Inst.bmp
- Lb.bmp
- Lose.bmp
- NA.bmp
- PE.bmp
- Play.bmp
- PN.bmp
- PS.bmp
- PW.bmp
- Screen.bmp
- Win.bmp
- 1r.bat
- 1d.bat
- TD.tr

סביבת העבודה

ב Turbo Assembler יש מספר שלבים ליצירת קובץ ההפעלה:

1. יצירת קובץ assembly (סיומת asm) בסביבות עבודה כמו ++notepad או atom.
2. פתיחת DosBox ובאמצעות פקודות cmd פשוטות (כמו cd, dir וכו') להגיע לתיקייה בה נשמר קובץ האסמבלי מהסעיף הראשון.
   יש לשים לב שבאותה תיקייה בה נשמר קובץ האסמבלי נמצאים גם הקבצים החיוניים לתוכנית לפעול (כמו קבצי bmp) וקבצי קומפילציה.
3. באמצעות פקודת tasm/zi נריץ קובץ TASM שמתרגם את קובץ האסמבלי לשפת מכונה (קובץ obj).
4. לאחר מכן נשתמש בפקודת tlink/v שתקשר (link) בין קבצים שונים כדי ליצור תכנית אחת.
   פעולה זו תתרגם את הקובץ בשפת המכונה (obj) לקובץ הרצה (exe).
5. כעת, שיש לנו קובץ הרצה נוכל להריץ אותו או לבחור שלדבג- Turbo debugger (פקודת td ולאחריה שם קובץ ההרצה).

סביבת פיתוח

atom editor
windows Paint
Adobe Photoshop

סביבת הרצה

DosBox

נושא העבודה

במשחק PACMAN המשתמש שולט על הפקמן שעובר במבוך תוך שמבצע מספר דברים:

1. צבירת נקודות- המבוך מלא בנקודות אותן צריך לאכול הפקמן כדי להרוויח ניקוד
2. בריחה- במשחק ישנן 4 רוחות אשר מסתובבות באופן רנדומלי ואוטונומי ברחבי המבוך. מטרתן היא לתפוס את הפקמן, דבר שיוביל לפסילה של השחקן.

חלק חשוב במשחק הוא המעבר של הפקמן מצד אחד של המבוך לאחר במעין שיגור, מה שמסייע לו במנוסה מהרוחות.

מטרתו של השחקן היא לסיים לאכול את כל הנקודות על גבי המסלול לפני שיאכל על ידי הרוחות.

במשחק שלי, הוספתי מגבלת זמן של 90 שניות לניצחון.

אופן ההפעלה

יש להריץ את הקובץ Main.asm בסביבת ההרצה DosBox

מסך הפתיחה:

במסך ישנם ארבעה כפתורים:

הלחצן **QUIT**- ליציאה

הכפתור יחזיר את המשתמש ל cmd בסביבת ההרצה.

הלחצן **GAME INSTRUCTIONS**- להוראות המשחק

יש ללחוץ על החיצים בתחתית המסך כדי לעבור בין עמודי ההוראות.

כדי לחזור לתפריט הראשי יש ללחוץ כפתור הכיבוי בפינה השמאלית העליונה.

הלחצן **LEADERBOARD** - טבלת מובילים

יראה הודעת NOT AVAILABLE ויחזיר את המסך בחזרה לתפריט הראשי לאחר זמן קצר.

הלחצן **PLAY** - למשחק

יעלה את מסך המשחק

- הלחצן **QUIT**- נשלט על ידי העכבר ויחזיר לתפריט הראשי בכל שלב במשחק.
- חיצים [**A,S,D,W**]- ישמשו לתזוזה של הפקמן לכיוונים שונים.

מסך ניצחון

המסך יופיע כשנגמרו כל הנקודות על המסלול ויחזיר לתפריט הראשי.

מסך הפסד

המסך יופיע כשנגמר זמן המשחק ויחזיר לתפריט הראשי.

מסך יציאה

המסך יופיע בעת לחיצה על כפתור היציאה ויחזיר לתפריט הראשי.

גרסאות המערכת

גרסה נוכחית:

גרפיקה:

- ❖ מבוך המשחק עם נקודות.
- ❖ כפתורים שנצבעים לפי מיקום העכבר.

זמנים:

- ❖ טיימר המצביע על משך המשחק.

קלט/ פלט:

- ❖ קלט מקלדת לכיווני הפקמן
- ❖ קלט עכבר לכפתורים

אלגוריתם

- ❖ קיים אלגוריתם המחשב את המיקום הבא של הפקמן בהתאם לכיוון תנועתו ומרחקו מהגבול.
- ❖ אלגוריתם נוסף אשר מחשב האם העכבר נמצא בטווח כפתור מסוים על פי פרמטרים שנשלחים.
- ❖ אלגוריתם שבודק האם נאכלו כל הנקודות על גבי המבוך.

גרסה עתידית:

גרפיקה:

- ❖ תהיה התאמה טובה יותר ביחס לגודלו של הפקמן לבין מרחקו מהגבולות.
- ❖ בכל צעד שאוכל הפקמן את הנקודות פיו ייסגר ויפתח.
- ❖ הפקמן תמיד יאכל נקודה אחת בשלמותה ולא חצי.

זמנים:

- ❖ טיימר שסופר מהסוף להתחלה.

אלגוריתם

- ❖ מימוש אלגוריתם (שנכתב ב java) לאסמבלי האחראי על התנועה האוטונומית של הרוחות במשחק.

כתיבה לקבצים:

- ❖ הצגת טבלת מובילים (באמצעות קלט מהמשתמש ושמירת ניקוד).

שמע:

- ❖ התנועה של הפקמן תהיה מלווה בצליל.

תיעוד והסבר הפתרון

התכנית מורכבת מקובץ עיקרי, main.asm המאחד תחתיו מספר קבצים נוספים:

- **Game**
  - gameEqu.asm
  - gameData.asm
  - gameProc.asm

- **Open**
  - openEqu.asm
  - openData.asm
  - openProc.asm

הקובץ הראשי, main קורא לפעולות העיקריות [ה main של כל אחד מהקבצים] ומאחד אותן תחתיו.

הלוגיקה- אלגוריתם מרכזי

מסך הפתיחה

התכנית בתחילתה פותחת את מסך הפתיחה המתפצל לשלושה חלקים; מסך הוראות, טבלת מובילים והמשחק עצמו.

מסך הוראות

לחיצה עם העכבר על החץ הימני תעביר לעמוד הבא והחץ השמאלי לעמוד הקודם. בכל דף שהוא ניתן ללחוץ על הכפתור יציאה בפינה השמאלית תחזיר למסך הפתיחה.

טבלת מובילים

יעלה מסך של NOT AVAILABLE והמשתמש יוחזר לתפריט הראשי.

המשחק

בעת לחיצה על כפתור המשחק במסך הבית, המשתמש מועבר למסך המשחק ומספר פעולות נכנסות לפעולה:

1. הצגת מסך המשחק
2. אתחול פרמטרים של הפקמן תוך הצגת דמותו של הפקמן (במידה וזו אינה הפעם הראשונה שנכנסים למשחק, הפרמטרים משתנים. על כן בכל כניסה לפרוצדורת המשחק יש לאתחל את כל הפרמטרים).
3. הפעלת טיימר אסינכרוני והצגתו

לאחר אתחול המשחק, מופעלת הלולאה הראשית ובה מספר פעולות החוזרות על עצמן:

1. הצגת ניקוד
2. החבאת העכבר
3. בדיקת יציאה - מחזירה אל מסך הפתיחה
   3.1. לחיצה על כפתור יציאה
   3.2. זמן תם
   3.3. ניצחון
4. קבלת קלט המייצג את הכיוון אליו יתקדם הפקמן בתזוזה הבאה
5. חישוב הצעד הבא של הפקמן בהתאם לגבולות

6. חישוב הניקוד שמתווסף בין הנקודה הנוכחית לנקודה הבאה
7. שינוי מקומו [גרפית ובקוד] של הפקמן בהתאם לסעיפים 4 ו- 5.
   7.1. מחיקת הפקמן והניקוד אותו דרס בתנועתו.
   7.2. ציור הפקמן במיקומו החדש

פעולות התזוזה של הפקמן

1. בדיקה האם נלחץ מקש במקלדת
   1.1. אם לא, חזרה לתחילת הלולאה הראשית [עדכון פרמטרים וכו'] ושאילת השאלה 1 שוב.
   1.2. אם כן, בדיקה - איזה לחצן נלחץ.
2. בדיקה האם הלחצן רלוונטי (מייצג כיוון ['a', 'w', 'd', 's'])
   2.1. אם לא, חזרה ללולאה הראשית.
   2.2. אם כן, עדכון ערך הנקודה הבאה אליה יתקדם הפקמן והצגתו שם.

אלגוריתם- חישוב ערך ה **X/ Y** החדש של הפקמן

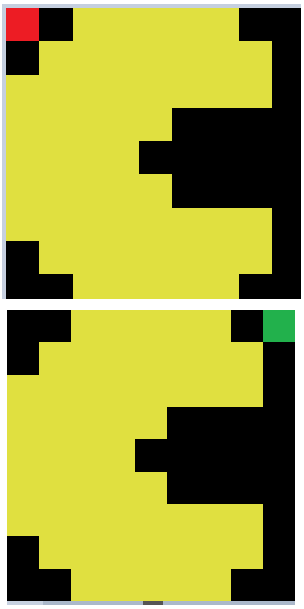
מטרת האלגוריתם היא להגדיר את הגבולות של הפקמן במשחק. באמצעות חישוב וקריאת הפיקסלים מהמסך ניתן לדעת האם הפקמן מתנגש בקיר ואם כן כיצד לפעול.

הפעולה FindNextAddedX/Y_Direction בעלת בסיס דומה ומימוש שונה לכל כיוון תנועה של הפקמן.
הפעולה מקבלת את ערכי ה X וה- Y הנוכחיים של הפקמן ובמקביל יוצרת שני משתנים פנימיים:
*לכיוונים על ציר X כמו ימין ושמאל המשתנים יהיו כמייצגים את ציר ה X [כפי שנכתב בדוגמה הבאה]

- המשנה **nextX**- מכיל בו את ערך ה X הבא כברירת מחדל [ערך ה X הנוכחי ועוד מספר צעדים קבוע NEXT_POS_ADDED_PIXELS_X לאותו הכיוון].
- המשתנה **saveX-** המייצג בתור התחלה את ערכו של ה X הנוכחי וגדל בעת הבדיקה.

*לכיוונים על ציר Y כמו מעלה ומטה המשתנים יהיו כמייצגים את ציר ה Y

- המשנה **nextY-** מכיל בו את ערך ה Y הבא כברירת מחדל [ערך ה Y הנוכחי ועוד מספר צעדים קבוע NEXT_POS_ADDED_PIXELS_Y לאותו הכיוון].
- המשתנה **saveY-** המייצג בתור התחלה את ערכו של ה Y הנוכחי וגדל בעת הבדיקה



*לכיוונים כמו ימינה ומטה נוסיף משתנה **normalizedX/Y** המייצג את ה X/Y הקיצוניים ביותר לאותו הכיון.
למשל, בדוגמה זו נתייחס למקרה בו הפקמן מתקדם בכיוון הימני.
מיקומו של הפקמן [ערכי X ו- Y] מייצגים את הפינה השמאלית העליונה [הצבועה באדום] בכל תמונה של פקמן.
כדי לבדוק הימצאות גבול בצעד הבא, נדקק לערכי ה X המייצגים את החזית של הפקמן באותו הכיוון. לשם כך, ניצור משתנה **normalizedX** שערכו יהיה לערך הנקודה האדומה + מספר העמודות של הפקמן.
כעת, המשתנה normalizedX ייצג את הפקמן בחזית האופיינית לכיוונו [הנקודה הירוקה].
הדבר נעשה בהתאם גם לכיוון מטה, רק שהערך יהיה של Y ולא של X.
*המשתנה אינו נדרש בכיוונים שמאלה ולמעלה היות וערכי ה X ו- Y מייצגים את החזית של הדמות לאותו הכיוון.
*בהסבר הנוכחי אתייחס לתנועה על ציר X. תנועה על ציר Y מתנהלת בדיוק באותה צורה כך שניתן להחליף בהסבר את ההתייחסות לציר ה X ל Y וההסבר יהיה תקף גם לתנועה בכיוון השני.

כעת, שכל כיוון בעל הערכים הדרושים לו, ניתן להתחיל בבדיקה באמצעות לולאה מקוננת.

1. בלולאה החיצונית- נגדיל בכל איטרציה את מצביע ה X באחד.
2. בלולאה הפנימית- על כל מצביע X נבדוק באמצעות מצביע Y אחר על כל פיקסל בחזית של הפקמן. הבדיקה תתבצע באמצעות פסיקה int 10h/ ah = 0Dh שמחזירה מה צבע הפיקסל במיקום הנוכחי.
   a. במידה והצבע שהוחזר מתאים לצבע של הגבול BLUE_BOUNDARY_COLOR, הפעולה תחזיר את ערך ה X-1 לאותו הכיוון בו הפקמן התנגש בגבול.

b. במידה והצבע המוחזר אינו עונה על התנאי הבדיקה תמשיך להתבצע. מצביעי ה Y ימשיכו להתקדם על החזית של הפקמן וכשיעברו על כולה, יעברו לאיטרציה הבאה, בה ערך ה X גדל באחד לאותו הכיוון והבדיקה מתבצעת.
*אם הפקמן לא נפגש בגבול עד לערך ה X הדיפולטי שהוגדר בהתחלה [nextX], הוא זה שיוחזר.

3. מציאת מינימום הצעדים- במידה ונמצא ערך ה X הבא נעשית בדיקה נוספת כדי לוודא שערך ה X הבא אינו גדול מערך ה X הדיפולטי. יוחזר הערך המינימלי.
4. בדיקת מרחק מהגבול- כדי להימנע ממצב שבתנועה מסוימת הפקמן יתקדם בתנועתו אפילו פיקסל בודד וכך ייצמד לגבול, נעשה חישוב הבודק האם ההפרש בין ערך ה X הדיפולטי הבא לבין ערך ה X הבא שחישבנו באמצעות האלגוריתם קטן מההפרש הקבוע שהפקמן נמצא מהגבול.
במידה וכן, ערך ה X ישתנה כך שישמור את המרחק המבוקש.
במידה ולא, ערך ה X אינו ישתנה.

החזקת הנתונים

רוב הנתונים נשמרים ב Data Segment. ישנם מספר משתנים בתכנית:

- משתנים המתעדכנים בזמן ריצה
  - המשתנים MouseX ו- MouseY [ערכי עכבר]
  - המשתנים pacmanX ו- pacmanY [ערכי פקמן]
- משתנים בוליאניים [העיקריים]
  - המשתנה Bool- מאחסן בו בעיקר מידע מהפעולה isInRange
  - המשתנה Play- מייצג את סיבת היציאה ממסך הבית
  - המשתנה isScoreExists- פלט של פעולת CheckFinish
  - המשתנה isTimeUp- מאוחסן ב Code Segment כמו מספר משתנים שרלוונטיים לטיימר האסינכרוני. המשתנה מתעדכן במהלך ריצת הטיימר במידה והזמן תם.
- משתני גרפיקה
  - המשתנה matrix- מערך המייצג אוסף של פיקסלים המסודרים על פי מידות מסוימות.
  - משתנים עם התחילית Filename- שומרים בהם שם של קבצי bmp.

בנוסף, בתכנית שלי ישנו שימוש נרחב בפורמט ה BMP- מפת סיביות [BitMap].
פורמט זה מאפשר לאחסן מידע על פיקסל של תמונה. כל פיקסל בעל ערך מספרי המייצג תכונות גרפיות [RGB, Transparency].

הצגת הגרפיקה

מרבית הגרפיקה מתבססת על פורמט BMP.
באמצעות פעולה openShowBmp ניתן להציג את התמונות בפורמט זה.
הפעולה מאחדת תחתיה מספר פעולות חשובות המאפשרות את הצגת התמונה:

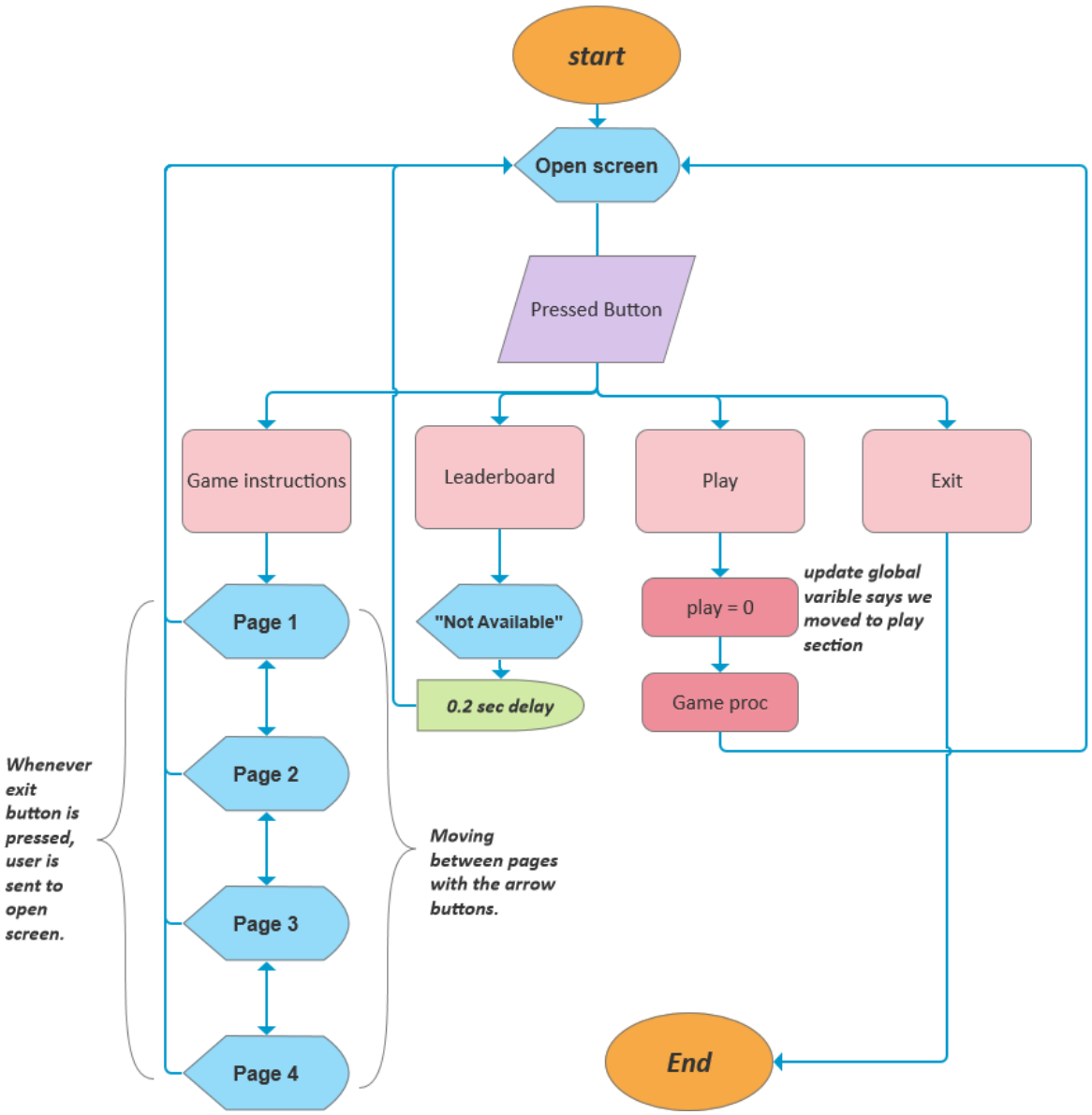
- ❖ הפעולה **OpenBmpFile**- פותחת את הקובץ באמצעות פסיקה ומעבירה את הידית שהתקבלה מהפסיקה למשתנה FileHandle.
- ❖ הפעולה **ReadBmpHeader**- קוראת את הבתים הראשונים של הקובץ [Header]. באמצעות פסיקה שמקבלת בין היתר את כתובת המשתנה בעל שם הקובץ [נמצא ב dx], ההקדמה נקראת.

- הפעולה **ReadBmpPalette**- קוראת את הפלטה של הקובץ ושומרת אותה במשתנה [.Palette]
- הפעולה **CopyBmpPalette-** מעתיקה את הפלטה ששמרנו בסעיף הקודם אל זיכרון המסך באמצעות הפקודה OUT שמעבירה מידע בין ports שניתנים כקלט.
- הפעולה **ShowBMP**- עוברת על החלק השלישי בקובץ [אחרי הפלטה וההקדמה] ומעבירה כל בית מהקובץ אל זיכרון המסך. ההעברה של הבתים נעשית בשורות.
  המידע בקובץ BMP נשמר בסדר הפוך, שיקוף מראה. כדי לסדר זאת מצביע המסך נמצא בשורה האחרונה באזור המסך של הזיכרון והקריאה מהקבוץ נעשית באופן רגיל.
- הפעולה **CloseBmpFile**- הפעולה סוגרת את הקובץ.

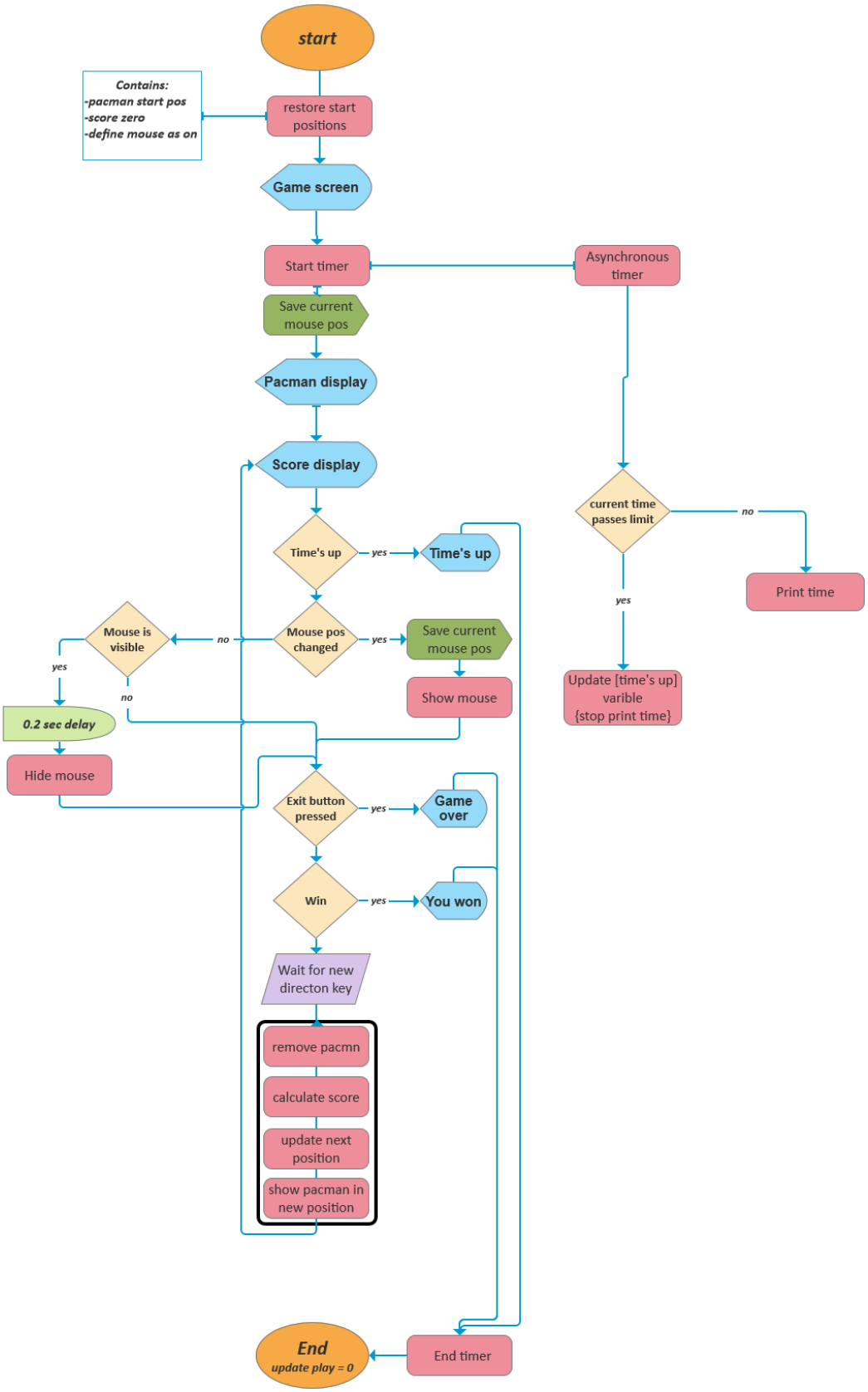
בנוסף, ישנו שימוש בהצגת מטריצות.
באמצעות הפעולה putMatrixInScreen, ניתן להציג מערך שכל תא בו מייצג צבעו של פיקסל במיקומו על המסך. הפעולה מעתיקה בצורה לולאתית כל פעם שורה מהמטריצה אל המסך.

תרשימי זרימה

תכנית ה **main**



תרשים זרימה - המשחק [GameProc]



רשימת פעולות

Main.asm

startgraphicMode

- ❖ טענת כניסה: הפעולה מעבירה למצב graphic mode.
- ❖ טענת יציאה: -
- ❖ מטרה: העברת DosBox למצב טקסט

finishGraphicMode

- ❖ טענת כניסה: הפעולה מעבירה למצב text mode.
- ❖ טענת יציאה: -
- ❖ מטרה: העברת DosBox למצב טקסט.

OpenProc.asm

openScreen

הפעולה מציגה את מסך הפתיחה של התוכנית כולל כל הפעולות שמופעלות עליו.
[אפשר להגיד "תכנית ה main" של מסך הפתיחה]

- ❖ טענת כניסה: הפעולה מציגה את מסך הפתיחה של המשחק ומגיבה לכפתורים בהתאם.
  טענת יציאה: הפעולה תעדכן במשתנים גלובליים מיהו הכפתור שנלחץ ובהתאם תצא מהפעולה.
- ❖ מטרה: להציג את מסך הפתיחה בפרוצדורה אחת.

GameIstructionsPage

- ❖ טענת כניסה: הפעולה מציגה את מסך ההוראות של המשחק.
- ❖ טענת יציאה: הפעולה תגמר בעת לחיצה על הכפתור.
- ❖ מטרה: להציג את מסך ההוראות.

Delay

- ❖ טענת כניסה: הפעולה מחכה0.2 שניות.
- ❖ טענת יציאה: -
- ❖ מטרה: לעכב זמן ריצה של פרוצדורות כדי ליצור רצף איטי יותר מזמן הריצה של המערכת.

ShowMouse

- ❖ טענת כניסה: הפעולה הופכת את העכבר ל visible.
- ❖ טענת יציאה: -
- ❖ מטרה: להציג את העכבר.

HideMouse

- ❖ טענת כניסה: הפעולה הופכת את העכבר ל invisible.
- ❖ טענת יציאה: -
- ❖ מטרה: להחביא את העכבר.

GetMousePos

- ❖ טענת כניסה: הפעולה קוראת לפסיקה שמקבלת את ערכי העכבר הנוכחיים [פוזיציה על המסך] לרגיסטרים.
- ❖ טענת יציאה: -
- ❖ מטרה: לקבל את ערכי העכבר.

ShortBmp

- ❖ טענת כניסה: הפעולה מקצרת את פתיחת קובץ BMP.
  הפעולה מתאימה רק לתמונות בגודל 320*200
  הפעולה תקבל את offset שם הקובץ לרגיסטר dx.
- ❖ טענת יציאה: פתיחת הקובץ על המסך.
- ❖ מטרה: לקצר את הקריאה לפעולות פתיחת ה BMP

isInRange

- ❖ טענת כניסה: הפעולה מקבלת מספר ערכים
  - ◇ ערך X של נקודה נבדקת
  - ◇ ערך Y של נקודה נבדקת
  - ◇ ערך X של טור שמאלי של הכפתור
  - ◇ ערך X של טור ימני של הכפתור
  - ◇ ערך Y של שורה עליונה של הכפתור
  - ◇ ערך Y של שורה תחתונה של הכפתור
- ❖ טענת יציאה: הפעולה תעדכן במשתנה גלובלי [Bool] האם הנקודה הנבדקת נמצאת בטווח הנתון [1 נמצא, 0 לא].
- ❖ מטרה: הפעולה בודקת האם העכבר נמצא בטווח של כפתור המוכנס כקלט.

BMP PROCS

הפרוצדורות הבאות בעלות את אותן טענות כניסה ויציאה.

- ❖ טענת כניסה: הפעולה מציגה את מסך הפתיחה של מסך הבית.
- ❖ טענת יציאה: פתיחת התמונה על המסך.

StratScreen_OPEN

- ❖ מטרה: להציג את מסך הבית של כל המשחק.

PlayButtonDisplay

- ❖ מטרה: להציג את כפתור ה PLAY כצבוע.

LbButtonDisplay

- ❖ מטרה: להציג את כפתור ה LEADER BOARD כצבוע.

InstButtonDisplay

- ❖ מטרה: להציג את כפתור ה INSTRUCTIONS/ GAME MANUAL כצבוע.

NADisplay

- ❖ מטרה: להציג את מסך ה NOT AVAILBLE.

GameProc.asm

Game

הפעולה מציגה את מסך המשחק ואחראית על כל הפעולות שמשחק עושה.
[אפשר להגיד "תכנית ה main" של המשחק]

- ❖ טענת כניסה: הפעולה מציגה את מסך המשחק ומגיבה לכפתורים בהתאם.
  טענת יציאה: הפעולה תכבה את הטיימר ותאפס משתנה גלובלי play המעיד על הכניסה לחלק המשחק בתכנית.
- ❖ מטרה: להציג את המשחק תחת פרוצדורה אחת.

CheckFinish

- ❖ טענת כניסה: הפעולה מקבלת את ערך ה X ו- Y של המיקום הנוכחי של הפקמן
- ❖ טענת יציאה: הפעולה תעדכן במשתנה גלובלי [isScoreExists] האם יש ניצחון [1לא, 0 כן].
- ❖ מטרה: הפעולה בודקת האם בין גבולות המסלול ישנו פיקסל צהוב שאינו הפקמן- במידה וישנו, המשחק לא נוצח, אחרת כן.

restoreGameDis

- ❖ טענת כניסה: הפעולה לא מקבלת ערכים
- ❖ טענת יציאה: -
- ❖ מטרה: לאתחל את ערכי הפקמן והמשחק לפני משחק חוזר [אחזור נקודת התחלת הפקמן וכיוונו, וניקוי keyboard buffer במידה ונעשתה "יציאת חירום" במהלך התכנית הראשית]

Timer

- ❖ טענת כניסה: -
- ❖ טענת יציאה: -
- ❖ מטרה: הפעלת הטיימר.

EndTimer

- ❖ טענת כניסה: הפעולה מסיימת את ריצת הטיימר ומאפשרת את משתניו לקראת הריצה הבאה.
  טענת יציאה: -
- ❖ מטרה: לאפשר שימוש מבוקר בטיימר בעצם הפעלתו וכיבויו.

PrintSecondsElapse

- ❖ טענת כניסה: -
- ❖ טענת יציאה: הדפסת הטיימר על המסך
- ❖ מטרה: ספירת הטיימר והדפסתו. במידה והזמן עובר את ההגבלה, משתנה בוליאני [isTimeUp] מתעדכן.

printAxDec

- ❖ טענת כניסה: ערך ב ax
- ❖ טענת יציאה: הפעולה תדפיס את ax בבסיס דצימלי למסך.

- מטרה: להדפיס את AX בבסיס דצימלי כדי להקל על דיבאגינג ולהציג ערך שבעבור המשתמש הוא הגיוני.

putMatrixInScreen

- טענת כניסה: הפעולה מקבלת מספר ערכים:
  - רגיסטר dx- מספר העמודות
  - רגיסטר dx- מספר השורות
  - מטריצה- הערכים של הפיקסלים הנצבעים.
  - רגיסטר di- כתובת המטריצה.
- טענת יציאה: הצגת המטריצה על המסך
- מטרה: שימוש במטריצה על מנת להציג תמונה.

FindNextAddedX_West

- טענת כניסה: הפעולה מקבלת את ערכי ה X ו- Y של הפקמן.
- טענת יציאה: הפעולה מחזירה את ערך ה X הבא אליו יגיע הפקמן.
- מטרה: להתקדם את מספר הצעדים המקסימלי (בין 0 צעדים לערך ברירת מחדל [NEXT_POS_ADDED_PIXELS_X]) על מנת לא לפגוע בגבולות המשחק.

FindNextAddedX_East

- טענת כניסה: הפעולה מקבלת את ערכי ה X ו- Y של הפקמן.
- טענת יציאה: הפעולה מחזירה את ערך ה X הבא אליו יגיע הפקמן.
- מטרה: להתקדם את מספר הצעדים המקסימלי (בין 0 צעדים לערך ברירת מחדל [NEXT_POS_ADDED_PIXELS_X]) על מנת לא לפגוע בגבולות המשחק.

FindNextAddedY_South

- טענת כניסה: הפעולה מקבלת את ערכי ה X ו- Y של הפקמן.
- טענת יציאה: הפעולה מחזירה את ערך ה Y הבא אליו יגיע הפקמן.
- מטרה: להתקדם את מספר הצעדים המקסימלי (בין 0 צעדים לערך ברירת מחדל [NEXT_POS_ADDED_PIXELS_Y]) על מנת לא לפגוע בגבולות המשחק.

FindNextAddedY_North

- טענת כניסה: הפעולה מקבלת את ערכי ה X ו- Y של הפקמן.
- טענת יציאה: הפעולה מחזירה את ערך ה Y הבא אליו יגיע הפקמן.
- מטרה: להתקדם את מספר הצעדים המקסימלי (בין 0 צעדים לערך ברירת מחדל [NEXT_POS_ADDED_PIXELS_Y]) על מנת לא לפגוע בגבולות המשחק.

AddedScore_West

- טענת כניסה: הפעולה מקבלת את ערכי ה X ו- Y של הפקמן.
- טענת יציאה: הפעולה מעדכנת משתנה גלובלי [score] את הניקוד שהוסף לפקמן בין מקומו הנוכחי להבא.
- מטרה: לחשב את הניקוד שהתווסף בצורה גרפית כדי להימנע ממצב בו הפקמן מדלג על נקודה שאינה נקלטת כהוספת ניקוד.

AddedScore_East

- ❖ טענת כניסה: הפעולה מקבלת את ערכי ה X ו- Y של הפקמן.
- ❖ טענת יציאה: הפעולה מעדכנת משתנה גלובלי [score] את הניקוד שהוסף לפקמן בין מקומו הנוכחי להבא.
- ❖ מטרה: לחשב את הניקוד שהתווסף בצורה גרפית כדי להימנע ממצב בו הפקמן מדלג על נקודה שאינה נקלטת כהוספת ניקוד.

AddedScore_South

- ❖ טענת כניסה: הפעולה מקבלת את ערכי ה X ו- Y של הפקמן.
- ❖ טענת יציאה: הפעולה מעדכנת משתנה גלובלי [score] את הניקוד שהוסף לפקמן בין מקומו הנוכחי להבא.
- ❖ מטרה: לחשב את הניקוד שהתווסף בצורה גרפית כדי להימנע ממצב בו הפקמן מדלג על נקודה שאינה נקלטת כהוספת ניקוד.

AddedScore_North

- ❖ טענת כניסה: הפעולה מקבלת את ערכי ה X ו- Y של הפקמן.
- ❖ טענת יציאה: הפעולה מעדכנת משתנה גלובלי [score] את הניקוד שהוסף לפקמן בין מקומו הנוכחי להבא.
- ❖ מטרה: לחשב את הניקוד שהתווסף בצורה גרפית כדי להימנע ממצב בו הפקמן מדלג על נקודה שאינה נקלטת כהוספת ניקוד.

removePacman

- ❖ טענת כניסה: הפעולה מקבלת את ערכי ה X ו- Y של הפקמן.
- ❖ טענת יציאה: הפעולה תמחק את הפקמן במיקומו הנוכחי [תצבע בשחור].
- ❖ מטרה: למחוק את הפקמן לפני שמבצע צעד הבא.

pacmanFigureDisplay

- ❖ טענת כניסה: הפעולה מקבלת את ערכי ה X ו- Y וכיוונו של הפקמן.
- ❖ טענת יציאה: הפעולה תצייר את הפקמן במיקומו החדש בהתאם לכיוונו.

openShowBmp

- ❖ טענת כניסה: הפעולה מקבלת מספר ערכים:
  - ◇ ערכי ה X ו- Y בהן נרצה להציג את התמונה.
  - ◇ גודל התמונה עמודות ושורות.
  - ◇ אופסט של משתנה המכיל בו את שם הקובץ.
- ❖ טענת יציאה: הפעולה תציג את התמונה על המסך.

ראו פירוט

BMP PROCS

הפרוצדורות הבאות בעלות את אותן טענות כניסה ויציאה.

- ❖ טענת כניסה: הפעולות מציגות מסך בשימוש במהלך המשחק.
- ❖ טענת יציאה: פתיחת התמונה על המסך.

StratScreen_Game

- ❖ מטרה: להציג את מסך המשחק [המבוך, הניקוד וכיו"ב].

TimesUpDisplay

- ❖ מטרה: להציג מסך עם הכתובית TIME'S UP

GameOverDisplay

- ❖ מטרה: להציג את מסך עם הכתובית GAME OVER

WinDisplay

- ❖ מטרה: להציג מסך עם הכתובית YOU WIN

ScoreDisplay

- ❖ מטרה: להציג את הניקוד

פרטים הכרחיים להפעלה

על מנת להפעיל את התוכנית, יש צורך בקבצי הbmp של התוכנית וקבצי ההרצה.

קוד התכנית

Main.asm

```
IDEAL

MODEL small
STACK 256h
jumps
p186

include "OpenEqu.asm"
include "GameEqu.asm"

DATASEG

include "OpenData.asm"
include "GameData.asm"


CODESEG

  ORG 100h
Start:
 mov ax, @data
 mov ds,ax


 call stratGraphicMode

 mov ax,0h ;initilaize mouse
 int 33h

Main:

 call OpenScreen ;open screen display
 cmp [Play], 1 ;check- exit by Play button?
 jne EXIT

 call Game ;game play display
 call hideMouse
```

```
 call Delay
 call Delay
 jmp Main

EXIT:

  call finishGraphicMode
  mov ax, 4C00h ; returns control to dos
  int 21h

include "OpenProc.asm"
include "GameProc.asm"

END Start
```

OpenEqu.asm

```
FILENAME_SCREEN equ 'Screen.bmp'
FILENAME_PLAY equ 'Play.bmp'
FILENAME_LB equ 'lb.bmp'
FILENAME_INST equ 'Inst.bmp'
FILENAME_NA equ 'NA.bmp'
FILENAME_INST_1 equ '1.bmp'
FILENAME_INST_2 equ '2.bmp'
FILENAME_INST_3 equ '3.bmp'
FILENAME_INST_4 equ '4.bmp'

FILE_ROWS = 200
FILE_COLS = 320

;Play Banner values
PLAY_RIGHT_COL = 187
PLAY_LEFT_COL = 137
PLAY_TOP_ROW = 50
PLAY_BOTTOM_ROW = 71

;Leaderboard Banner values
LB_RIGHT_COL = 239
LB_LEFT_COL = 92
LB_TOP_ROW = 78
LB_BOTTOM_ROW = 89

;Game instructions Banner values
INST_RIGHT_COL = 270
INST_LEFT_COL = 51
INST_TOP_ROW = 96
INST_BOTTOM_ROW = 107

;Quit Banner values
QUIT_RIGHT_COL_OPEN = 34
QUIT_LEFT_COL_OPEN = 10
QUIT_TOP_ROW_OPEN = 6
QUIT_BOTTOM_ROW_OPEN = 31

;Arrow forward game manual
ARROW_FORWARD_RIGHT_COL = 195
```

```
ARROW_FORWARD_LEFT_COL = 177
ARROW_FORWARD_TOP_ROW = 183
ARROW_FORWARD_BOTTOM_ROW = 198

;Arrow backward  game manual
ARROW_BACKWARD_RIGHT_COL = 160
ARROW_BACKWARD_LEFT_COL = 144
ARROW_BACKWARD_TOP_ROW = 183
ARROW_BACKWARD_BOTTOM_ROW = 193
```

OpenData.asm

```
Filename_StartScreen db FILENAME_SCREEN, 0
Filename_PlayButton db FILENAME_PLAY, 0
Filename_LbButton db FILENAME_LB, 0
Filename_Inst_button db FILENAME_INST, 0
Filename_NA_Dis db FILENAME_NA, 0
Filename_FirstInst db FILENAME_INST_1, 0
Filename_SecondInst db FILENAME_INST_2, 0
Filename_ThirdInst db FILENAME_INST_3, 0
Filename_FourthInst db FILENAME_INST_4, 0

;Mouse Varibles
MouseX dw ?
MouseY dw ?
isMouseOn dw 1

;Boolean
Bool db 0
isButtonOn db 0

;Buttons on
play db 0
```

OpenProc.asm

```
;=============================================
;Open screen graphics
;=============================================
proc OpenScreen

call Delay
@@MainLoop:
call StratScreen_OPEN

call ShowMouse

CheckStatus: ;maon loop

call GetMousePos

shr cx, 1
mov [MouseX], cx
mov [MouseY], dx

;check if quit button available
push [MouseX]
push [MouseY]
push QUIT_LEFT_COL_OPEN
push QUIT_RIGHT_COL_OPEN
push QUIT_TOP_ROW_OPEN
push QUIT_BOTTOM_ROW_OPEN
call isInRange

cmp [Bool], 1 ;if not, continue
jne PlayBanner

cmp bx, 1 ;check quit pressed
je ExitShourtcut

PlayBanner:
;check if play button available
push [MouseX]
push [MouseY]
push PLAY_LEFT_COL
push PLAY_RIGHT_COL
```

```
push PLAY_TOP_ROW
push PLAY_BOTTOM_ROW
call isInRange

cmp [Bool], 1;if not, continue
jne LeaderBoardBanner

cmp bx, 1;check play pressed, if so jump
je PlayClick

cmp [isButtonOn], 1 ;if button is not pressed but on, keep it so
je CheckStatusShourtcut

call HideMouse ;if button is not pressed and off, turn it on
mov [isButtonOn], 1
call PlayButtonDisplay
call ShowMouse

jmp CheckStatus

PlayClick:
mov [play], 1 ;bool varible explins exit reason

ExitShourtcut:
jmp @@ExitProc


LeaderBoardBanner:
push [MouseX]
push [MouseY]
push LB_LEFT_COL
push LB_RIGHT_COL
push LB_TOP_ROW
push LB_BOTTOM_ROW
call isInRange ;check if leaderboard

cmp [Bool], 1 ;if not, continue
jne GameInstructionsBanner

cmp bx, 1 ;if clicked, displays
je LbClick
```

```
cmp [isButtonOn], 1;if button is not pressed but on, keep it so
je CheckStatusShourtcut

call HideMouse;if button is not pressed and off, turn it on
mov [isButtonOn], 1
call LbButtonDisplay
call ShowMouse

jmp CheckStatus

CheckStatusShourtcut:
jmp CheckStatus

LbClick:
;Display "NOT available" and continue to main loop
mov [play], 0
call HideMouse
call NADisplay
call Delay
call LbButtonDisplay
call ShowMouse
jmp CheckStatusShourtcut

GameInstructionsBanner:
push [MouseX]
push [MouseY]
push INST_LEFT_COL
push INST_RIGHT_COL
push INST_TOP_ROW
push INST_BOTTOM_ROW
call isInRange ;check if game instructions


cmp [Bool], 1
jne @@CleanScreen

cmp bx, 1 ;i was clicked, display
je InstClick

cmp [isButtonOn], 1 ;if button is not pressed but on, keep it so
```

```
je CheckStatusShourtcut

call HideMouse;if button is not pressed and off, turn it on
mov [isButtonOn], 1
call InstButtonDisplay
call ShowMouse
jmp CheckStatus


InstClick:
mov [play], 0
call GameInstructionsPages ;display rules
call InstButtonDisplay ;return to last preview
jmp CheckStatusShourtcut

@@CleanScreen:
;if none of the keys are available restore screen to default
cmp [isButtonOn], 1
jne CheckStatusShourtcut

call HideMouse

mov [isButtonOn], 0
call StratScreen_OPEN

call ShowMouse
jmp CheckStatus

@@ExitProc:
call Delay
ret

endp OpenScreen



;==============================================
;Displays Game manual
;---------------------------------------------
;Output:
;Screen
```

```
;=============================================
count equ [word bp - 2]
proc GameInstructionsPages

 push bp
 mov bp, sp
 sub sp, 2


 mov count, 1
 jmp FirstPage

InstructionsLoop:

 call GetMousePos
 cmp bx, 1 ;if anything pressed -> check what was pressed.
        ;otherwise, wait for input
 jne InstructionsLoop

 shr cx, 1
 mov [MouseX], cx
 mov [MouseY], dx

 push [MouseX]
 push [MouseY]
 push QUIT_LEFT_COL_OPEN
 push QUIT_RIGHT_COL_OPEN
 push QUIT_TOP_ROW_OPEN
 push QUIT_BOTTOM_ROW_OPEN
 call isInRange ;check quit button

 cmp [Bool], 1
 jne NextPage ;if not available, continue
 je @@ExitShourtcut

NextPage:
 push [MouseX]
 push [MouseY]
 push ARROW_FORWARD_LEFT_COL
 push ARROW_FORWARD_RIGHT_COL
 push ARROW_FORWARD_TOP_ROW
```

```
 push ARROW_FORWARD_BOTTOM_ROW
 call isInRange ;check forward arrow

 cmp [Bool], 1
 jne BackPage;if not available, continue

 cmp count, 4 ;top limit -> page cant be bigger than 4
 je InstructionsLoop
 inc count
 jmp CheckCount

@@ExitShourtcut:
 jmp @@ExitProc

BackPage:
 push [MouseX]
 push [MouseY]
 push ARROW_BACKWARD_LEFT_COL
 push ARROW_BACKWARD_RIGHT_COL
 push ARROW_BACKWARD_TOP_ROW
 push ARROW_BACKWARD_BOTTOM_ROW
 call isInRange;check backwards arrow

 cmp [Bool], 1
 jne InstructionsLoop;if not available, continue

 cmp count, 0;top limit -> page cant be smaller than 4
 je InstructionsLoop
 dec count
 jmp CheckCount


CheckCount:
;print page display by counter
 cmp count, 1
 je FirstPage
 cmp count, 2
 je SecondPage
 cmp count, 3
 je ThirdPage
 cmp count, 4
```

```
 je FourthPage
 jne InstructionsLoop

FirstPage:

 call HideMouse
 mov dx, offset Filename_FirstInst
 call ShortBmp
 call ShowMouse
 call Delay
 jmp InstructionsLoop


SecondPage:
 call HideMouse
 mov dx, offset Filename_SecondInst
 call ShortBmp
 call ShowMouse
 call Delay
 jmp InstructionsLoop

ThirdPage:

 call HideMouse
 mov dx, offset Filename_ThirdInst
 call ShortBmp
 call ShowMouse
 call Delay
 call Delay
 jmp InstructionsLoop

FourthPage:

 call HideMouse
 mov dx, offset Filename_FourthInst
 call ShortBmp
 call ShowMouse
 call Delay
 jmp InstructionsLoop
```

```
@@ExitProc:
call Delay

        add sp, 2
        pop bp

        ret

endp GameInstructionsPages

;================
;       Delay - wait 0.2 s
;===============
proc Delay

        pusha
        mov cx, 3h
        mov dx, 0D40h
        mov al, 0
        mov ah, 86h
        int 15h

        popa

        ret
endp Delay
;================
;       Show Mouse
;===============
proc ShowMouse

mov ax, 1h
int 33h

ret

endp ShowMouse
;================
;       Hide Mouse
;===============
proc HideMouse
```

```
mov ax, 2h
int 33h
ret

endp HideMouse
;================
;          GetMousePos
;===============
proc GetMousePos

mov ax, 3h
int 33h
ret

endp GetMousePos
;=============================================
;Open Bmp file
;Proc fits BMP's with the size of 320*200
;---------------------------------------------
;Input:
;filename offset in dx
;=============================================
proc ShortBmp

          mov [BmpLeft],0 ;start point
          mov [BmpTop],0
          mov [BmpColSize], FILE_COLS
          mov [BmpRowSize] ,FILE_ROWS
          call OpenShowBmp
          ret

endp ShortBmp

;=============================================
;Check given point position on buttons
;---------------------------------------------
;Input:
;1- Current X [MouseX -> cx (shr cx, 1)]
;2- Current Y [MouseY -> dx]
;Stack inputs:
```

```
;left column, right column
;top row, bottom row
;-------------------------------------------
;Registers:
; ax, bp
;-------------------------------------------
;Output:
;varible Bool 1 true/ 0 false
;=============================================
;current point checked [mouse values etc]
currentX equ [bp + 14]
currentY equ [bp + 12]
;Button values
leftCol equ [bp + 10]
rightCol equ [bp + 8]
topRow equ [bp + 6]
bottomRow equ [bp + 4]

proc isInRange

 push bp
 mov bp, sp

 push ax

 mov [Bool], 0

@@Rows_Check:

 ;current pos bigger than button edge
 mov ax, currentX
 ;check if currentX checked is in given row range
 cmp ax, rightCol
 ja @@ExitProc
 cmp ax, leftCol
 jb @@ExitProc

@@Col_Check:

 mov ax, currentY
 ;check if currentY checked is in given col range
```

```
 cmp ax, topRow
 jb @@ExitProc
 cmp ax, bottomRow
 ja @@ExitProc

 mov [Bool], 1


@@ExitProc:

 pop ax
 pop bp

 ret 12

endp isInRange

;=======================
;start screen dispaly
;=====================
proc StratScreen_OPEN

        mov dx, offset Filename_StartScreen
  call ShortBmp

        ret

endp StratScreen_OPEN

;==================================
;play button colored screen dispaly
;==================================
proc PlayButtonDisplay

        mov dx, offset Filename_PlayButton
        call ShortBmp

        ret

endp PlayButtonDisplay
```

```
;=========================================
;leaderboard button colored screen dispaly
;=========================================
proc LbButtonDisplay

        mov dx, offset Filename_LbButton
        call ShortBmp

  ret

endp LbButtonDisplay

;==============================================
;Game instructions button colored screen dispaly
;==============================================
proc InstButtonDisplay

        mov dx, offset Filename_Inst_button
        call ShortBmp

 ret

endp InstButtonDisplay
;==============================================
;Game instructions button colored screen dispaly
;==============================================
proc NADisplay

        mov dx, offset Filename_NA_Dis
        call ShortBmp

 ret

endp NADisplay
```

GameEqu.asm

```
FILENAME_GAME_DISPLAY equ 'Game.bmp'
FILENAME_PACMAN_NORTH equ 'PN.bmp'
FILENAME_PACMAN_SOUTH equ 'PS.bmp'
FILENAME_PACMAN_EAST equ 'PE.bmp'
FILENAME_PACMAN_WEST equ 'PW.bmp'
FILENAME_WIN equ 'win.bmp'
FILENAME_LOSE equ 'Lose.bmp'
FILENAME_GAMEOVER equ 'GameOver.bmp'

;Pacman maze
START_POS_X = 86
START_POS_Y = 146
DEFAULT_DIRECTION = 'A'

;Pacman figure
FILE_ROWS_PACMAN = 9
FILE_COLS_PACMAN = 9

;Maze
FILE_ROWS_MAZE = 200
FILE_COLS_MAZE = 320

;Maze colors
BLUE_BOUNDARY_COLOR = 0FCh
YELLOW_DOTS_COLOR_1 = 07Fh
YELLOW_DOTS_COLOR_2 = 0FBh

;Quit Banner values
QUIT_RIGHT_COL_GAME = 313
QUIT_LEFT_COL_GAME = 289
QUIT_TOP_ROW_GAME = 9
QUIT_BOTTOM_ROW_GAME = 33

;Maze values
MAZE_RIGHT_BOUNDARY_X = 167
MAZE_LEFT_BOUNDARY_X = 13
MAZE_RIGHT_EDGE_X = 172
MAZE_LEFT_EDGE_X = 8
```

```
NEXT_POS_ADDED_PIXELS_Y = 7
NEXT_POS_ADDED_PIXELS_X = 7

;Needed when turn:
DISTANCE_FROM_BOUNDARY_X = 2; when moving on Y - distanc1e between pacman and boundary
DISTANCE_FROM_BOUNDARY_Y = 2; when moving on X - distance between  pacman and boundary


;----- Equates Timeer
ticks           EQU     18
BIOSData        EQU     040h
LowTimer        EQU     006Ch
PIC8259                 EQU     0020h
EOI                 EQU 0020h

TIMEOUT = 90
;cursor pos
TIME_ROW = 46
TIME_COL = 20


;Score
SCORE_ADDED_POINTS = 10
;cursor pos
SCORE_ROW = 36
SCORE_COL = 20
```

GameData.asm

```
Filename_Maze db FILENAME_GAME_DISPLAY, 0
Filename_PacmanNorth db FILENAME_PACMAN_NORTH, 0
Filename_PacmanSouth db FILENAME_PACMAN_SOUTH, 0
Filename_PacmanEast db FILENAME_PACMAN_EAST, 0
Filename_PacmanWest db FILENAME_PACMAN_WEST, 0
Filename_Lose_Dis db FILENAME_LOSE, 0
Filename_Win_Screen db FILENAME_WIN, 0
Filename_GameOver_Dis db FILENAME_GAMEOVER, 0
ScrLine db FILE_COLS_MAZE dup (0)  ; One Color line read buffer

FileHandle      dw ?
Header          db 54 dup(0)
Palette         db 400h dup (0)

BmpFileErrorMsg     db 'Error At Opening Bmp File ',FILE_COLS_PACMAN, 0dh,
0ah,'$'
ErrorFile       db 0

BmpLeft dw ?
BmpTop dw ?
BmpColSize dw ?
BmpRowSize dw ?

matrix dw ?

;Current Position
pacmanX dw START_POS_X
pacmanY dw START_POS_Y


pacmanCurrentDirection dw DEFAULT_DIRECTION

;Boolean
isScoreExits db 0
;Timer
exitCode1       db      0
timerSeg        dw      ?
timerOfs        dw      ?

;Score
```

```
score dw 0

pacmanBlank db      0,0,0,0,0,0,0
                                db      0,0,0,0,0,0,0
                                db      0,0,0,0,0,0,0
                                db      0,0,0,0,0,0,0
                                db      0,0,0,0,0,0,0
                                db      0,0,0,0,0,0,0
                                db      0,0,0,0,0,0,0
```

GameProc.asm

```
;============
;          game
;===========
proc Game

  call Delay
  call hideMouse
  call restoreGameDis
  call StratScreen_Game
  call Timer
  ;Show pacman figure according tp direction and Ticurrent Position
  push [pacmanCurrentDirection]
  push [pacmanX]
  push [pacmanY]
  call PacmanFigureDisplay

 call showMouse
 cmp [isMouseOn],1

;update mouse start positiob
 call GetMousePos
 shr cx, 1
 mov [MouseX], cx
 mov [MouseY], dx


MainLoop:

  call ScoreDisplay ;present score on screen

  ;time exit check
  ;if time is up -> exit
  cmp [cs:isTimeUp], 1
  je @@TimesUp


  ;Absorb mouse position
  call GetMousePos
          shr cx, 1
  ;if mouse current pos changed, show mouse
```

```
  ;cmp with previous values
  cmp cx, [MouseX]
  jne ShowMouseWhenMoved
  cmp dx, [MouseY]
  jne ShowMouseWhenMoved
  cmp [isMouseOn],1
  jne MouseContinue

  ;wait a while till turning off mouse
  call Delay
  call hideMouse
  mov [isMouseOn], 0
  jmp MouseContinue

ShowMouseWhenMoved:
  ;if mouse posiition had changed., present it
  call showMouse
          mov [MouseX], cx
          mov [MouseY], dx
  mov [isMouseOn], 1

MouseContinue:
  ;check if mouse is on quit button
  push [MouseX]
  push [MouseY]
          push QUIT_LEFT_COL_GAME
          push QUIT_RIGHT_COL_GAME
          push QUIT_TOP_ROW_GAME
          push QUIT_BOTTOM_ROW_GAME
          call isInRange ;returns value in bool
          cmp [Bool], 1
          jne continue

          cmp bx, 1 ;if mouse was pressed
  jne continue
  jmp @@Gameover

continue:

  ;if maze is empty from dots -> display game over
          push [pacmanY]
```

```
            push [pacmanX]
            call CheckFinish
            cmp [isScoreExits],0
            je @@Win

  ;check if keyboard key available
            mov ah, 1
            int 16h
            jz MainLoop ;not available -> check mouse new pos

   ;get key value present new direction
            mov ah, 0
            int 16h

            cmp al, 'W'
            je North
            cmp al, 'w'
            je North

            cmp al, 'S'
            je South
            cmp al, 's'
            je South

            cmp al, 'D'
            je East
            cmp al, 'd'
            je East

            cmp al, 'A'
            je West
            cmp al, 'a'
            je West

            jne MainLoop
North:

;proc color pacman in black
 push [pacmanX]
 push [pacmanY]
 call removePacman
```

```
 mov [pacmanCurrentDirection], 'W' ;update direction

;calculate score according to next eaten spot
 push [pacmanY]
 push [pacmanX]
 call AddScore_North

;find next pos
 push [pacmanY]
 push [pacmanX]
 call FindNextAddedY_North
 pop [pacmanY] ;new value tranformed into varible

;display figure in next position
 push [pacmanCurrentDirection]
 push [pacmanX]
 push [pacmanY]
 call PacmanFigureDisplay

 jmp MainLoop


South:
;proc color pacman in black
 push [pacmanX]
 push [pacmanY]
 call removePacman

 mov [pacmanCurrentDirection], 'S';update direction

 ;calculate score according to next eaten spot
 push [pacmanY]
 push [pacmanX]
 call AddScore_South

 ;find next pos
 push [pacmanY]
 push [pacmanX]
 call FindNextAddedY_South
 pop [pacmanY] ;next value tranformed into varible
```

```
;present figure in new position
 push [pacmanCurrentDirection]
 push [pacmanX]
 push [pacmanY]
 call PacmanFigureDisplay

 jmp MainLoop

East:
;proc color pacman in black
 push [pacmanX]
 push [pacmanY]
 call removePacman

 mov [pacmanCurrentDirection], 'D' ;update direction

 ;calculate score according to next eaten spot
 push [pacmanX]
 push [pacmanY]
 call AddScore_East

 ;find next pos
 push [pacmanX]
 push [pacmanY]
 call FindNextAddedX_East
 pop [pacmanX] ;next value tranformed into vvatible

;present figure in new position
 push [pacmanCurrentDirection]
 push [pacmanX]
 push [pacmanY]
 call PacmanFigureDisplay

 jmp MainLoop

West:

;proc color pacman in black
 push [pacmanX]
 push [pacmanY]
```

```
 call removePacman

 mov [pacmanCurrentDirection], 'A' ;update direction

 ;calculate score according to next eaten spot
 push [pacmanX]
 push [pacmanY]
 call AddScore_West

 ;find next pos
 push [pacmanX]
 push [pacmanY]
 call FindNextAddedX_West
 pop [pacmanX] ;next value tranformed into varible

;display figure in new position
 push [pacmanCurrentDirection]
 push [pacmanX]
 push [pacmanY]
 call PacmanFigureDisplay

 jmp MainLoop

;if time is up- present screen and exit
@@TimesUp:
  call Delay
  call Delay
  call Delay
  call Delay
  call hideMouse
  call EndTimer
          call TimesUpDisplay
  call ShowMouse
  call Delay
  jmp @@ExitProc

;if win- present screen and exit
@@Win:
  call EndTimer
  call Delay
  call Delay
```

```
  call Delay
  call Delay
  call hideMouse
          call WinDisplay
  call ShowMouse
  call Delay
  jmp @@ExitProc

;if exit button pressed- present screen and exit
@@Gameover:
  call EndTimer
  call Delay
  call hideMouse
          call GameoverDisplay
  call ShowMouse
  call Delay
  ;jmp @@ExitProc

@@ExitProc:
  call showMouse
  mov [play],0 ;update -> no longer in game
  call EndTimer
          ret

endp Game
;=============================================
;Check if win or loose
;---------------------------------------------
;Input:
;1- Pacman's current X
;2- Pacman's current Y
;Stack inputs:
;---------------------------------------------
;Registers:
; ax, bx, cx, dx, si, di, bp
;---------------------------------------------
;Output:
;varible isScoreExits 1 true [loose]/ 0 false
;=============================================
curX equ [word bp + 4] ;left col
curY equ [word bp + 6] ;top row
```

```
rightCol equ [word bp - 2]
bottomRow equ [word bp - 4]

proc CheckFinish

  push bp
  mov bp, sp
  sub sp, 4

  mov [isScoreExits],0 ;varible present wether score left

;pacman edges values:
mov ax, curX
mov rightCol, FILE_COLS_PACMAN
add rightCol, ax

mov ax, curY
mov bottomRow, FILE_ROWS_PACMAN
add bottomRow, ax

mov di, MAZE_RIGHT_EDGE_X - MAZE_LEFT_EDGE_X ;cols counter
mov si, 200 ;max Y value
mov dx, 0 ;min Y value


Rows:
  mov cx, MAZE_LEFT_EDGE_X ;min X value
        mov di, MAZE_RIGHT_BOUNDARY_X ;max X value
Cols:
        ; ◄■■ Get pixel color of X&Y
        mov ah,0Dh
        int 10H ; AL = COLOR
        cmp al, YELLOW_DOTS_COLOR_1
        je writeYellow
        cmp al, YELLOW_DOTS_COLOR_2
        je writeYellow

cont:

        inc cx ;x pointer
        dec di ;loop counter
```

```
        cmp di, 0;cols check finished
        jne Cols

        inc dx ;increase y counter
        dec si ;decrese row loop counter
        cmp si, 0; row check finished
        jne Rows

        jmp done

writeYellow:
   ;find if current check refers to pacman -if so skip
   ;is the yellow in pacman's range
   cmp cx, curX
   jb yesYellow
   cmp cx, rightCol
   ja yesYellow
   cmp dx, curY
   jb yesYellow
   cmp dx, bottomRow
   ja yesYellow

SkipPacman:
                jmp cont

yesYellow:
   mov [isScoreExits], 1 ;score appears in maze- no win
   jmp done

done:

   add sp, 4
   pop bp
   ret 4
endp CheckFinish

;=====================
;start screen dispaly
;=====================
proc StratScreen_Game
```

```
        mov dx, offset Filename_Maze
  call ShortBmp
        ret

endp StratScreen_Game

;=====================
;time's up screen dispaly
;====================
proc TimesUpDisplay

        mov dx, offset Filename_Lose_Dis
  call ShortBmp
        ret

endp TimesUpDisplay
;=====================
;game over screen dispaly
;====================
proc GameoverDisplay

        mov dx, offset Filename_GameOver_Dis
  call ShortBmp
        ret

endp GameoverDisplay
;=====================
;win screen dispaly
;====================
proc WinDisplay

        mov dx, offset Filename_Win_Screen
  call ShortBmp
        ret

endp WinDisplay
;=========================================
;restore pacman values before restarting game
;=========================================
proc restoreGameDis
```

```
  mov [pacmanX], START_POS_X
  mov [pacmanY], START_POS_Y
  mov [pacmanCurrentDirection], DEFAULT_DIRECTION
  mov [score], 0
  mov [isMouseOn], 1

  ;clear keyboard buffer
  mov ah,0ch
  mov al,0
  int 21h

  ret
endp restoreGameDis

;=====================
;Timer initalizing
;====================
proc Timer
            mov    ax,@data
   mov   es,ax
   mov   [word cs:difference],ticks

            ; save the current interrupt verctor.
            ; the timer interrupt number is 1C
            push   es
            mov    ax, 351Ch
            int    21h
            mov    [timerSeg],es
            mov    [timerOfs],bx
            pop    es

            ;set the new inerrupt vector with our function
            push   ds
            mov    ax,251Ch
            push   cs
            pop    ds
            mov    dx, offset PrintSecondsElapse
            int    21h
            pop    ds
            ret
```

```
endp Timer
;============
;          End Timer
;===========
proc EndTimer

;restore time values
mov [cs: inProgress], 0
mov [cs:difference], 0
mov [cs:lastTimer], 0
mov [cs:fixDrift], 5
mov [cs:counter], 0
mov [cs:isTimeUp], 0

;restore interrupt
push    ds
mov     ax,251Ch
mov     dx,[timerOfs]
mov     ds,[timerSeg]
int     21h
pop     ds

ret

endp EndTimer
;======================
;Score Display
;=====================
proc ScoreDisplay

            push SCORE_ROW ;cursor position
            push SCORE_COL
            mov ax, [score]
            call printAxDec
            ret

endp ScoreDisplay
;=============================================
;Find next X value considering bounderies.
;-------------------------------------------
```

```
;Input:
;1- CurrentXPos
;2- CurrentYPos
;-------------------------------------------
;Registers:
;bp, cx, dx, ax
;-------------------------------------------
;Output:
;nextX value - stack
;=============================================
currentX equ [word bp + 6]
currentY equ [word bp + 4]
nextX equ [word bp - 8]
saveX equ [word bp - 10]

proc FindNextAddedX_West

push bp
mov bp, sp

push ax
push dx
push cx

sub sp, 4

;nextX is the defualt value of currentX in the next step
mov ax, currentX
mov nextX, ax
sub nextX, NEXT_POS_ADDED_PIXELS_X

;saveX value which changes according to boundary check
mov cx, currentX
mov saveX, cx

@@IsTouchingBoundray:
  ;==============================================
  ; loop checks when pacman meets boundary by:
  ;1. one added pixel forward
  ;2. pacman pront col
  ;when color is equal to boundary -> return smallest X
```

```
 ;=============================================

 mov dx, currentY
 mov cx, FILE_COLS_PACMAN ;check of front col

 @@CheckByYs:
 push cx ;save counter value

 ;dx value is initilaized before inner loop
 mov cx, saveX
 mov ah,0Dh
        int 10h ;absorb color
        cmp al, BLUE_BOUNDARY_COLOR
        je @@PopStack

 pop cx
 inc dx ;raise col pointer

 loop @@CheckByYs

 ;stop counting if next value smaller than next defualt value
 mov cx, saveX
        cmp cx, nextX
        jbe @@FindMinSteps

        dec saveX ;decrease nextX counter
        jmp @@IsTouchingBoundray

@@PopStack:
 pop cx

@@FindMinSteps:
 ;find the minimum steps forward
 mov cx, saveX
        inc cx
        cmp cx, nextX
        jb @@CheckAddedSteps

@@CountedSteps:

mov nextX, cx ;cx value is next X value
```

```
@@CheckAddedSteps:

;calc the distance from boundary
mov cx, currentX
sub cx, nextX

;Check if the difference between current pos to next sticks pacman to boundary
;if so, move the pacman to the proper distance from boundary.
cmp cx, DISTANCE_FROM_BOUNDARY_X
jnae @@ExitProc

add nextX, DISTANCE_FROM_BOUNDARY_X

@@CheckEdge:
  ;lanuch pacman to the other side
        cmp nextX, MAZE_LEFT_EDGE_X
        jnbe @@ExitProc

        mov nextX, MAZE_RIGHT_EDGE_X - FILE_COLS_PACMAN

@@ExitProc:

;save nextX in top Stack cell
mov cx, nextX
mov currentX, cx

add sp, 4

pop cx
pop dx
pop ax

pop bp

ret 2

endp FindNextAddedX_West
```

```
;=============================================
;Find next X value considering bounderies.
;---------------------------------------------
;Input:
;1- CurrentXPos
;2- CurrentYPos
;---------------------------------------------
;Registers:
;bp, cx, dx, ax
;---------------------------------------------
;Output:
;nextX value - stack
;=============================================
currentX equ [word bp + 6]
currentY equ [word bp + 4]
nextX equ [word bp - 8]
saveX equ [word bp - 10]
normalizedX equ [word bp - 12]

proc FindNextAddedX_East

push bp
mov bp, sp

push ax
push dx
push cx

sub sp, 6

;Containing next defualt value
;CurrentX is top left pacman point -> dosen't present the east muserments properly
;normalizedX is the top left pacman X value
mov ax, currentX
mov normalizedX, ax
add normalizedX, FILE_COLS_PACMAN

;nextX is the defualt value of currentX in the next step
mov ax, normalizedX
mov nextX, ax
add nextX, NEXT_POS_ADDED_PIXELS_X
```

```
;saveX value which changes according to boundary check
mov cx, normalizedX
mov saveX, cx

@@IsTouchingBoundray:
;==============================================
; loop checks when pacman meets boundary by:
;1. one added pixel forward
;2. pacman pront col
;when color is equal to boundary -> return smallest X
;=============================================

  mov dx, currentY
  mov cx, FILE_COLS_PACMAN;check of front col

  @@CheckByYs:
  push cx ;save counter value

  ;dx value is initilaized before inner loop
  mov cx, saveX
  mov ah,0Dh
          int 10h ;absorb color
  cmp al, BLUE_BOUNDARY_COLOR
  je @@PopStack

  pop cx
  inc dx ;decrease col pointer

  loop @@CheckByYs

  ;stop counting if next value bigger than next defualt value
  mov cx, saveX
          cmp cx, nextX
          jae @@FindMinSteps

          inc saveX;decrease nextX counter
          jmp @@IsTouchingBoundray

@@PopStack:
  pop cx
```

```
@@FindMinSteps:
;find the minimum steps forward
  mov cx, saveX ;restore x value
        dec cx
        cmp cx, nextX
        ja @@CheckAddedSteps

@@CountedSteps:

        mov nextX, cx ;cx value is next X value

@@CheckAddedSteps:
;calc the distance from boundary
mov cx, nextX
sub cx, normalizedX

;Check if the difference between current pos to next sticks pacman to boundary
;if so, move the pacman to the proper distance from boundary.
cmp cx, DISTANCE_FROM_BOUNDARY_X
jnae @@ExitProc

sub nextX, DISTANCE_FROM_BOUNDARY_X

@@CheckEdge:
;lanuch pacman to the other side
        cmp nextX, MAZE_RIGHT_EDGE_X
        jnae @@ExitProc

        mov nextX, MAZE_LEFT_EDGE_X + FILE_COLS_PACMAN

@@ExitProc:
;restore normalized value
sub nextX, FILE_COLS_PACMAN
;save nextX in top Stack cell
mov cx, nextX
mov currentX, cx

add sp, 6

pop cx
pop dx
```

```
pop ax
pop bp

ret 2

endp FindNextAddedX_East

;=============================================
;Find next Y value considering bounderies.
;---------------------------------------------
;Input:
;1- CurrentYPos
;2- CurrentXPos
;---------------------------------------------
;Registers:
;bp, cx, dx, ax
;---------------------------------------------
;Output:
;nextX value - stack
;=============================================
currentY equ [word bp + 6]
currentX equ [word bp + 4]
nextY equ [word bp - 8]
saveX equ [word bp - 10]
normalizedY equ [word bp - 12]

proc FindNextAddedY_South

push bp
mov bp, sp

push ax
push dx
push cx

sub sp, 6

;presents bottom row of pacman [instead of top]
mov ax, currentY
mov normalizedY, ax
add normalizedY, FILE_ROWS_PACMAN
```

```
;defualt next Y value
mov ax, normalizedY
mov nextY, ax
add nextY, NEXT_POS_ADDED_PIXELS_Y


;initalizing dx value
mov dx, normalizedY

@@IsTouchingBoundray:
;=============================================
; loop checks when pacman meets boundary by:
;1. one added pixel forward
;2. pacman pront row
;when color is equal to boundary -> return smallest Y
;=============================================

;default checked X value
mov cx, currentX
mov saveX, cx

mov cx, FILE_COLS_PACMAN;check of front col
@@CheckByXs:
push cx ;save counter value

;dx y value already saved
mov cx, saveX
mov ah,0Dh
int 10h;absorb color
cmp al, BLUE_BOUNDARY_COLOR
je @@PopStack

inc saveX  ;raise row pointer
pop cx

loop @@CheckByXs

;stop counting if next value bigger than next defualt value
cmp dx, nextY
jae @@FindMinSteps
```

```
inc dx
jmp @@IsTouchingBoundray

@@PopStack:
pop cx

@@FindMinSteps:
  ;find minimum steps
                    dec dx
                    cmp dx, nextY
                    ja @@CheckAddedSteps

@@CountedSteps:

                    mov nextY, dx ;dx value is next Y value

@@CheckAddedSteps:
;Check if the difference between current pos to next sticks pacman to boundary
;if so, move the pacman to the proper distance from boundary.
          mov dx, nextY
          sub dx, normalizedY

          cmp dx, DISTANCE_FROM_BOUNDARY_Y
          jnae @@ExitProc

          sub nextY, DISTANCE_FROM_BOUNDARY_Y

@@ExitProc:
  ;restore not normalized pacman values
          sub nextY, FILE_COLS_PACMAN
  ;save next Y value in top stack cell
          mov dx, nextY
          mov currentY, dx

add sp, 6

pop cx
pop dx
pop ax
pop bp
```

```
ret 2

endp FindNextAddedY_South
;=============================================
;Find next Y value considering bounderies.
;---------------------------------------------
;Input:
;1- CurrentYPos
;2- CurrentXPos
;---------------------------------------------
;Registers:
;bp, cx, dx, ax
;---------------------------------------------
;Output:
;nextX value - stack
;=============================================
currentY equ [word bp + 6]
currentX equ [word bp + 4]
nextY equ [word bp - 8]
saveX equ [word bp - 10]

proc FindNextAddedY_North

  push bp
  mov bp, sp

  push ax
  push dx
  push cx

  sub sp, 4

  ;defualt next Y value
  mov ax, currentY
  mov nextY, ax
  sub nextY, NEXT_POS_ADDED_PIXELS_Y

  ;initalizing dx value
  mov dx, currentY
```

```
@@IsTouchingBoundray:
;==============================================
; loop checks when pacman meets boundary by:
;1. one added pixel forward
;2. pacman pront row
;when color is equal to boundary -> return smallest Y
;=============================================
  ;default checked X value
  mov cx, currentX
  mov saveX, cx

  mov cx, FILE_COLS_PACMAN ;check of front col
  @@CheckByXs:
  push cx ;save counter value

  ;dx y value already saved
  mov cx, saveX
  mov ah,0Dh
  int 10h ;absorb color
  cmp al, BLUE_BOUNDARY_COLOR
  je @@PopStack

  inc saveX ;raise row pointer
  pop cx

  loop @@CheckByXs

  ;stop counting if next value smaller than next defualt value
  cmp dx, nextY
  jbe @@FindMinSteps

  dec dx ;decrease Y value
  jmp @@IsTouchingBoundray

@@PopStack:
  pop cx

@@FindMinSteps:
;find minimum steps
                    inc dx
                    cmp dx, nextY
```

```
                jb @@CheckAddedSteps

@@CountedSteps:

                mov nextY, dx ;dx value is next Y value

@@CheckAddedSteps:
;Check if the difference between current pos to next sticks pacman to boundary
;if so, move the pacman to the proper distance from boundary.
        mov dx, currentY
        sub dx, nextY

        cmp dx, DISTANCE_FROM_BOUNDARY_Y
        jnae @@ExitProc

        add nextY, DISTANCE_FROM_BOUNDARY_Y

@@ExitProc:

        mov dx, nextY
        mov currentY, dx
 ;save next Y value in top stack cell

 add sp, 4

 pop cx
 pop dx
 pop ax
 pop bp

 ret 2

endp FindNextAddedY_North

;=============================================
;Changes score.
;---------------------------------------------
;Input:
;1- CurrentXPos
;2- CurrentYPos
;---------------------------------------------
```

```
;Registers:
;bp, cx, dx, ax
;-------------------------------------------
;Output:
;score - global varible
;=============================================
currentX equ [word bp + 6]
currentY equ [word bp + 4]
nextX equ [word bp - 8]
saveX equ [word bp - 10]

proc AddScore_West

        push bp
        mov bp, sp

        push ax
        push dx
        push cx

        sub sp, 4

        push currentX
        push currentY
        call FindNextAddedX_West
        pop nextX

  mov cx, currentX
  mov saveX, cx

@@FindNextDot:

  mov dx, currentY
  mov cx, FILE_COLS_PACMAN ;check of front col

  @@CheckByYs:
  push cx ;save counter value

  ;dx value is initilaized before inner loop
  mov cx, saveX
  mov ah,0Dh
```

```
 int 10h ;absorb color
 ;each meeting point increases points
        cmp al, YELLOW_DOTS_COLOR_1
 je @@PopStack
        cmp al, YELLOW_DOTS_COLOR_2
 je @@PopStack

 pop cx
 inc dx ;raise col pointer

 loop @@CheckByYs

 ;stop counting if next value smaller than next defualt value
 mov cx, saveX
 cmp cx, nextX
 jbe @@ExitProc

 dec saveX ;decrease nextX counter
 jmp @@FindNextDot

@@PopStack:
 pop cx
@@IncScore:
 ;inc score by defualt val
        add [score], SCORE_ADDED_POINTS

@@ExitProc:

        add sp, 4

        pop cx
        pop dx
        pop ax

        pop bp

        ret 4

endp AddScore_West
;=============================================
;Changes score.
```

```
;-------------------------------------------
;Input:
;1- CurrentXPos
;2- CurrentYPos
;-------------------------------------------
;Registers:
;bp, cx, dx, ax
;-------------------------------------------
;Output:
;score - global varible
;===============================================
currentX equ [word bp + 6]
currentY equ [word bp + 4]
nextX equ [word bp - 8]
saveX equ [word bp - 10]
normalizedX equ [word bp - 12]

proc AddScore_East

  push bp
  mov bp, sp

  push ax
  push dx
  push cx

  sub sp, 6

  ;Containing next defualt value
  ;CurrentX is top left pacman point -> dosen't present the east muserments properly
  mov ax, currentX
  mov normalizedX, ax
  add normalizedX, FILE_COLS_PACMAN

  push currentX
  push currentY
  call FindNextAddedX_East
  pop nextX
  add nextX, FILE_COLS_PACMAN

  mov cx, normalizedX
```

```
mov saveX, cx

@@FindNextDot:

mov dx, currentY
mov cx, FILE_COLS_PACMAN ;check of front col

@@CheckByYs:
push cx ;save counter value

;dx value is initilaized before inner loop
mov cx, saveX
mov ah,0Dh
int 10h ;absorb color
;each meeting point increases points
cmp al, YELLOW_DOTS_COLOR_1
je @@PopStack
cmp al, YELLOW_DOTS_COLOR_2
je @@PopStack

pop cx
inc dx ;raise col pointer

loop @@CheckByYs

;stop counting if next value smaller than next defualt value
mov cx, saveX
cmp cx, nextX
jae @@ExitProc

inc saveX ;increase nextX counter
jmp @@FindNextDot

@@PopStack:
pop cx

@@IncScore:

add [score], SCORE_ADDED_POINTS

@@ExitProc:
```

```
  add sp, 6

  pop cx
  pop dx
  pop ax
  pop bp

  ret 4

endp AddScore_East
;=============================================
;Changes score.
;---------------------------------------------
;Input:
;1- CurrentXPos
;2- CurrentYPos
;---------------------------------------------
;Registers:
;bp, cx, dx, ax
;---------------------------------------------
;Output:
;score - global varible
;=============================================
currentY equ [word bp + 6]
currentX equ [word bp + 4]
nextY equ [word bp - 8]
saveX equ [word bp - 10]
normalizedY equ [word bp - 12]

proc AddScore_South

        push bp
        mov bp, sp

        push ax
        push dx
        push cx

        sub sp, 6
```

```
        mov ax, currentY
        mov normalizedY, ax
        add normalizedY, FILE_ROWS_PACMAN
;for calculation of dots betweeen cur pos to next
        push currentY
        push currentX
        call FindNextAddedY_South
        pop nextY
        add nextY, FILE_ROWS_PACMAN

mov dx, normalizedY
@@FindNextDot:

;default checked X value
mov cx, currentX
mov saveX, cx

mov cx, FILE_COLS_PACMAN;check of front col
@@CheckByXs:
push cx ;save counter value

;dx y value already saved
mov cx, saveX
mov ah,0Dh
int 10h ;absorb color
;each meeting point increases points
        cmp al, YELLOW_DOTS_COLOR_1
je @@PopStack
        cmp al, YELLOW_DOTS_COLOR_2
je @@PopStack

inc saveX  ;raise row pointer
pop cx

loop @@CheckByXs

        cmp dx, nextY
        jae @@ExitProc

        inc dx
        jmp @@FindNextDot
```

```
@@PopStack:
  pop cx

@@IncScore:
        add [score], SCORE_ADDED_POINTS

@@ExitProc:

        add sp, 6

        pop cx
        pop dx
        pop ax
        pop bp

        ret 4

endp AddScore_South
;=============================================
;Changes score.
;---------------------------------------------
;Input:
;1- CurrentXPos
;2- CurrentYPos
;---------------------------------------------
;Registers:
;bp, cx, dx, ax
;---------------------------------------------
;Output:
;score - global varible
;=============================================
currentY equ [word bp + 6]
currentX equ [word bp + 4]
nextY equ [word bp - 8]
saveX equ [word bp - 10]

proc AddScore_North

        push bp
        mov bp, sp
```

```
        push ax
        push dx
        push cx

        sub sp, 4

 ;for calculation of dots betweeen cur pos to next
        push currentY
        push currentX
        call FindNextAddedY_North
        pop nextY

 ;initalizing dx value
 mov dx, currentY

@@FindNextDot:
 ;default checked X value
 mov cx, currentX
 mov saveX, cx

 mov cx, FILE_COLS_PACMAN ;check of front col
 @@CheckByXs:
 push cx ;save counter value

 ;dx y value already saved
 mov cx, saveX
 mov ah,0Dh
 int 10h ;absorb color
 ;each meeting point increases points
        cmp al, YELLOW_DOTS_COLOR_1
 je @@PopStack
        cmp al, YELLOW_DOTS_COLOR_2
 je @@PopStack

 inc saveX ;raise row pointer
 pop cx

 loop @@CheckByXs

        cmp dx, nextY
```

```
        jbe @@ExitProc

        dec dx ;continue loop
        jmp @@FindNextDot

@@PopStack:
  pop cx
@@IncScore:

        add [score], SCORE_ADDED_POINTS

@@ExitProc:

        add sp, 4

        pop cx
        pop dx
        pop ax
        pop bp

        ret 4

endp AddScore_North
;=============================================
;Remove pacman and dots (9*9)
;---------------------------------------------
;Input:
;1- CurrentXPos
;2- CurrentYPos
;---------------------------------------------
;Registers:
;bp, cx, dx, di, ax
;---------------------------------------------
;Output:
;screen
;=============================================
currentX equ [word bp + 6]
currentY equ [word bp + 4]

proc removePacman
```

```
        push bp
        mov bp, sp

        push cx ; save cx
  lea cx, [pacmanBlank] ;save matrix in varible
  mov [matrix] , cx

        push dx; save dx
  mov dx, FILE_COLS_PACMAN ;define matrix size
  mov cx, FILE_ROWS_PACMAN

        push di
        push ax

  ;calculate coloring position
        mov di, currentY
        mov ax, currentY
        ;currentY * 320
        shl di, 8
        shl ax, 6
        add di, ax
        add di, currentX

 call putMatrixInScreen

        pop ax
        pop di
        pop dx
        pop cx
        pop bp

        ret 4

endp removePacman

;=============================================
;Shows pacman on screen
;---------------------------------------------
;Input:
;1- Direction (by big letter [North -> N])
;2- CurrentXPos
```

```
;3- CurrentYPos
;-------------------------------------------
;Registers:
; dx, bp
;-------------------------------------------
;Output:
;screen
;==============================================

        dirction equ [word bp + 8]
        xPos equ [word bp + 6]
        yPos equ [word bp + 4]

proc PacmanFigureDisplay

        push bp
        mov bp, sp

        push dx
        push ax

;print pacman according to dirction and X,Y pos

@@North:
 mov ax, dirction
        cmp ax, 'W'
        jne @@South

        mov dx, offset Filename_PacmanNorth
        jmp PacmanDisplay

@@South:

 mov ax, dirction
        cmp ax, 'S'
        jne @@East

        mov dx, offset Filename_PacmanSouth
        jmp PacmanDisplay

@@East:
```

```
 mov ax, dirction
        cmp ax, 'D'
        jne @@West

        mov dx, offset Filename_PacmanEast
        jmp PacmanDisplay

@@West:

        mov dx, offset Filename_PacmanWest


PacmanDisplay:

        mov ax, xPos
        mov [BmpLeft],ax
        mov ax, yPos
        mov [BmpTop],ax
        mov [BmpColSize], FILE_COLS_PACMAN
        mov [BmpRowSize] ,FILE_ROWS_PACMAN
        call OpenShowBmp

        pop ax
        pop dx
        pop bp

        ret 6

endp PacmanFigureDisplay


;==============================================================================
==================
;=======================
;Put bmp file on screen
;======================
proc OpenShowBmp near


call OpenBmpFile
```

```
cmp [ErrorFile],1
je @@ExitProc

call ReadBmpHeader

call ReadBmpPalette

call CopyBmpPalette

call  ShowBMP


call CloseBmpFile

@@ExitProc:
ret
endp OpenShowBmp



; input dx filename to open
proc OpenBmpFile      near
mov ah, 3Dh
xor al, al
int 21h
jc @@ErrorAtOpen
mov [FileHandle], ax
jmp @@ExitProc

@@ErrorAtOpen:
mov [ErrorFile],1
@@ExitProc:
ret
endp OpenBmpFile


proc CloseBmpFile near
mov ah,3Eh
mov bx, [FileHandle]
int 21h
```

```
ret
endp CloseBmpFile


; Read 54 bytes the Header
proc ReadBmpHeadernear
push cx
push dx

mov ah,3fh
mov bx, [FileHandle]
mov cx,54
mov dx,offset Header
int 21h

pop dx
pop cx
ret
endp ReadBmpHeader


proc ReadBmpPalette near ; Read BMP file color palette, 256 colors * 4 bytes (400h)
                                                ; 4 bytes for each color BGR + null)
push cx
push dx

mov ah,3fh
mov cx,400h
mov dx,offset Palette
int 21h

pop dx
pop cx

ret
endp ReadBmpPalette
;=============================================
; Will move out to screen memory the colors
; video ports are 3C8h for number of first color
; and 3C9h for all rest
;=============================================
```

```
proc CopyBmpPalette            near

push cx
push dx

mov si,offset Palette
mov cx,256
mov dx,3C8h
mov al,0  ; black first
out dx,al ;3C8h
inc dx    ;3C9h
CopyNextColor:
mov al,[si+2]              ; Red
shr al,2                           ; divide by 4 Max (cos max is 63 and we have here max 255 )
(loosing color resolution).
out dx,al
mov al,[si+1]              ; Green.
shr al,2
out dx,al
mov al,[si]                ; Blue.
shr al,2
out dx,al
add si,4                           ; Point to next color.  (4 bytes for each color BGR + null)

loop CopyNextColor

pop dx
pop cx

ret
endp CopyBmpPalette

;=================================================
; BMP graphics are saved upside-down.
; Read the graphic line by line (BmpRowSize lines in VGA format),
; PacmanDispalying the lines from bottom to top.
;=================================================
proc ShowBMP

push cx
```

```
mov ax, 0A000h
mov es, ax

mov cx,[BmpRowSize]


mov ax,[BmpColSize] ; row size must dived by 4 so if it less we must calculate the extra
padding bytes
xor dx,dx
mov si,4
div si
cmp dx,0
mov bp,0
jz @@row_ok
mov bp,4
sub bp,dx

@@row_ok:
mov dx,[BmpLeft]

@@NextLine:
push cx
push dx

mov di,cx  ; Current Row at the small bmp (each time -1)
add di,[BmpTop] ; add the Y on entire screen


; next 5 lines  di will be  = cx*320 + dx , point to the correct screen line
dec di
mov cx,di
shl cx,6
shl di,8
add di,cx
add di,dx

; small Read one line
mov ah,3fh
mov cx,[BmpColSize]
add cx,bp  ; extra  bytes to each row must be divided by 4
mov dx,offset ScrLine
```

```
int 21h
; Copy one line into video memory
cld ; Clear direction flag, for movsb
mov cx,[BmpColSize]
mov si,offset ScrLine
rep movsb ; Copy line to the screen

pop dx
pop cx

loop @@NextLine

pop cx
ret
endp ShowBMP


;=================================================
; Description: Graphic Mode
; INPUT: None
; OUTPUT: Screen
; Register Usage: AX
;=================================================

proc stratGraphicMode

 mov ax, 13h
 int 10h

 ret

endp stratGraphicMode

;=================================================
; Description: End Graphic Mode
; INPUT: None
; OUTPUT: Screen
; Register Usage: AX
;=================================================

proc finishGraphicMode
```

```
 mov al, 3
 mov ah, 0
 int 10h

 ret

endp finishGraphicMode

;-----------------------------------------------------------------------
; PrintSecondsElapse -   Interrupt Service Routine (ISR)
;-----------------------------------------------------------------------
;          Input:  none
;          Output: print counter every ticks elapsed
;      Registers:  none
;-----------------------------------------------------------------------
inProgress      db      0
difference      dw      0
lastTimer   dw  0
fixDrift    db  5
counter     dw  0
isTimeUp    db  0

PROC   PrintSecondsElapse
          cmp     [byte cs:inProgress],0
          jne     @@99
          inc     [byte cs:inProgress]
          ; this needs for the processor to be able
   ; recognizing again an external interrupt signals
   sti
          push    ax
          push    ds
          push    dx

   ; this needs to tell the 8259 chip tp pass the
   ; interrupts it receives along to the processor
          mov     al,EOI
          out     PIC8259,al
          mov     ax,BIOSData
          mov     ds,ax
```

```
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;
  ; Y O U R   C O D E
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;
  mov dx, [word LowTimer]     ; read the timmer
  push dx                     ; save the timmer
        sub     dx,[cs:lastTimer]
        cmp     dx, [cs:difference]
        pop dx
  jb    @@20
  dec [cs:fixDrift]
  jnz @@10
  mov [cs:fixDrift], 5
  inc dx
@@10:
  mov [cs:lastTimer], dx
  mov ax, [cs:counter]
                cmp ax, TIMEOUT
                ja @@TimeoutEnd

                push TIME_ROW
                push TIME_COL
  call printAxDec
  inc [cs:counter]
                jmp @@20

@@TimeoutEnd:
        mov [cs:isTimeUp], 1
 ;TODO: display Time is over

@@20:
        cli
        dec     [byte cs:inProgress]
        pop     dx
        pop     ds
        pop     ax
@@99:
        iret
ENDP    PrintSecondsElapse

;-----------------------------------------------------------------------
```

```
;  KeyWaiting - checks if a key press is available
;-----------------------------------------------------------------
;          Input:  none
;          Output:          zf = 0 : (JNZ) Character is waiting to be read
;                    zf = 1 : (JZ) No character is waiting
;          Registers: none (flags only)
;-----------------------------------------------------------------
          ;cursor pos values
          col equ [bp + 4]
          row equ [bp + 6]

PROC printAxDec
                    push bp
                    mov bp, sp
            push ax
    push bx
            push dx
            push cx

                    push ax

                    mov ax, col
                    mov  dl, al   ;Column
                    mov ax, row
                    mov  dh, al   ;Row
                    mov  bh, 0    ;Display page
                    mov  ah, 02h  ;SetCursorPosition
                    int  10h

                    pop ax
    mov cx,0   ; will count how many time we did push
    mov bx,10  ; the divider

put_next_to_stack:

      xor dx,dx
      div bx
      add dl,30h
            ; dl is the current LSB digit
            ; we cant push only dl so we push all dx
      push dx
```

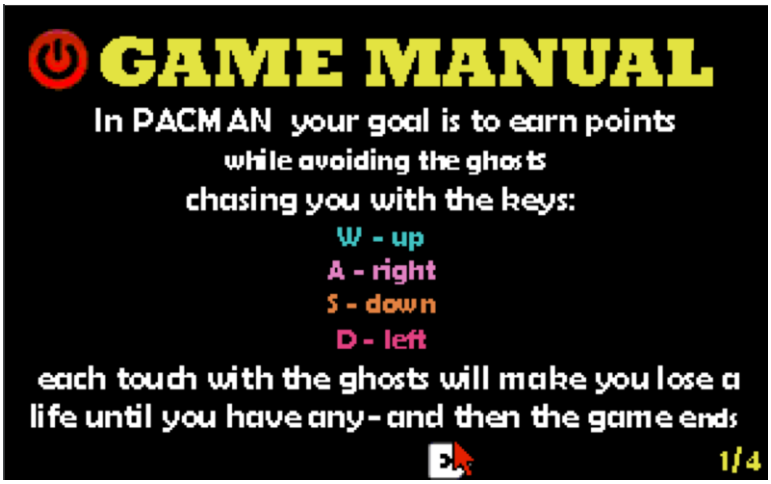
```
    inc cx
    cmp ax,9   ; check if it is the last time to div
    jg put_next_to_stack

        cmp ax,0
        jz pop_next_from_stack  ; jump if ax was totally 0

    add al,30h
        mov dl, al
        mov ah, 2h
        int 21h       ; show first digit MSB



pop_next_from_stack:

    pop ax    ; remove all rest LIFO (reverse) (MSB to LSB)
        mov dl, al
   mov ah, 2h
        int 21h       ; show all rest digits
    loop pop_next_from_stack

        pop cx
        pop dx
        pop bx
        pop ax
            pop bp
   ret 4
endp printAxDec
;=================================================
; Description: Put Matrix on Screen
; INPUT: DX [COL], CX [ROWS], Matrix offset, DI[Adress]
; OUTPUT: Screen
; Register Usage: AX, CX, DX, SI
;=================================================
proc putMatrixInScreen
push es
push ax
push si

cld
```

```
push dx
mov ax,cx
mul dx
mov bp,ax
pop dx


mov si,[matrix]

NextRow:
push cx

mov cx, dx
rep movsb ; Copy line to the screen
sub di,dx
add di, 320


pop cx
loop NextRow


endProc:

pop si
pop ax
pop es
        ret
endp putMatrixInScreen
```

דוגמאות הרצה

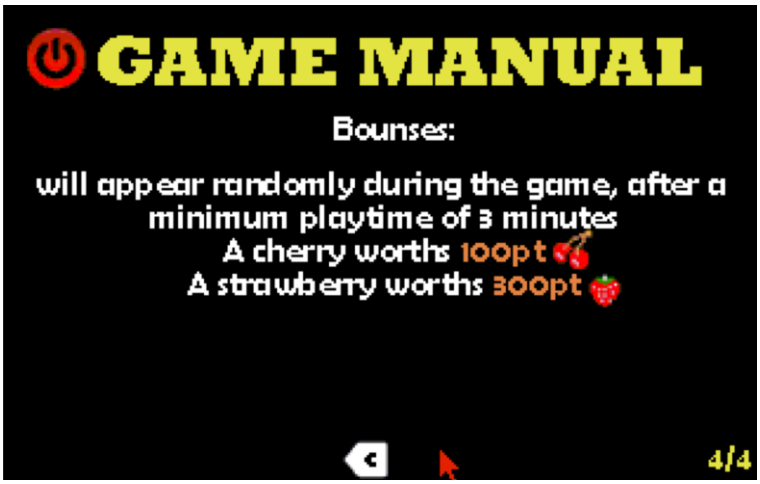
מסך פתיחה:

מסך חוקים:

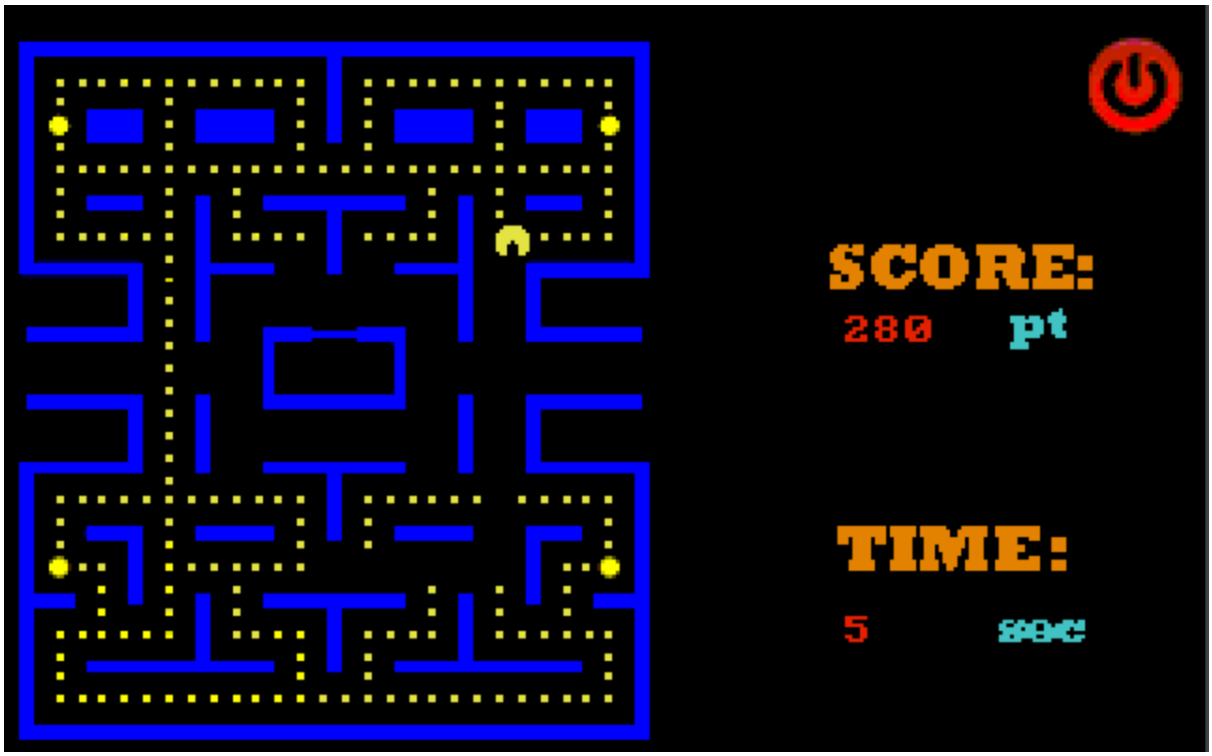

GAME MANUAL
In PACMAN your goal is to earn points
while avoiding the ghosts
chasing you with the keys:
W - up
A - right
S - down
D - left
each touch with the ghosts will make you lose a
life until you have any- and then the game ends
1/4


GAME MANUAL
You can earn points in several ways:
Dots are the simplest way of earning points.
In the game field, dots are spotted all along
the Pacman's paths
Each dot you reach will earn 10pt
2/4


GAME MANUAL
Energize is a big dot
and a special game element that can earn a few things:
Energize will add 50pt to your total score
makes the ghosts disabled - harmless
when ghosts disabled you can eat them:
1st ghost - 200 pt
2nd ghost - 400 pt
3rd ghost- 800pt
4th ghost- 1600pt
3/4


GAME MANUAL
Bounses:
will appear randomly during the game, after a
minimum playtime of 3 minutes
A cherry worths 100pt
A strawberry worths 300pt
4/4

מסך המשחק:



מסך יציאה:

מסך תם הזמן

מסך ניצחון

מסך יציאה לתפריט הראשי

סיכום אישי

כתיבת הפרויקט היא הישג עצום עבורי. בעברי עסקתי בתוכנה ותמיד בליווי הדרכתי כך שמעולם לא התנסתי בכתיבת קוד בממדים האלו לגמרי לבדי. הפרויקט היה הזדמנות מעולה ללמוד על הכוחות שלי ולהאמין בעצמי שהדבר בהחלט אפשרי.

במהלך כתיבת הקוד היו לי לא מעט בעיות גרפיות שדרשו תמיכה תוכנית כדי לפתור אותם. הדוגמה הכי רלוונטית לכך הוא האלגוריתם המוצא את הפוזיציה הבאה של הפקמן. על פניו, הרעיון היה נשמע אפשרי, אבל רק בדיעבד הבנתי שלכל כיוון צריכה להיות מערכת דומה ובכל זאת, שונה. תחילה הפקמן זיהה את הגבול רק לפי הפיקסל המרכזי של החזית ובהמשך שדרגתי את האלגוריתם כך שהבדיקה תערך על כל החזית שלו.

הכתיבה העצמאית לימדה אותי להתעקש על פתירת הבעיות בעצמי. לפני כל קטע קוד שביצעתי, נתתי חשיבות גדולה להבנה של כל שורה כך שלא ייווצר מצב שהקוד שלי בנוי משורות קוד שאינן קשורות, לא משרתות או אפילו מעכבות את העבודה שלי. בעיניי, כדי שאוכל באמת להגיד שאני כתבתי את הפרויקט, עליי להיות בקיאה בכל שורה בו ולהבין אותה.

במהלך הפרויקט אני וחבריי סייענו אחד לשני בעת הצורך. המצב הזה גרם לי להבין כיצד לגשת לקוד שמעולם לא נתקלתי בו.
בנוסף, באמצעות הידע שרכשתי במהלך הכתיבה העצמאית, הפכתי להיות בקיאה בבעיות נפוצות שעלולות לקרות, דבר שתרם לי בעת ההתמודדות עם הקוד שלי ועם הקוד של חברי כשנזקקו לעזרתי.

במסגרת הפרויקט השתמשתי ב GitHub [Git repository hosting service], שהפך את העבודה שלי להרבה יותר יעילה ונכונה בעצם האפשרות לראות את השינויים שעשיתי בין הגרסה הנוכחית לקודמת ובעבודה מסודרת.

לסיום, הפרויקט היה לאתגר עבורי- אבל אחד כזה שבאמת אפשרי במסגרת הזמן שניתן. העבודה לימדה אותי רבות ותרמה לי ידע שבהחלט אנצור עימי להמשך הדרך.