ספר פרוייקט סייבר - הדמקה של איתמר

מוגש על ידי: איתמר ליטוין

מורה: שמואל מצא

בית ספר: הנדסאים הרצליה

תוכן עניינים

הפרוייקט.....3

הוראות המשחק.....4

תיאור החלקים המרכזיים בפרוייקט.....5

תיאור זרימת המשחק.....8

פרוצדורות בקוד.....9

משתנים בקוד.....12

תרשימי זרימה.....15

צילומי מסך.....17

רפלקציה.....20

הפרוייקט

המשחק "הדמקה של איתמר" הוא משחק חשיבה ואסטרטגיה לשני שחקנים הוא דומה מאוד למשחק דמקה אבל אין אפשרות לבצע אכילה כפולה ואין מלכות אבל אפשר לאכול ולזוז לאחור.
הפרוייקט נכתב בשפת ASSEMBLY, ב - "Graphics Mode".

הפרוייקט מורכב מ-5 מסכים:

1. MENU - מסך פתיחה המציג את הפרוייקט
2. INSTRUCTIONS - מסך ההוראות של המשחק
3. GAME - מסך המשחק
4. END - מסך סיום המשחק, המציג את המנצח ואת האפשרות לצאת מהמשחק
5. EXIT - מסך יציאה (מסך סיום שחור ריק)

הוראות המשחק

המשחק "הדמקה של איתמר" הוא משחק חשיבה ואסטרטגיה לשני שחקנים.
לכל שחקן יש 12 חיילים בצבע שלו (אדום - שחקן 1), (ורוד - שחקן 2).
כל שחקן בתורו בוחר להזיז את החייל שלו או לאכול את החייל של השחקן השני.
המקומות החוקיים מסומנים בריבועים ירוקים ואין אפשרות להזיז את החייל למקום אחר חוץ מהריבועים הירוקים.
שחקן יכול לבחור חייל אחד ואם הוא לא מזיז אותו, הוא יכול להחליף את הבחירה לחייל אחר.
מטרת המשחק: לאכול את כל החיילים של השחקן השני.
בחירת החייל והמיקום אליו מזיזים אותו נעשה באמצעות העכבר והמקש השמאלי.

תיאור החלקים המרכזיים בפרוייקט

המערך הדו ממדי של הלוח וההתייחסות ללוח במשחק - במשחק שלי אני עושה שימוש במערך דו מימדי שמקשר בין המסך (הלוח) לבין התזוזה של החיילים, כל פעם שחייל זז אני משנה ערכים בהתאם במערך הדו ממדי.
הערכים במערך הם:
12 = שחקן 1
11 = שחקן 2
2 = ריבוע שחור
0 = ריבוע לבן
3 = ריבוע ירוק (מקום תזוזה חוקי)
יש לי פרוצדורות שאיתם אני יכול לפנות למערך לשנות ערכים וגם לפנות למערך ולהבין מהו הערך שנמצא בתא במערך שקשור לערך (x,y) שבלוח שעל המסך.

אני בחרתי להתייחס ללוח במשחק כך שה - X הוא בין 7 - 0 וה - Y בין 7 - 0 וכך אני יכול להוסיף למשתנים שהם (x_block ו - y_block) כדי לגשת לערכים ולשנות ערכים במערך בזיכרון וכך לבדוק מהלכים חוקיים ולבצע "אכילה" של חיילים.

דוגמא לשינוי ערכים לפי x,y נתונים.

(x-1, y-1)	top	(x+1, y-1)
	(x, y)	
(x-1, y+1)	bottom	(x+1, y+1)

אני בחרתי שה - 0 יהיה בצד שמאל וה - 7 יהיה בימין.
בקוד שלי אני הבאתי כיוונים (top ו - bottom) לתזוזה (ניתן לראות בדוגמא) **(בכל הקוד התייחסתי לכיוונים שבחרתי לכיוונים של המשתנים)**.
אבל בגלל שהכיוון הקדמי של כל שחקן היא שונה אני צריך לשלוח ערכים שונים כדי לשנות את הערכים של ה(x,y) ולכן אני צריך להתייחס למשתנים של המהלכים בצורות שונות ביחס לשחקן שעושה את תורו.

המשתנים שאיתם אני יודע איזה מהלך לבצע הם:

```
free_block_jump_top_right db 0
capture_top_right db 0
free_block_jump_bottom_right db 0
capture_bottom_right db 0
free_block_jump_top_left db 0
capture_top_left db 0
free_block_jump_bottom_left db 0
capture_bottom_left db 0
```

בשחקן 1 הכיוונים הם:

top
Player1
bottom

בשחקן 2 הכיוונים הם:

bottom
Player2
top

וכך כאשר אני זז לימין למעלה בשחקן 1 אני זז ימין למטה בשחקן 2 ובקוד אמנם הבדיקה היא אותה בדיקה (בפרוצדורה check_what_to_do) אבל ההתייחסות למשתנים של המהלכים היא שונה.

דוגמא בבדיקת מהלך של שחקן 2:

```
;; top right is bottom right in player2
cmp free_block_jump_top_right,1
je draw_player2_place
```

דוגמא בבדיקת מהלך של שחקן 1:

```
;; top right is right in player1
cmp free_block_jump_top_right,1
je draw_player1_place
```

<u>הבחירה שכל הגרפיקה במשחק תהיה באותו גודל</u> - במשחק שלי לא רציתי להתעסק עם גרפיקה אלא עניין אותי יותר הקוד לכן כדי לחסוך זמן וחישובים מיותרים בחרתי שכל דבר גרפי בפרוייקט (חוץ מהכיתוב) יהיה בגודל אחיד של 25 * 25 פיקסלים (זה הגודל שאפשר לי לעשות ריבוע של 200 * 200 פיקסלים על המסך כדי שיהיה לי לוח של 8 * 8 ריבועים) כך אני לא צריך לשלוח ערכים שונים של גובה ואורך לפעולה ואני יכול להשאיר את הגדלים כקבוע וגם הבחירה שכל הגרפיקה תהיה בגודל אחיד איפשרה לי שכל

הגרפיקה במשחק תהיה מיוצרת על ידי אותה פרוצדורה דבר שחסך לי שורות קוד רבות וגם זמן רב.

<u>ההתייחסות לx_block ולy_block -</u> בקוד שלי יש 2 משתנים מרכזיים (x_block, y_block) שמחזיקים את הערכי x,y של הריבוע בלוח וכך אני יכול לפנות לערכים במערך שמייצג את הלוח וגם לשנות אותם.
בגלל שהרבה מהפרוצדורות מקבלות את הx_block וy_block אז בחרתי שיהיו שני קבועים ושכאשר פעולה מקבלת את המשתנים האלה אני אשלח אותם באותו סדר כל פעם.
הקבועים הם:

```
X_BLOCK_PUSHED EQU bp + 6
Y_BLOCK_PUSHED EQU bp + 4
```

כדי שיהיה יותר קל למשתמש להבין את הערך של הקבועים בכל פרוצדורה שיש אותם רשמתי את הקבועים ואת הערך שלהם (במקום לכתוב את הקבועים פעם אחת בכל הקוד).

תיאור זרימת המשחק

1. מסך הפתיחה והמידע המוצג על המסך.
2. קליטת לחיצת מקש וביצוע פעולה בהתאם:
   - טיפול בלחיצה על 'Y' - הצגת מסך המשחק
   - טיפול בלחיצה על 'I' - הצגת מסך ההוראות
   - טיפול בלחיצה על 'X' - הצגת מסך יציאה
3. מסך המשחק מוצג על גבי המסך.
4. קליטת לחיצת עכבר וביצוע פעולה בהתאם:
   - לחיצה ראשונה - בחירת שחקן והצגת המקומות האפשריים לזוז אליהם
   - לחיצה שנייה - הזזת החייל למקום המיועד/בחירת חייל אחר וחזרה לסעיף 4.א.
5. בדיקת ניצחון והצגת מסך ניצחון וטיפול בלחיצות המקשים בהתאם:
   לפי סעיף 2, אפשרות לצאת מהמשחק(לחיצה על X) או לחזור למסך הפתיחה (לחיצה על Y).

פרוצדורות בקוד

פרוצדורה	הסבר	הפעולה מקבלת	פלט	הדפסה
get_mouse_pos	מחזיר את מצב הכפתורים של העכבר ואת הערך X,Y של הלחיצה על המסך	**הפעולה לא מקבלת כלום**	מיקום האיקס ל x_pos מיקום הווי ל y_pos מצב הכפתורים לmouse	**אין הדפסה**
get_block	מוצא את הx,y בלוח של המקום שנלחץ	הפעולה מקבלת: x_pos , y_pos	הערך x,y על הלוח נכנס ל(x_block, y_block)	**אין הדפסה**
get_pixel	מחזיר ערך פיקסל לפי הערך x,y של הריבוע בלוח (בשביל לצייר)	הפעולה מקבלת: x_block, y_block	(הערך של הפיקסל השמאלי העליון נכנס למשתנה image_pos_block)	**אין הדפסה**
find_num_in_arr	מוצא את הערך במערך הדו מימדי לפי הx,y של הריבוע בלוח	הפעולה מקבלת: x_block, y_block	הערך של המספר שנמצא בתא המשוייך לערך (x,y) של הלוח שנשלחו נכנס ל(num_in_arr)	**אין הדפסה**
find_num_in_block	מקבל מיקום עכבר ומוצא את המספר שנמצא בריבוע בלוח המתאים	הפעולה מקבלת: x_pos, y_pos x_block,y_block	הפעולה מכניסה לnum_in_arr את המספר המתאים לפי מה שיש בריבוע בלוח	**אין הדפסה**
change_num_in_arr	משנה את הערך במערך בזיכרון לפי הx,y בלוח ולפי הערך שאני רוצה לשנות	הפעולה מקבלת: x_block, y_block, ומספר שאני דוחף (המספר שאני רוצה להחליף)	משנים את המספר במערך	**אין הדפסה**
draw_different_blocks	מצייר את הריבוע במיקום לפי מיקום פיקסל על המסך	הפעולה מקבלת: x_block, y_block והאופסט של הריבוע	**אין פלט**	הריבוע מצוייר על המסך

	שהוא מוצא על ידי get_pixel (שימוש בx_block ו y_block)	שרוצים לצייר (חייל / ריבוע בצבע)		
draw_screen_by_arr	מצייר על המסך את הלוח לפי המערך בזיכרון	**הפעולה לא מקבלת כלום**	**אין פלט**	הלוח מצוייר לפי המערך בזיכרון
check_legal_move_player	מוצא את כל המהלכים החוקיים לשחקן מכל 4 הצדדים	הפעולה מקבלת סוג החייל ו,x_block y_block	רק שינויים במערך בזיכרון	**אין הדפסה**
check_what_to_do	בודק איזה מהלך החייל מבצע ומכניס לתוך משתנה 1 אם ניתן לעשות את הפעולה (המשתנה משמש כמשתנה בוליאני) ולפי המשתנה יודעים לאן להזיז את החייל (איזה פרמטרים לשלוח ל-move_player)	הפעולה מקבלת: x_block, y_block ו,x_block_temp y_block_temp	רק שינויים במשתנים של הכיוונים	**אין הדפסה**
reset_screen	מוחק במערך בזיכרון את כל הערכים 3 (מקום תזוזה חוקי) ומחליף אותם במקומות ריקים (2) וכך ניתן לצייר את הלוח ללא ריבועים ירוקים	**הפעולה לא מקבלת כלום**	רק שינויים במערך בזיכרון	**אין הדפסה**
reset_all	אם השחקן רוצה לשחק שוב אני משתמש במערך השני של הלוח כדי לשנות את כל הערכים ללוח למצב הראשוני כדי לשחק שוב	**הפעולה לא מקבלת כלום**	רק שינויים במערך בזיכרון	**אין הדפסה**
delay	יוצר עיכוב קבוע	**הפעולה לא מקבלת כלום**	רק דיליי נוצר	**אין הדפסה**
check_mouse_on_board	בודק אם העכבר על הלוח	**הפעולה לא מקבלת כלום**	**אין פלט (**הפעולה נמצאת בלולאה עד שהעכבר על הלוח**)**	**אין הדפסה**

find_player_pressed	מראה את המיקומים האפשריים לתזוזה של החייל, מציג את החייל כנלחץ ומשנה במערך בזיכרון את הערך של הריבוע החייל נמצא עליו לריק (2)	הפעולה מקבלת את סוג החייל האופסט של החייל הלחוץ (משתה לפי התורות) ו x_block,y_block	שינויים במערך בזיכרון (המקומות החוקיים ושינוי הריבוע שהשחקן עליו לריק (2))	המסך החדש מצוייר עם החייל הלחוץ והמקומות החוקיים לתזוזה
move_player	מצייר את החייל לפי המיקום בלוח (מוחק את כל האפשרויות לתזוזה(ריבועים ירוקים) ומצייר מחדש את הלוח) משמש לתזוזה של חייל או כאשר השחקן מחליף חייל	הפעולה מקבלת את סוג החייל וה x_block, y_block ואם השחקן משנה את החייל שהוא בחר הפעולה מקבלת סוג החייל ו,x_block_temp y_block_temp	שינויים במערך בזיכרון מחיקת כל המקומות החוקיים	הציור של השחקן על הלוח
capture_player	משנה את הערך במערך בזיכרון של החייל שנלכד ל2 (ריק) בעזרת חישובים של פרמטרים שנשלחים לפרוצדורה דרך המחסנית.	הפעולה מקבלת את המספר שאני מוסיף לx_block המספר שאני מוסיף לy_block ו x_block, y_block	רק שינויים במערך בזיכרון	**אין הדפסה**
print_line	מדפיס שורה במיקום לפי הפרמטרים שנשלחים במחסנית.	הפעולה מקבלת את ערכי ה (x,y) של המחרוזת והאופסט של המחרוזת	אין פלט	הדפסת המחרוזת
player_winner_screen	מציג את המנצח ואת האפשרויות הבאות לשחקן (X = לצאת מהתוכנה, Y = משחק חדש)	הפעולה מקבלת את האופסט של המחרוזת ניצחון המתאים (לפי התור)	אין פלט	הדפסת המחרוזת ניצחון ומחרוזות של האפשרויות

משתנים בקוד

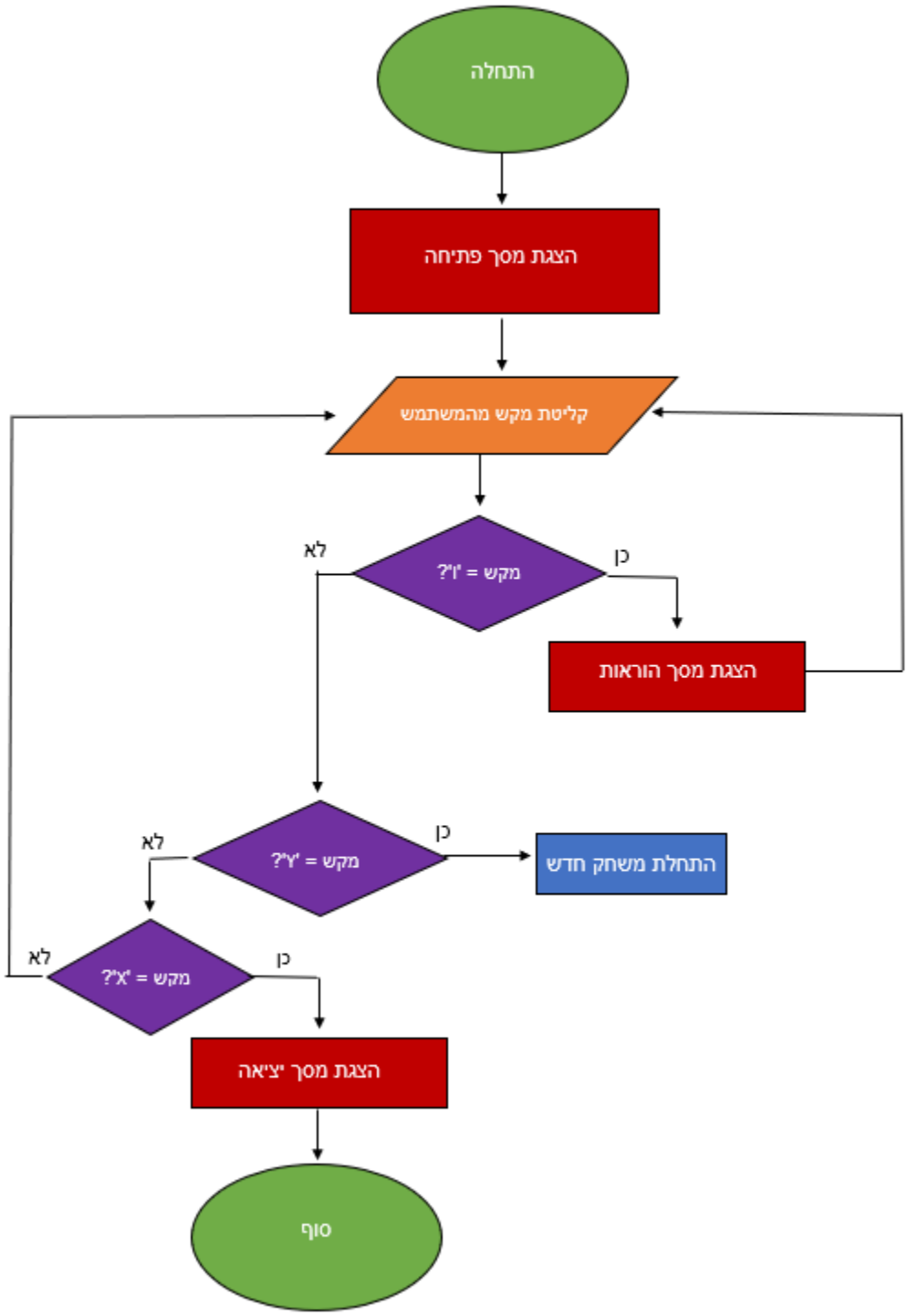
משתנה	הסבר
gray_block	מערך של 25 * 25 של ריבוע אפור
dark_gray_block	מערך של 25 * 25 של ריבוע אפור כהה
green_block	מערך של 25 * 25 של ריבוע ירוק
player1	מערך של 25 * 25 של ריבוע שבו שחקן 1
player2	מערך של 25 * 25 של ריבוע שבו שחקן 2
player_pressed	מערך של 25 * 25 של ריבוע שבו שחקן 1 לחוץ
player2_pressed	מערך של 25 * 25 של ריבוע שבו שחקן 2 לחוץ
board	מערך של 8 * 8 של הלוח
board2	מערך של 8 * 8 של הלוח שמשמש להחזיר את הערכים במערך הראשון לערכים ראשוניים למשחק חוזר
image_pos_block	פיקסל על המסך המשמש מיקום להתחיל לצייר את הריבוע שאני רוצה
x_pos	מיקום x של העכבר
y_pos	מיקום y של העכבר
x_block	הערך X של הריבוע בלוח (0מ עד 7)
y_block	הערך Y של הריבוע בלוח (0מ עד 7)
num_in_arr	המספר במערך שנמצא בפרוצדורה של find_num_in_arr
player_turn	משתנה שמחזיק את תור השחקן (כך אני

	יודע תור מי)
mouse	מחזיק את מצב הכפתורים (אם כפתור שמאל לחוץ הוא הופך ל 1)
points_player1	הניקוד של שחקן 1
points_player2	הניקוד של שחקן 2
player1_wins	מערך ניצחון של שחקן 1
player2_wins	מערך ניצחון של שחקן 2
start_str	מערך פתיחה ראשון
start_str1	מערך פתיחה שני
start_str2	מערך פתיחה שלישי
start_str3	מערך פתיחה רביעי
end_str1	מערך סיום ראשון
end_str2	מערך סיום שני
instructions	מערך הוראות הראשון
instructions2	מערך הוראות שני
instructions3	מערך הוראות שלישי
instructions4	מערך הוראות רביעי
x_block_temp	מחזיק את הערך של הx_block לפני התזוזה (לפני שינוי)
y_block_temp	מחזיק את הערך של הy_block לפני התזוזה (לפני שינוי)
free_block_jump_top_right	משתנה שהוא או 1 או 0 ומשמש לדעת אם אני קופץ לריבוע ריק בצד ימין למעלה
capture_top_right	משתנה שהוא או 1 או 0 ומשמש לדעת אם אני לוכד חייל של שחקן אחר מימין למעלה

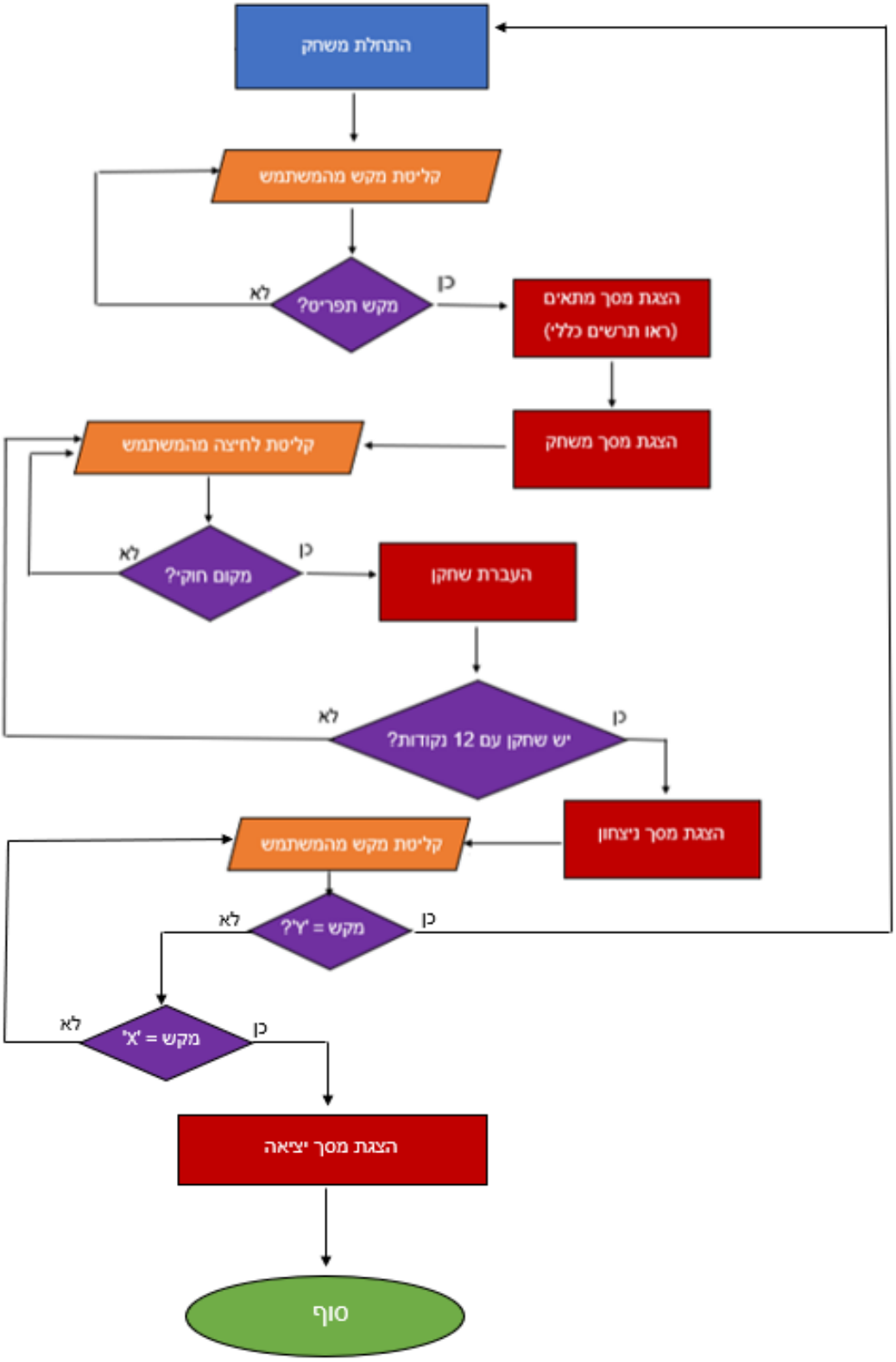
free_block_jump_bottom_right	משתנה שהוא או 1 או 0 ומשמש לדעת אם אני קופץ לריבוע ריק בצד ימין למטה
capture_bottom_right	משתנה שהוא או 1 או 0 ומשמש לדעת אם אני לוכד חייל של שחקן אחר מימין למטה
free_block_jump_top_left	משתנה שהוא או 1 או 0 ומשמש לדעת אם אני קופץ לריבוע ריק בצד שמאל למעלה
capture_top_left	משתנה שהוא או 1 או 0 ומשמש לדעת אם אני לוכד חייל של שחקן אחר משמאל למעלה
free_block_jump_bottom_left	משתנה שהוא או 1 או 0 ומשמש לדעת אם אני קופץ לריבוע ריק בצד שמאל למטה
capture_bottom_left	משתנה שהוא או 1 או 0 ומשמש לדעת אם אני לוכד חייל של שחקן אחר משמאל למטה

תרשימי זרימה

תרשים זרימה כללי:



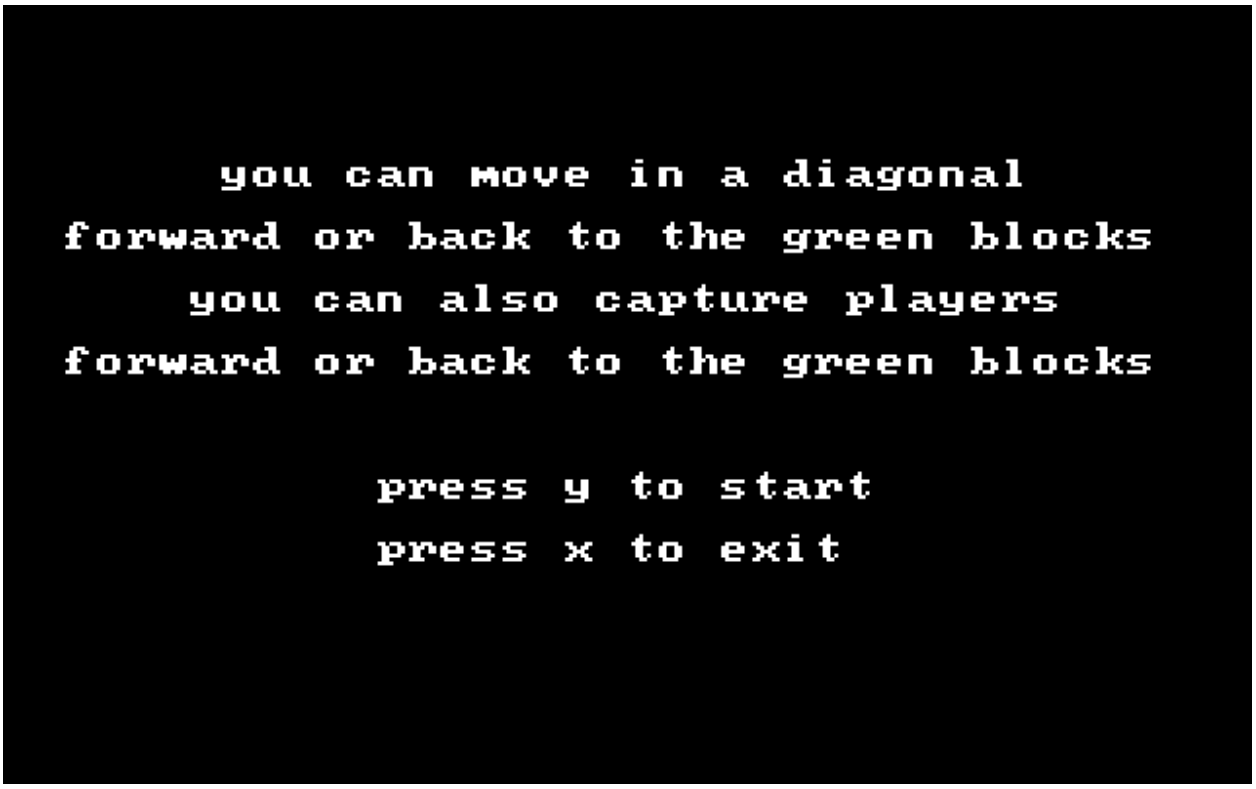
תרשים זרימה - מהלך המשחק:



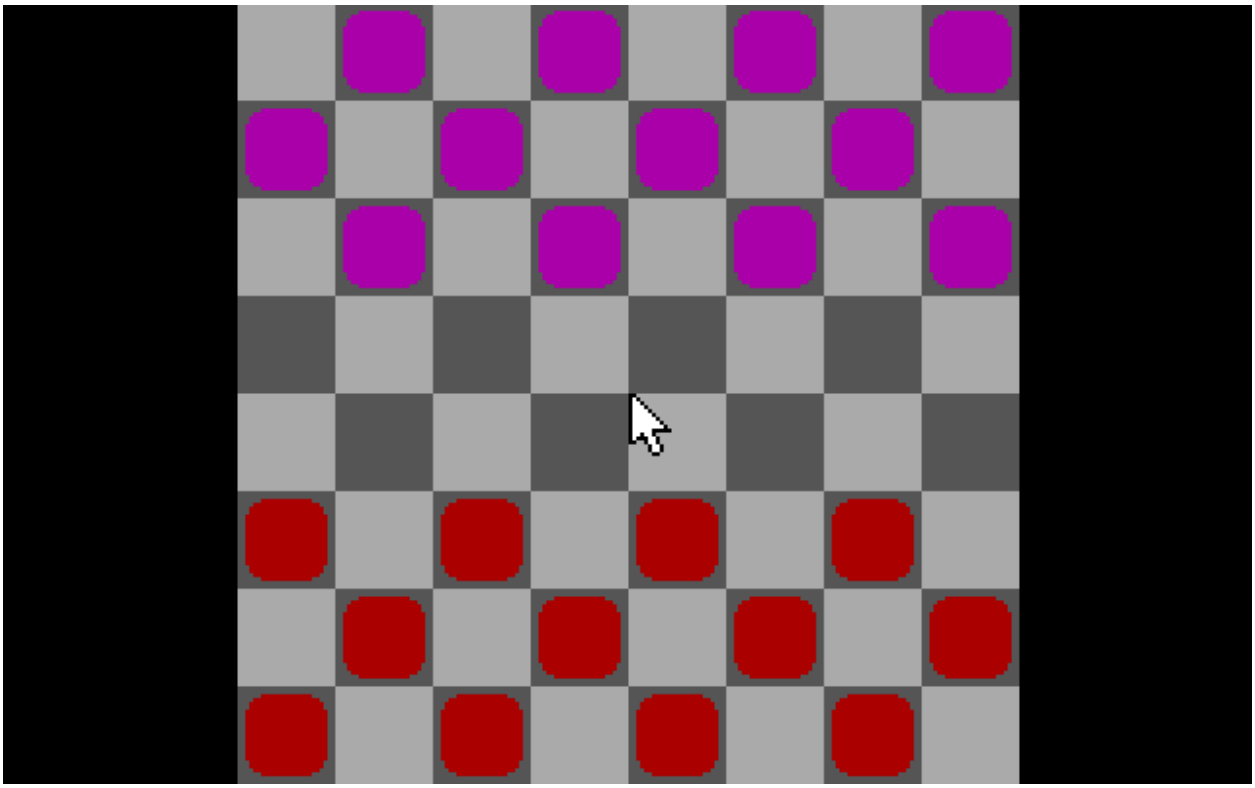
צילומי מסך

מסך פתיחה:

מסך הוראות:



מסך משחק:



מסך הניצחון משתנה לפי כמות החיילים של השחקן שניצח שנשארו.
דוגמאות למסך ניצחון לכל שחקן עם מספר שונה של חיילים:


player 2 wins
press y to play again
press x to exit

רפלקציה

בחרתי במשחק דמקה מכיון שזה היה משחק מעניין ומאתגר מבחינת התכנות וחשבתי שזה פרוייקט מקורי. הפרוייקט הזה היה לי קשה מאוד ולקח לי הרבה זמן. החלק הכי קשה היה להתחיל בגלל שלא ידעתי איזה פרוייקט אני רוצה לעשות ואיך להתחיל אפילו. בסוף בחרתי לעשות משהו קצת מיוחד והחלטתי לאתגר את עצמי ולעשות דמקה אבל בגלל שלא היה לי הרבה זמן וכבר עבדתי הרבה מאוד על המשחק שלי החלטתי לבטל אכילה כפולה ואת האפשרות להפוך למלכות ולהחליף את זה בתזוזה ואכילה אחורית דבר שלדעתי מייחד את הדמקה שלי.

החלק הכי קשה בכתיבת הקוד של הפרוייקט היה החשיבה של איך לחבר את הלוח שאני מציג לשחקן לבין המחשב (איך לחבר בין הקלט של הלחיצה לבין התזוזה של החיילים). בסוף עליתי על דרך שלדעתי היא מאוד חכמה, שימוש במערך בזיכרון שאני יכול לפנות אליו ולמצוא ערך וגם לפנות אליו ולשנות את הערך. אבל אחרי שהצלחתי לעבוד עם המערך שאר החלקים בפרוייקט היו יחסית קלים.

היו לי הרבה מאוד באגים בפרוייקט ולפעמים ישבתי ימים שלמים רק כדי לגלות שלא איפסתי משתנה או ששלחתי את הדבר הלא נכון למחסנית דבר שעיצבן אותי נורא אבל גם הצחיק אותי.

בתחילת הפרוייקט היה לי קשה ולא כיף אבל ככל שהתקדמתי למדתי והבנתי יותר על אסמבלי והתחיל להיות לי כיף לעבוד על הפרוייקט.