דמקה - Checkers Game

27.05.2016

פרטי הפרויקט

מגיש: אור נבו

ת"ז: 212212054

מורה מלווה: פימה ניקולייבסקי

כיתה: י'

בית ספר: בית הספר הדמוקרטי לב השרון

מידע נוסף

שם העבודה: דמקה
שם הקובץ: main.exe
קבצים נלווים: main.asm, AI.asm, convert.asm, graphics.asm, logic.asm, menu.asm, renderer.asm, select.asm
סביבת עבודה: Turbo Assembler
סביבת פיתוח: Atom text editor
סביבת הרצה: Dosbox

תוכן עניינים

על העבודה

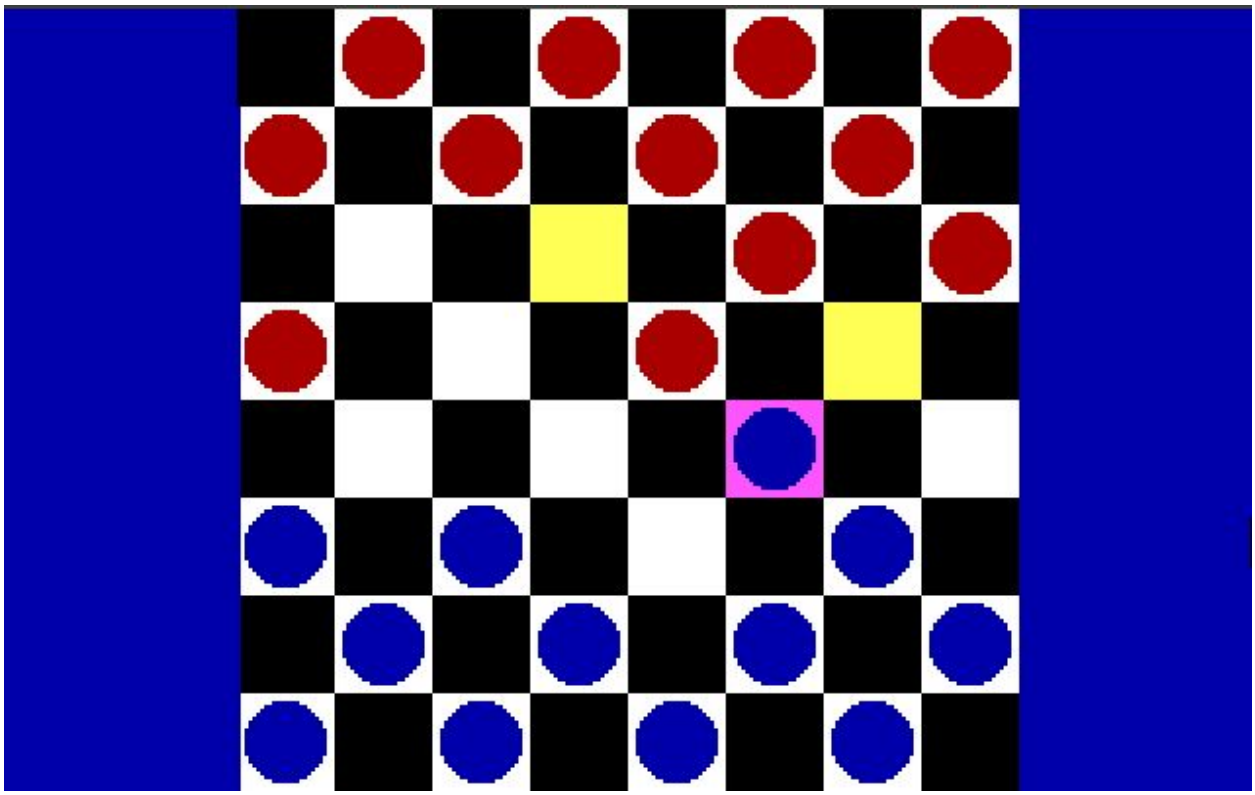
העבודה היא בנית המשחק הידוע דמקה. המשחק בנוי כולו באסמבלי, וכל הקוד נכתב מבסיסו על ידי.
במשחק שני מצבי משחק: שחקן נגד שחקן, ושחקן נגד בינה מלאכותית.
הממשק של התוכנה עם השחקן היא בעזרת העכבר, כשהוא לוחץ על המשבצות אותן הוא בוחר, המשחק מזהה איפה הוא לחץ, ופועל בהתאם. המשחק נגמר כלשאחד השחקנים נאכלו כל החיילים, או שלאחד השחקנים אין יותר מהלכים אפשריים.
היו מספר אתגרים בהם נתקלתי בדרכי לפרויקט הגמור, בינהם: שמירה בזכרון של פרטי הלוח, הצגתו על המסך, יעילות, זיהוי ופירוש של לחיצת עכבר, הגבה לפעולות מסוימות ואסירת אחרות ובניית הבינה המלאכותית באופן שיקרא תיגר על השחקן האנושי.

את כל המשחק בניתי לפי עקרונות תכנות פרוצדורלי והקפדתי על כתיבת קוד כללי וreusable.
בפרויקט 3220 שורות סך הכל, מתוכן 397 הן comments ו2178 הן שורות קוד.

חוקי המשחק

למשחק חוקים פשוטים למדי, והם זהים לחוקי משחק הדמקה המקורי. כל אחד משני השחקנים מתחיל עם 12 חיילים על הלוח בצד שלו. מטרת המשחק היא לאכול את כל חיילי השחקן השני, והמנצח הוא זה שחייליו הם היחידים שנשארו על הלוח - זה שאכל את כל חיילי היריב.
התזוזה נעשית באלכסון, כשמותר ללכת רק למשבצת ריקה.
אכילה מותרת במצב הבא: אם יש באלכסון לחייל שלך חייל אויב, ובמשבצת ש"מעבר" לו אין כלום, מותר לך לאכול את החייל הזה.

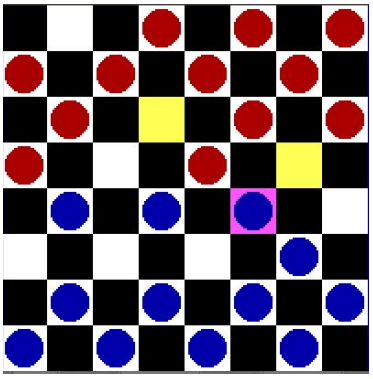
דוגמא: השחקן הכחול יכול לאכול את השחקן האדום.



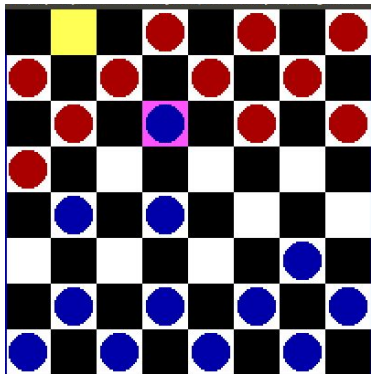
אכילה כפולה מותרת אם לאחר שאכלת יש לך עוד אפשרות אכילה עם אותו חייל שכרגע אכל.

דוגמה:

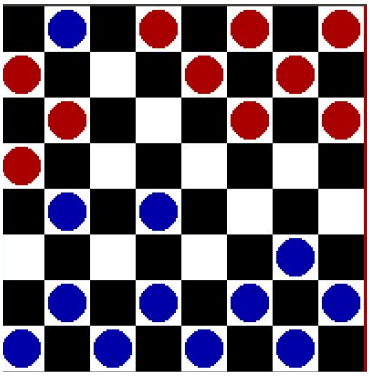
1: הכחול יכול לאכול את האדום



2: לאחר הכחול אכל, יש לו אפשרות לאכילה כפולה



3: הוא מימש אותה, ומאחר ואין לו עוד אפשרות אכילה, הועבר התור



כיוון שתזוזה ואכילה אחורה הן אסורות, יכול להיווצר מצב שבו חייל תקוע בסוף הלוח, והחייל הזה יהיה חייל "מנוטרל". המשחק נגמר באחד משני המצבים הבאים: לאחד השחקנים נאכלו כל החיילים או שלאחד השחקנים יש רק חיילים מנוטרלים.

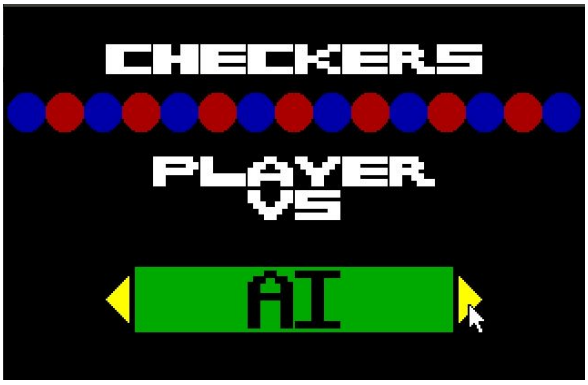
מדריך לשחקן

מסך התפריט

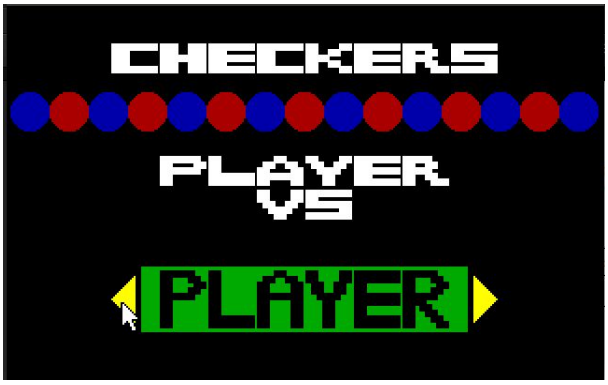
תפעול המשחק הוא פשוט למדי. עם הפעלת המשחק, יעלה מסך התפריט ובוא אפשרות לבחירת מצב משחק: שחקן נגד שחקן או שחקן נגד בינה מלאכותית.
בין האפשרויות ניתן להחליף בעזרת לחיצה על החצים הצמודים להן: חץ ימיני לשחקן נגד בינה מלאכותית, וחץ שמאלי לשחקן נגד שחקן.

דוגמא:

שחקן נגד בינה מלאכותית



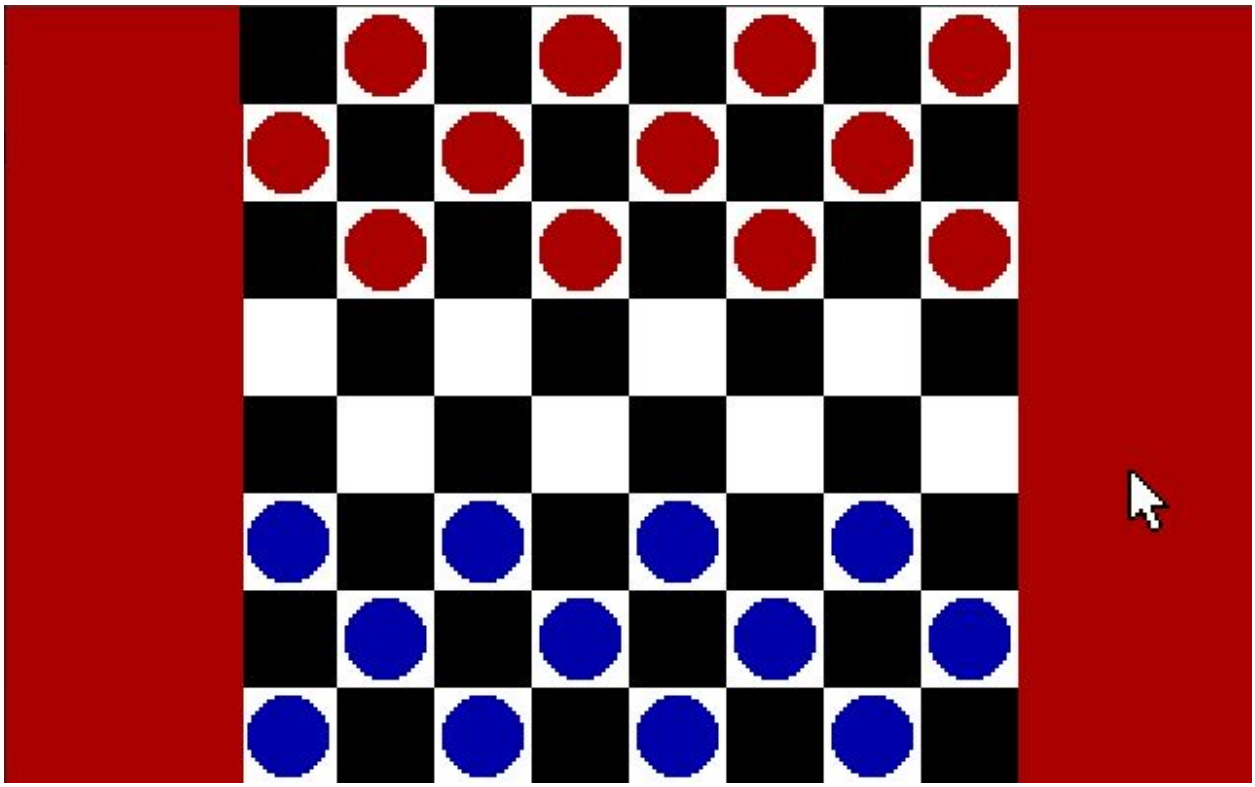
שחקן נגד שחקן



לאחר שמחליפים לאפשרות הרצויה, יש ללחוץ על מקש האנטר לבחירה. לאחר הבחירה, יעלה מסך המשחק.

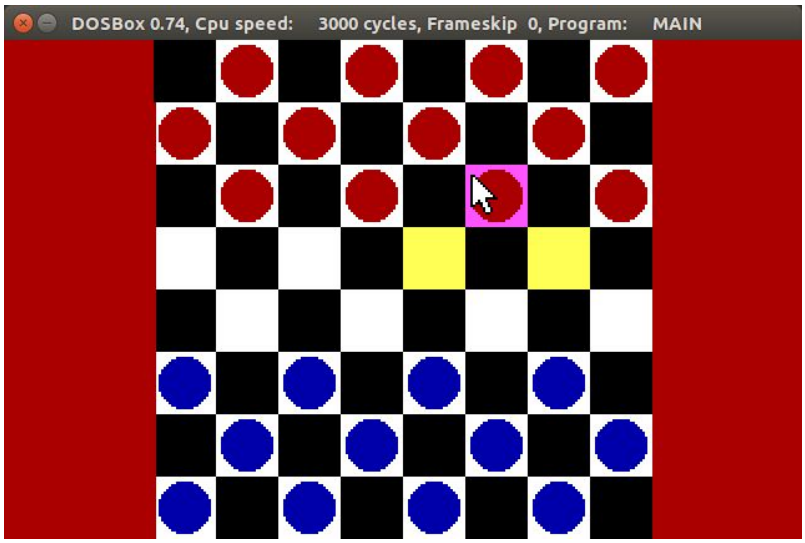
מסך המשחק ותזוזה

דוגמא: מסך המשחק



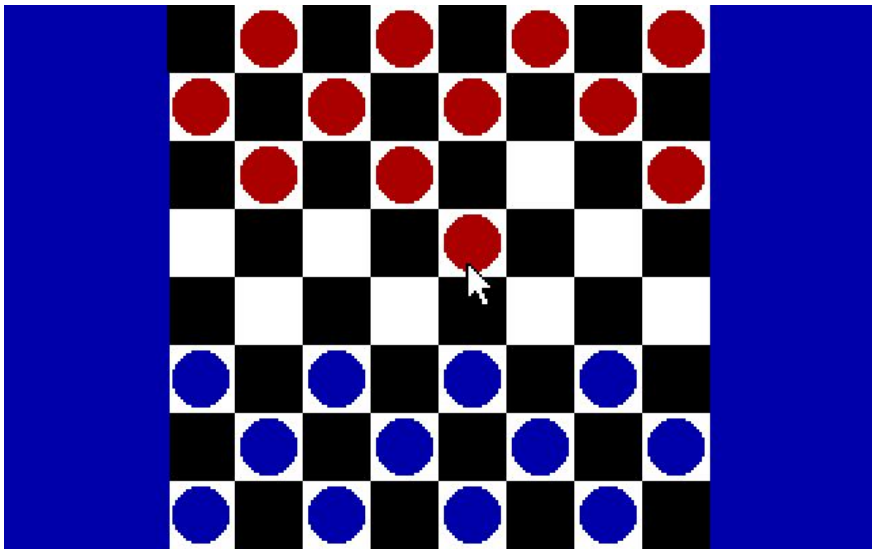
במסך המשחק יש את הלוח שבמרכז, שעליו החיילים של שני השחקנים במיקומיהם ההתחלתיים. משני צדיו של הלוח קיימות זוג רצועות שמראות את השחקן שהתור שלו. למשל, בדוגמא לעיל ניתן לראות שהשחקן האדום מתחיל, כיוון שהרצועות שבצידי המסך הן אדומות.

הממשק עם השחקן נעשה, כפי שכבר נאמר, בעזרת העכבר. כדי לבחור שחקן להליכה, יש ללחוץ על המשבצת בה הוא נמצא. לאחר בחירת שחקן, משבצת השחקן תצבע בסגול כסימון לבחירה, והמשבצות אליהן הוא יכול ללכת יצבעו בצהוב (דוגמא בעמוד הבא).



כדי ללכת לאחת המשבצות הללו, יש ללחוץ עליה. אם השחקן ראה את אפשרויות ההליכה שלו וברצונו להתחרט ולבחור חייל אחר, יש ביכולתו לשנות \ לבטל את הבחירה על ידי לחיצה על מקום שאינו משבצת מסומנת. התור של שחקן יגמר לאחר שהלך.

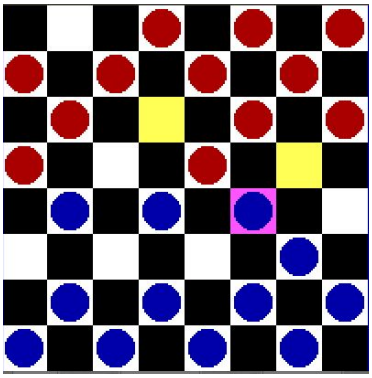
דוגמא: לאחר הליכה



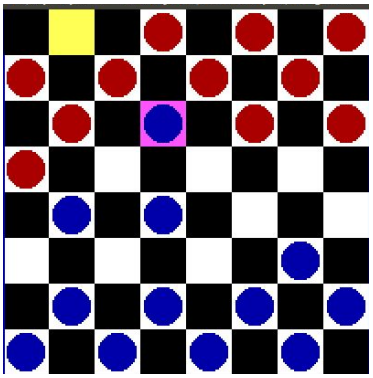
עתה, אחרי שהשחקן האדום הלך, נגמר תורו ועבר התור לשחקן הכחול.
אכילה נעשית באופן דומה להליכה: אם יש אפשרות אכילה, המקום אליו ניתו ללכת ובכך לאכול יסומן, ויהיה ניתן לבחורו בכדי לממש את אפשרות האכילה. לאחר אכילה, אם יש אפשרות לאכילה כפולה, תסומן האפשרות. אם השחקן לוחץ עליה, האכילה הכפולה מתבצעת. אם הוא לוחץ במקום שאינו מסומן, לא מתבצעת האכילה, והתור עובר לשחקן השני.
להלן הדוגמא שהובא גם ב"על העבודה".

דוגמה:

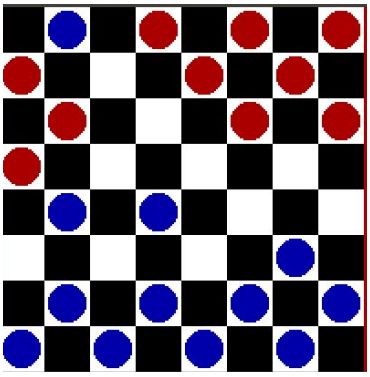
1: הכחול יכול לאכול את האדום



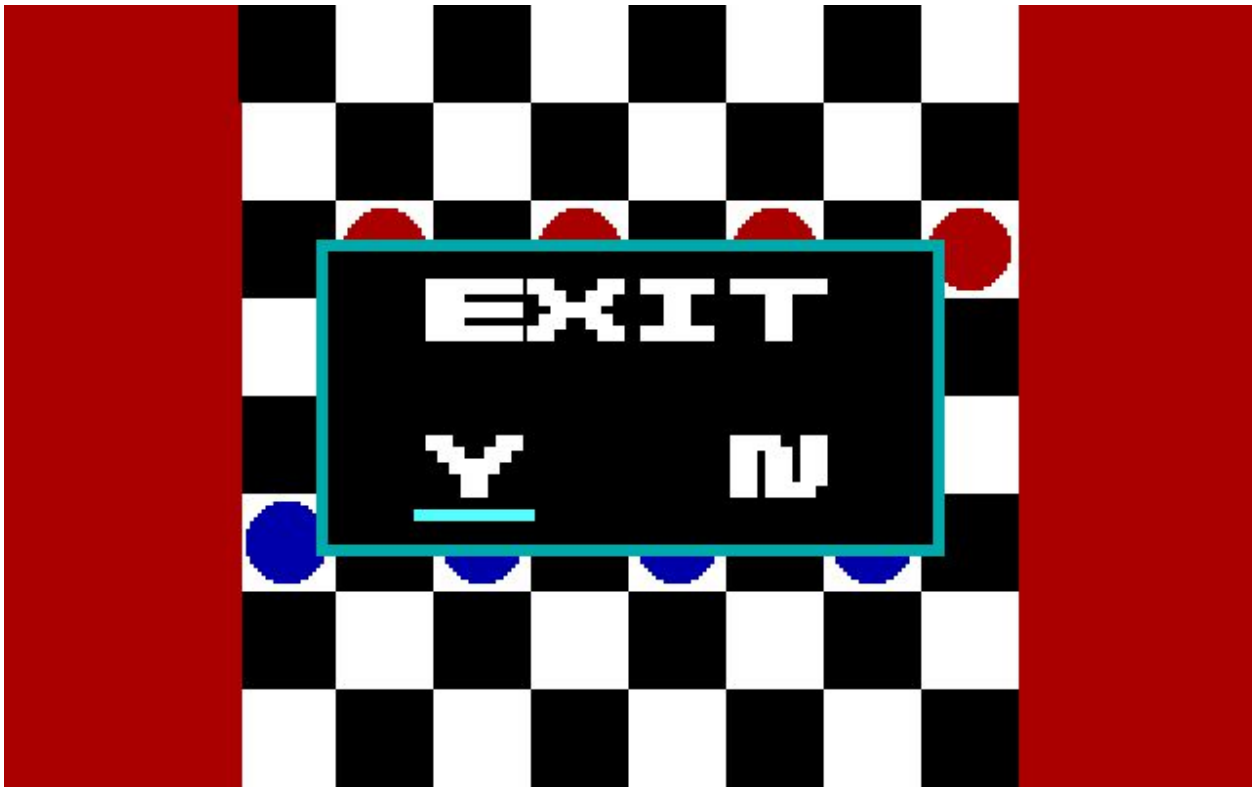
2: לאחר שהכחול אכל, יש לו אפשרות לאכילה כפולה



3: הוא מימש אותה, ומאחר ואין לו עוד אפשרות אכילה, הועבר התור



בכל שלב ניתן לצאת מהמשחק על ידי לחיצה על מקש הEsc. אם לוחצים על Esc בזמן המשחק עצמו (לא בזמן התפריט), יופיע מסך שישאל האם אתה בטוח שאתה רוצה לצאת. ניתן להחליף בין האפשרויות עם כפתורי הN והY, כשN זה לא וY זה כן. לבחירה יש ללחוץ Enter.



לאחר סיום המשחק, יופיע מסך הסיום. כדי לצאת ממנו ולסגור את המשחק, יש ללחוץ בכל מקום על המסך או ללחוץ על מקש הEsc.

מסך הסיום

פירוט גרסאות

I. V0.1.2

Added: Board saving and printing.

II. V0.2.5

Added: Players and circles added.

III. V0.3.7

Added: selecting panel on click and coloring it.

Added the nextTurn procedure.

IV. V0.5.0

Added: mark possible moves and more base procedures.

V. V0.6.2

Added: mark two sides, deselect and exit on esc.

VI. V0.7.5

Added: movement and eating.

VII. V0.8.7

Added: Double eating.

VIII. V1.0.0

Fixed double eating bugs.

IX. V1.1.2

Added two base functions for AI, started menu.

X. V1.2.5

Menu finished
Added the function getDefensiveScore for the AI, and finished other AI procedures.

XI. V1.3.7

Finished the getScore AI proc, the getOffensiveScore proc and helping procedures.

XII. V1.5.0

Fixed moving and eating bugs, started AITurn.

XIII. V1.6.2

AI is moving.

XIV. V1.7.5

AI is eating.

XV. V1.8.7

Added: to prevent eating can eat as respond, move by himself and move others to block eating.

XVI. V2.0.0

Added: game over recognition and screen.

XVII. V2.1.2

Fixed getOffensiveScore bug.

XVIII. V2.2.5

Re-wrote the 'getBeingEatenTimes' procedure.
Fixed getDefensiveScore bug.
Fixed 'IsExposedForEatingSide' bug.

XIX. V2.3.7

Fixed final bugs and added “Are you sure you want to exit?” prompt.

מבנה התוכנית

פורמט הזכרון

הלוח נשמר כמערך דו מימדי של בייטים בזכרון. באופן מעשי, מערך דו מימדי הוא פשוט מספר מערכים ששמורים אחד לאחר השני בזכרון. ה"גובה" של המערך הוא מספר התת-מערכים שיש בו, והרוחב הוא אורך התת-מערכים.
למשל, המערך הדו מימדי הבא:

VAL0,0	VAL0,1
VAL1,0	VAL1,1
VAL2,0	VAL2,1

ישמר בזכרון כך:

VAL0,0 | VAL0,1| VAL1,0 | VAL1,1 | VAL2,0 | VAL2,1

בכדי לשמור את הלוח, יצרתי מערך דו מימדי של 8 על 8, כמו לוח הדמקה, וכל תא במערך הזה מייצג משבצת בלוח.
כדי לשמור את כל המידע החיוני למשבצת בbyte בודד, החלטתי שאשמור את פרטי המשבצות בפורמט הבא:

Unused	Unused	Unused	**Is marked**	**Is selected**	**Which player**	**There is a player there**	**Panel color**
Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1

כפי שניתן לראות, כל משבצת נשמרת כ5 דגלים:

1. הדגל הראשון מסמן האם המשבצת לבנה (1) או שחורה (1).
2. הדגל השני מסמן האם יש שם חייל כרגע (1 אם יש).
3. הדגל השלישי מסמן איזה שחקן נמצא שם: 0 לשחקן האדום ו1 לשחקן הכחול.
4. הדגל הרביעי מסמן האם המשבצת נבחרה על ידי השחקן (כלומר, נלחצה וסומנה בסגול).
5. הדגל החמישי מסמן האם המשבצת מסומנת (בעלת פוטנציאל להליכה אליה, מסומנת בצהוב).
6. הביט השישי, הביט השביעי והביט השמיני אינם בשימוש.

גרפיקה

ישנן כמה פרוצדורות מרכזיות שעיסוקן הוא בגרפיקה. הפרוצדורות הבסיסיות ביותר הן פרוצדורות שמקבלות מיקום וצבע, ומציירות שם עיגול \ מרובע \ משולש (בהתאם לשיטה). מעליהן, יש פרוצדורה בשם renderPanel. הפרוצדורה renderPanel מקבלת מיקום בזכרון של מידע של משבצת, ולפיו היא מציירת אותו על המסך.
נניח שהיא קיבלה תא במיקום DS:0008. היא תפנה לפרוצדורה אחרת, שהופכת מיקום בזכרון למיקום על המסך, וכך תדע איפה לצייר את המשבצת. שאר המידע לאיך לרנדר נמצא בתוך תא המשבצת בזכרון: צבע המשבצת, האם יש שם שחקן, צבע השחקן, האם נבחרה והאם מסומנת. את המיקום על המסך הפרוצדורה הנפרדת יכולה להסיק מאחר ואנחנו יודעים את אורכי התת-מערכים. לכן, נוכל לדעת את המיקום בלוח, ואז להכפיל אותו בגודל כל משבצת.

מאחר ואורך צלע המשבצות היא 25px, ומיקום המשבצת DS:0008 היא Y=1, X=0 בלוח, אז המיקום על המסך יהיה Y=Y*25=25, X=X*25+marginLeft=60. הmarginLeft נוסף כדי למרכז את הלוח על המסך.
בואו ונניח שתוכן המשבצת DS:0008 היא 00000111b. נוכל להסיק מזה שהמשבצת היא לבנה, יש בה חייל כחול, היא לא נבחרה והיא לא מסומנת.
עכשיו, הפרוצדורה ראשית תצייר משבצת לבנה במיקום שחושב. לאחר מכן, היא תצייר שם עיגול בצבע כחול, ובכך תסיים את פעולתה.

יצרתי 6 פרוצדורות להמרה בין שלושת סוגי הנתונים: מיקום בזכרון, מיקום בלוח ומיקום על המסך. הם ממירים לפי המשוואות הבאות:

working according to this formula:

```
datasegLocation = the location in the dataseg
tableY  = datasegLocation / 8
tableX  = datasegLocation - tableY*8
pixelsX = tableX * panelSize + marginLeft
pixelsY = tableY * panelSize
```

where the table locations are cells numbers - first cell is 0,0 and last cell is 7,7

גרפיקה בתפריט

הגרפיקה בתפריט היתה מעט שונה. כדי לעשות את הכותרות, שמרתי את האותיות בזכרון כך:

```
11111111b, 11100111b, 11111111b, 11111111b, 11000011b, 11111111b, 11111000b, 11111111b
11100000b, 11100111b, 11100000b, 11100000b, 11001100b, 11100000b, 11001100b, 11000000b
11100000b, 11111111b, 11111111b, 11100000b, 11110000b, 11111111b, 11111000b, 11111111b
11100000b, 11100111b, 11100000b, 11100000b, 11001100b, 11100000b, 11001100b, 00000011b
11111111b, 11100111b, 11111111b, 11111111b, 11000011b, 11111111b, 11000111b, 11111111b
```

כשכל 1 יצג ריבוע של 3x3 של פיקסלים "צבועים", וכל 0 יצג ריבוע במימדים שווים אך "שקוף". הגדרתי שורה של מילה כ8 בייטים עוקבים, ומילה כ5 שורות עוקבות. אז, יצרתי פונקציב שעוברת על המילה ומציירת אותה על המסך.
מאחר והאפשרויות הן בכתב גבוהה יותר משאר המילים בתפריט, השתמשתי בשתי מילים אחת מעל השניה בכדי לעשות את האפשרויות.

זיהוי קלט והגבה אליו

בכל תחילת תור, המשחק מחכה ללחיצת עכבר מצד השחקן. לאחר קבלת הלחיצה, המשחק בודק האם השחקן לחץ במקום לא חוקי לבחירה: האם הוא בחר משבצת עם שחקן, והאם השחקן שבמשבצת הזו הוא שלו.

אם המשחק רואה שהמשבצת לא חוקית לבחירה, הוא מחכה לקלט חדש. אם הוא רואה שהקלט חוקי, הוא יסמן כנבחר את המקום שעליו לחץ השחקן, ואת המקומות אליהם הוא יכול ללכת יסמן כאופציות להליכה. כדי לסמן אותם, על המשחק להבין איזה של מי התור ובהתאם לכך הכיוון אליו השחקן הולך, ולבדוק האם יש כבר שחקן אחר באלכסון למקום הנבחר.
אם לא, מסמנים. אם כן, צריך לבדוק האם זה חייל של השחקן שהתור שלו, ואם זה כן, לא לסמן (כיוון שאי אפשר ללכת לשם). אם זה חייל של השחקן השני, יש לבדוק האם ניתן לאכול אותו. אם כן, התוכנה תסמן את המקום אליו ניתן לאכול.

סימון של משבצת כולל שני שלבים:
1. שינוי היצוג בזכרון של המשבצות הנבחרות על ידי הדלקת הביט הרביעי או החמישי (תלוי אם זה סימון אופצית הליכה או סימון בחירה).
2. ציור מחדש של המשבצות האלה כמסומנות.
לאחר שסומנו המקומות המתאימים, הם נשמרים בזכרון במשתנים שנועדו לכך, ואם אחד מהם הוא אפשרות לאכילה, גם הפרט הזה נשמר.

לאחר מכן המשחק מחכה לקלט נוסף מהמשתמש - לאן ללכת?
אם הפלט החדש מצביעה על משבצת שאינה אף אחת מהמשבצות המסומנות, הבחירה מתבטלת והמשחק חוזר לקבלת הקלט הראשון. אם הוא לחץ שנית על המשבצת שבחורה זה מכבר, לא קורה כלום. אם הוא לוחץ על משבצת של אופצית הליכה, המשחק יזיז את החייל לשם.
הכוונה בהזזת חייל היא הסרתו מהמשבצת הקודמת על ידי איפוס הביט השני בבייט שמייצג אותו, להוסיף אותו למשבצת החדשה על ידי הדלקת הביט השני ושינוי הביט השלישי בהתאם לצבע החייל, וציור מחדש של שניהם.
לאחר זה, המשחק מוריד את הסימון מהמשבצות שנבחרו קודם לכן, ומצייר אותן מחדש.

אם פעולת ההליכה היתה אכילה, נבדקת אפשרות של אכילה כפולה, והיא מוצעת לשחקן. לחיצה שלא על משבצת מסומנת תביא לסיום התור.

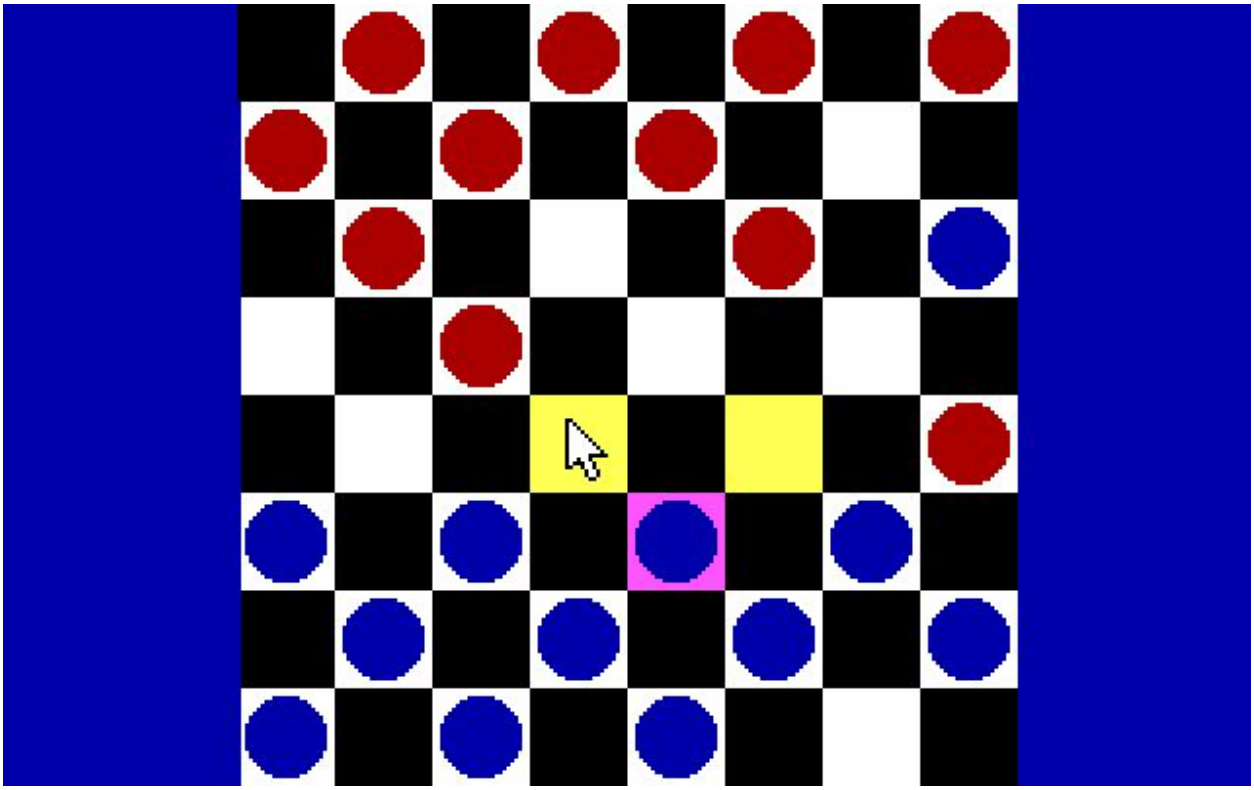
בינה מלאכותית (ב"מ) - AI (Artificial Intelligence)

יצירת הבינה המלאכותית היתה החלק המאתגר יותר מבחינה אלגוריתמית בפרויקט. בעצם הב"מ עובד באופן הבא:
ראשית, הב"מ בודק האם יש "מקרה חירום" שיש לטפל בו. מקרה חירום הוא מקרה בו אחד החיילים של הב"מ נמצא בסכנת אכילה מיידית (בתור הזה), וטיפול במקרה זה הוא בעדיפות עליונה. אם אכן יש כזה, הוא בודק מה המהלך הטוב ביותר שיוכל לעשות כדי למנוע את האכילה, ויבצע אותו.
אם אין, הב"מ עובר על כל המהלכים האפשריים שלו, ונותן ציון לכל אחד מהם, שעל איך הוא מחושב אכתוב בהמשך. במעברו על המהלכים, הוא שומר את המהלך בעל הציון הגבוהה ביותר, והוא זה שיבוצע לאחר סיום הלולאה.

ציון למהלך מחושב משילוב של הציון ההתקפי שניתן לו והציון ההגנתי שניתן לו. הציון ההגנתי מחושב כך:

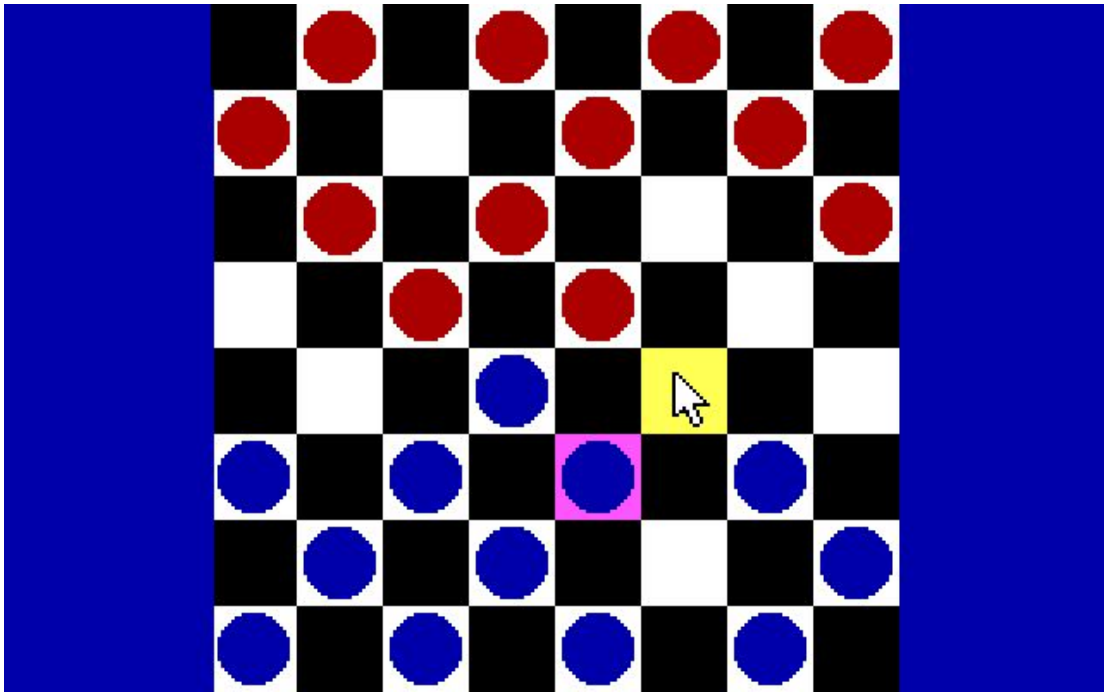
Defensive Score = 50 - (times being potentially eaten * 4) - (times causing other being eaten * 4)

הכוונה ב"times being potentially eaten": אם החייל זז למקום שבו השחקן השני יכול לאכול אותו בתור הבא, "times being potentially eaten" שווה למספר האכילות שהוא נחסף אליהן (בשל אכילות כפולות).
למשל, במצב הבא:



במצב של הליכה שמאלה times being potentially eaten שווה ל2, כיוון שזה מספר הפעמים שהשחקן האדום יוכל לאכול את הכחולים באכילה כפולה.

הכוונה ב"times causing other being eaten" היא דומה, אך מתייחסת למצב בו התזוזה גורמת לחשיפה של חייל אחר לאכילה.
למשל, במצב הבא:



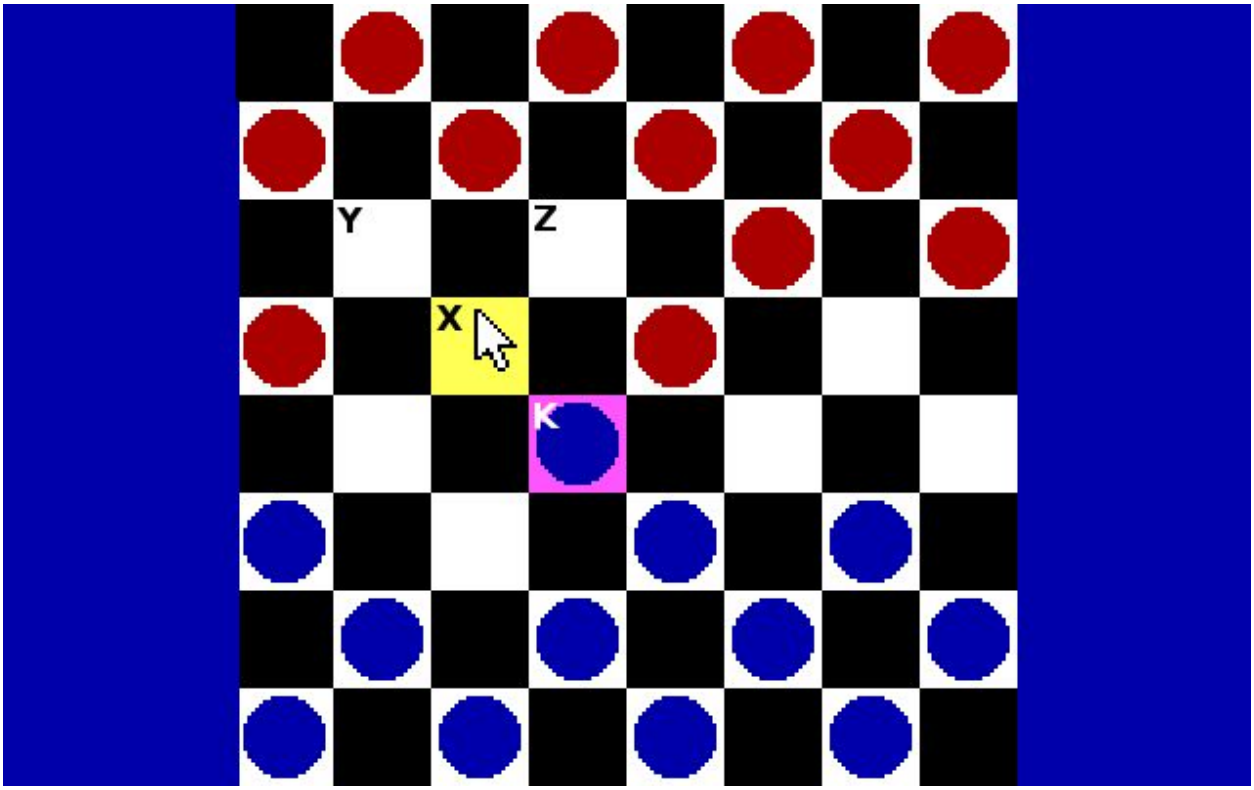
במהלך הזה times causing other being eaten שווה ל1, כיוון שהתזוזה תחסוף את החייל משמאל למעלה לאכילה אחת.

הציון ההתקפי מחושב כך:

Offensive Score = 50 + (times eating * 8)

כש"times eating" שווה למספר הפעמים שהחייל יכול לאוכל בתור הזה.

כדי להקנות לב"מ יכולת חשיבה קדימה, לציון הזה מוסף הציון של המהלכים הפוטנציאלית הבאים לחלק ל-2.
נניח שהמהלך שבשבילו אנחנו מחשבים את הציון הוא הליכה של חייל למשבצת X. אנחנו נחשב את ציון המהלך, ונוסיף לו את ציוני המהלכים שהשחקן יוכל לעשות תור הבא מהמשבצת X, מחולקים בשתיים.
למשל, אם מחושב הציון למהלך הבא:



אז זו תהיה התוצאה:

Score = Score(k→x) + Score(x→y)/2 + Score(x→z)/2

כלומר, ציון ההליכה מ-K ל-X, ועוד המהלכים שנוכל לעשות מ-X בתור הבא, לחלק ל-2. גם החישוב של ההליכה מ-X ל-Z תבדוק את ציוני המהלכים האפשריים מ-Z, ותוסיף אותם מחולקים לשתיים לציון. החלוקה ל-2 נועדה כדי שלמהלך המיידי תהיה יותר השפעה על הציון מהמהלכים שבתורות הבאים.

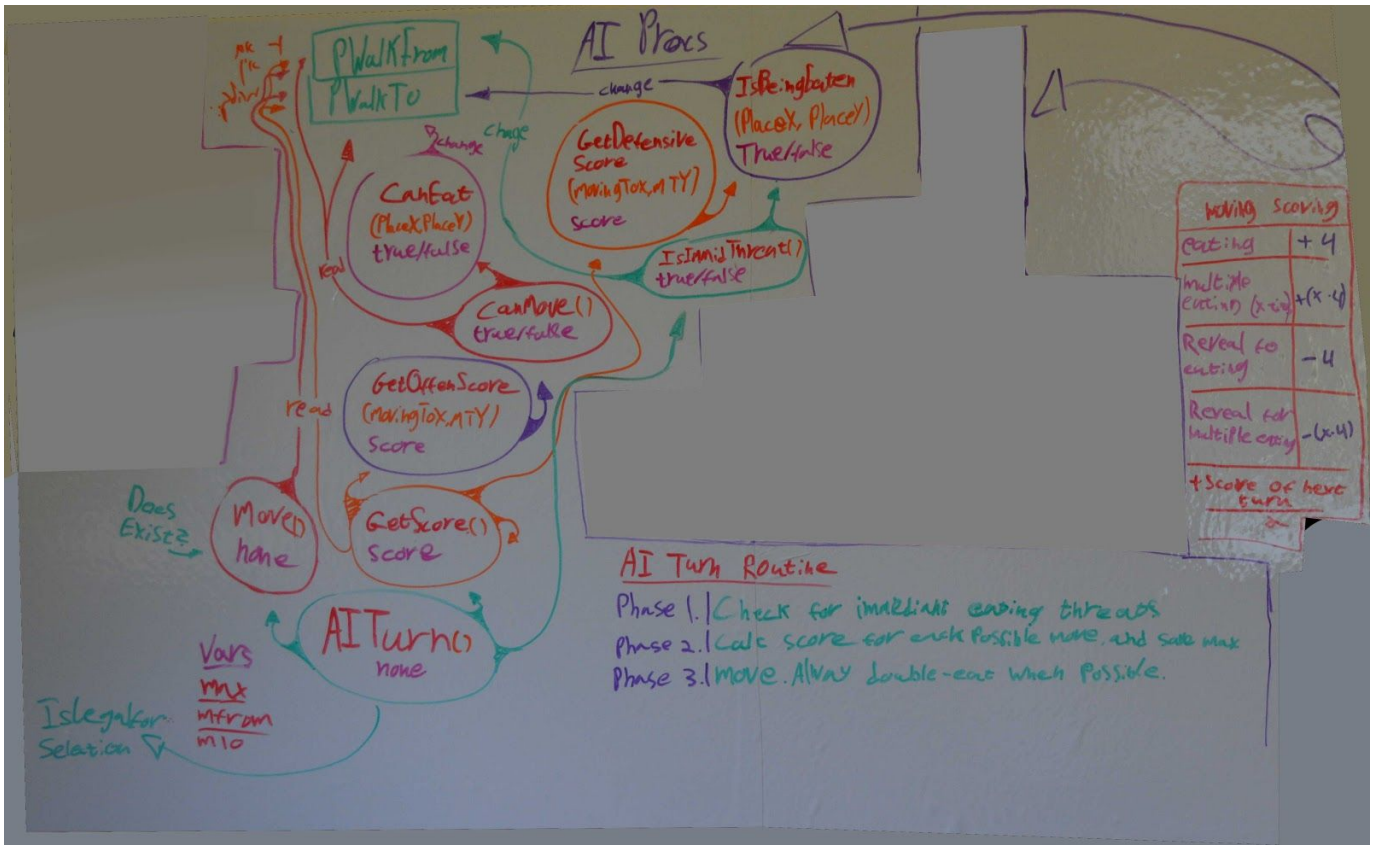
בעקרון הב"מ בודק שלוש מהלכים קדימה, אך כאן אין אפשרויות תזוזה לאחר המהלך השני.

הבעיה בזה היא שקיימת סכנה לרקורסיה אין סופית - אם כל ציון מהלך קורא לבדיקת ציון המהלך הבא, זה עלול לעולם לא להגמר. כדי למנוע את זה, החלטתי שאני מגביל את הציון לשלושה מהלכים קדימה.

לאחר שהבנו את זה, יהיה אפשר להגדיר את מתן הציון כך:

thisTurnsScore + nextTurn’sScore / 2 + nextTurn’sNextTurn’sScore / 4

כשהיה לי את האלגוריתם הראשוני בראש, ציירתי לעצמי flowchart של איך הוא יראה מבחינת פרוצדורות:



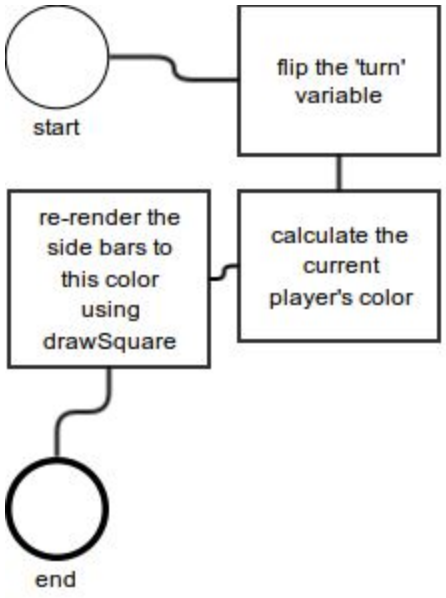
מצד שמאל ניתן לראות את תרשים הזרימה של הפרוצדורות. השם של הפרוצדורות באדום, הפרמטרים שהן מקבלות בכתום ומה הן מחזירות בסגול. אמנם כמה פרטים השתנו מאז, ובפועל הוספו עוד כמה פרוצדורות, אך העקרון נשאר דומה.

תרשימי זרימה

בחלק הזה, אציג תרשימי זרימה של הפרוצדורות המרכזיות בתוכנית. לא אציג את כל הפרוצדורות, אך אציג את המשמעותיות ביותר ואת אלו שמייצגות את דרך הפעולה של אלו הדומות לה.

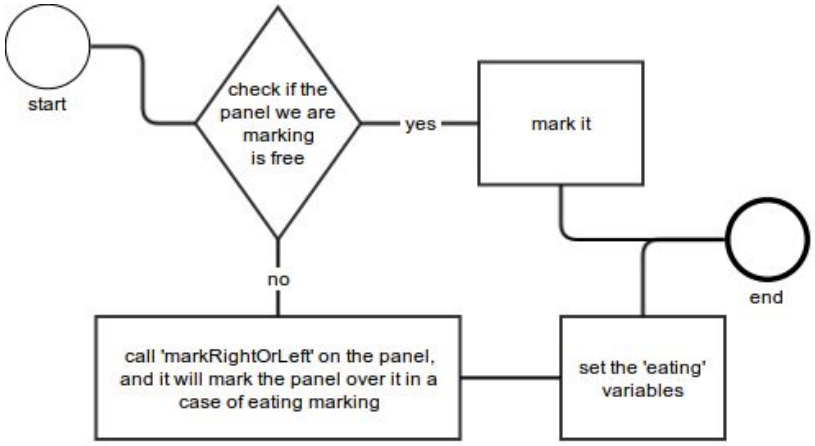
Main.asm

nextTurnSet

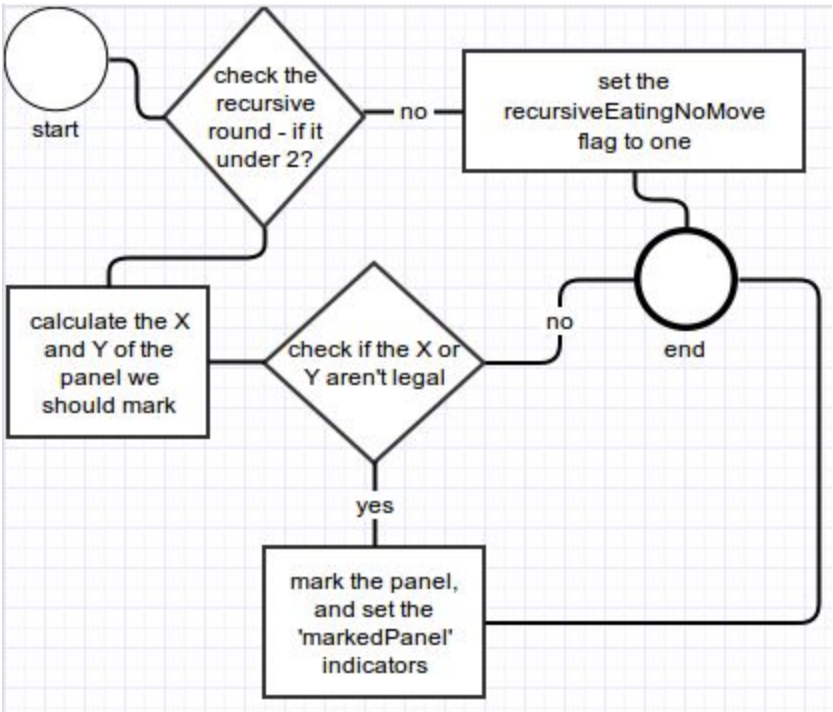


Select.asm

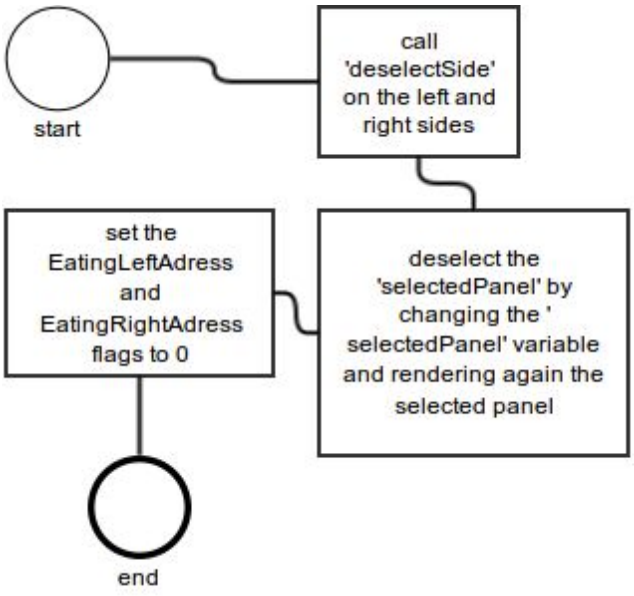
Mark



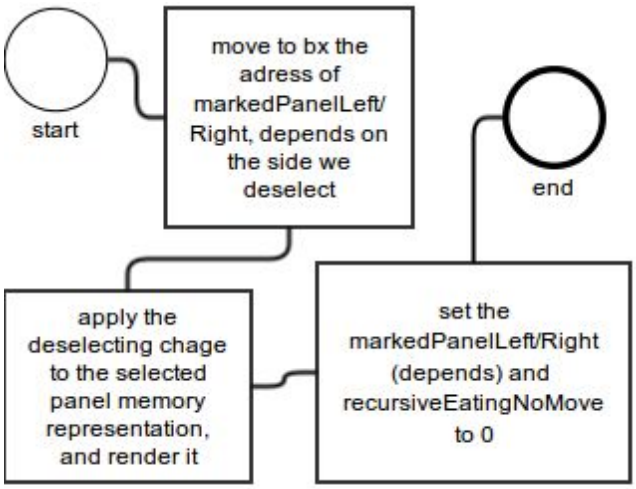
markRightOrLeft



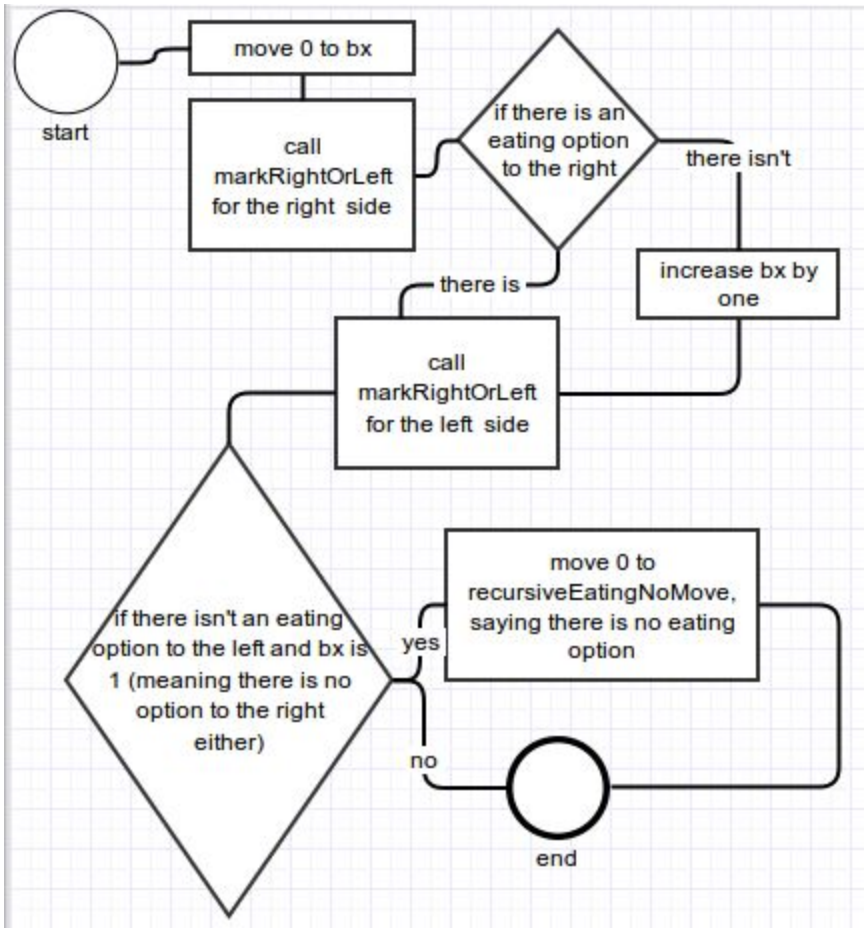
deselect



deselectSide

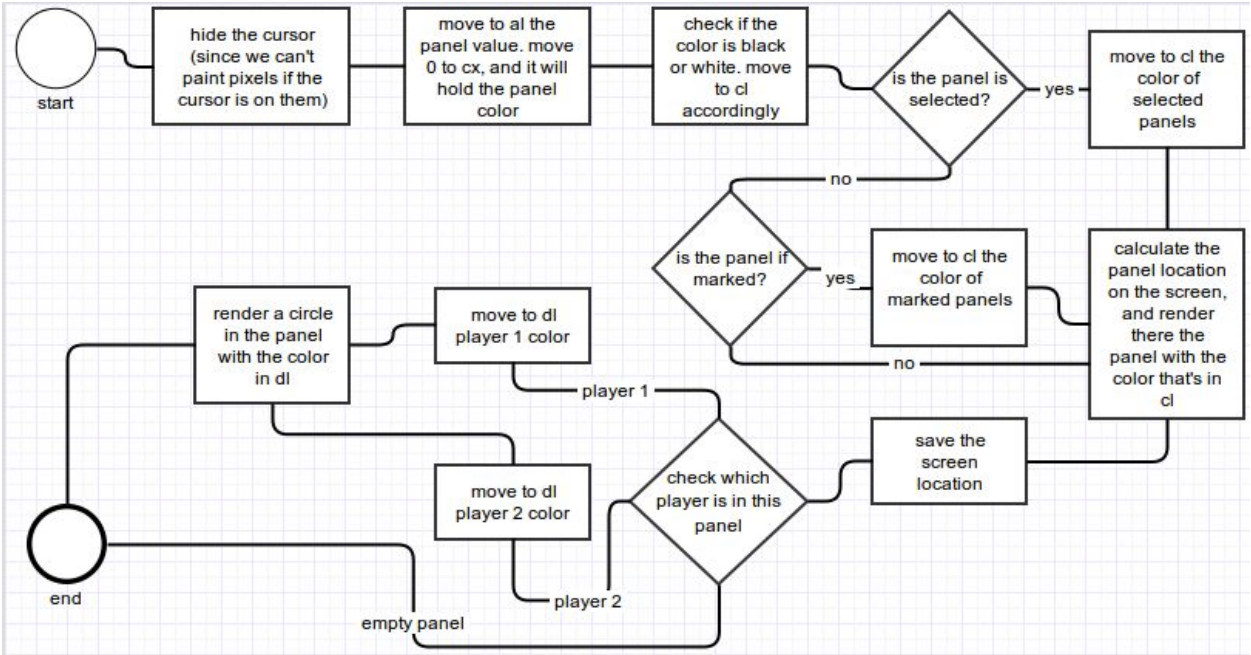


markMovingPossiblities



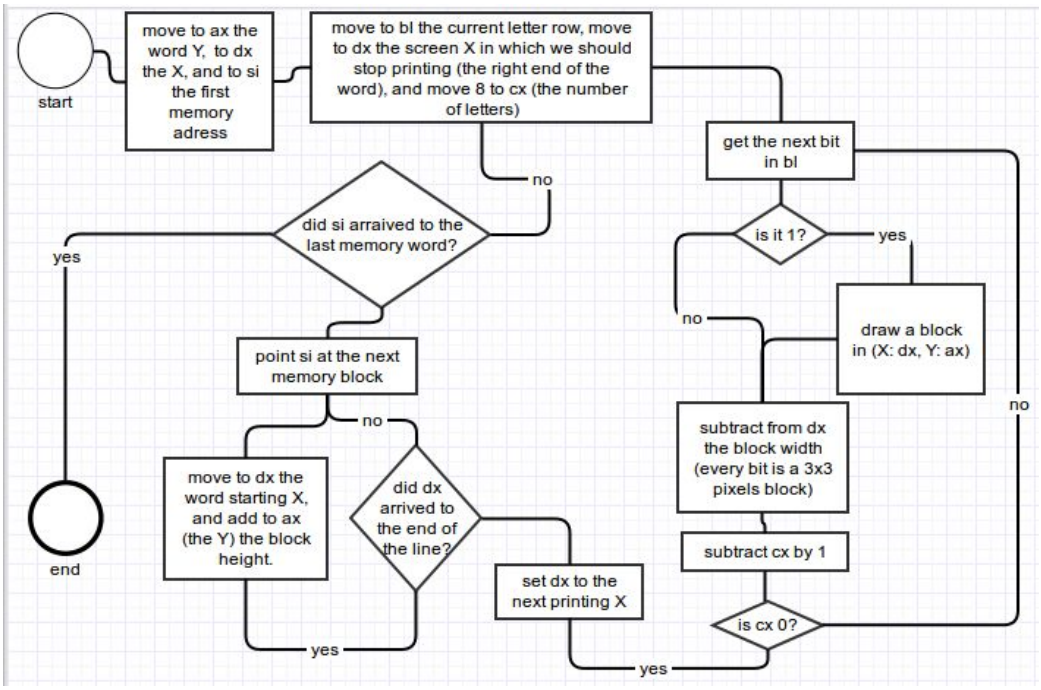
Renderer.asm

renderPanel

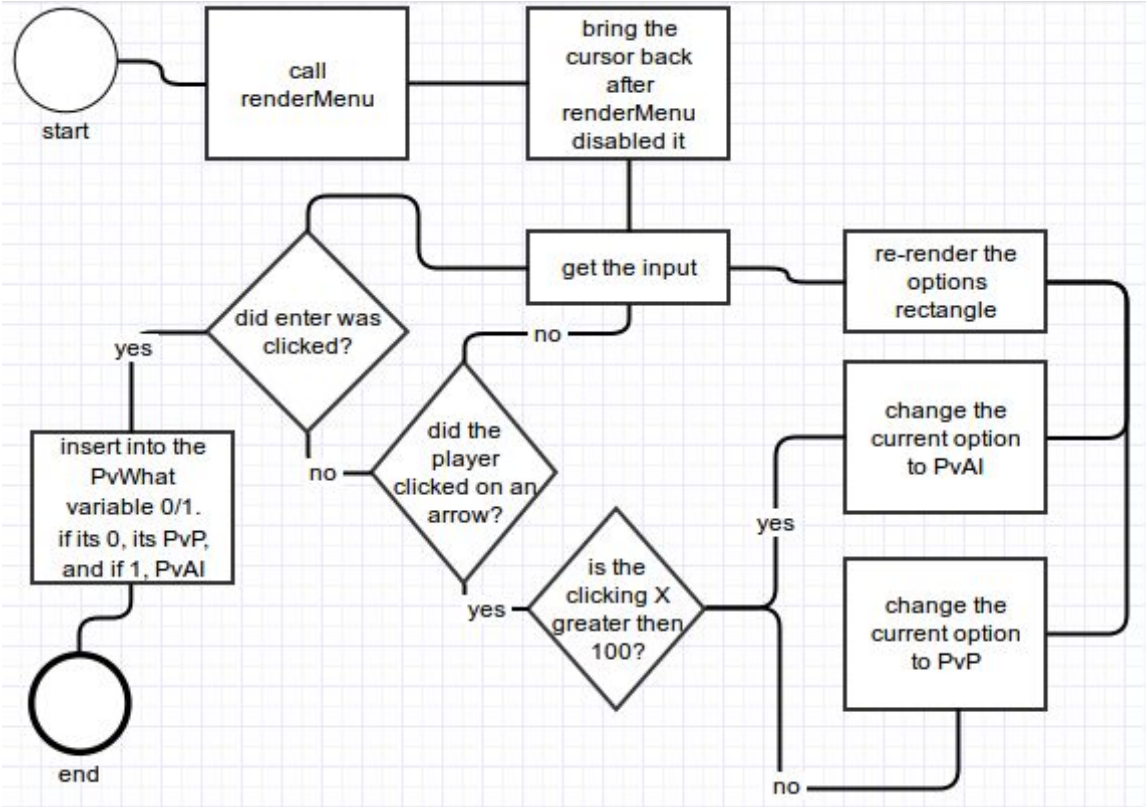


Menu.asm

renderWord

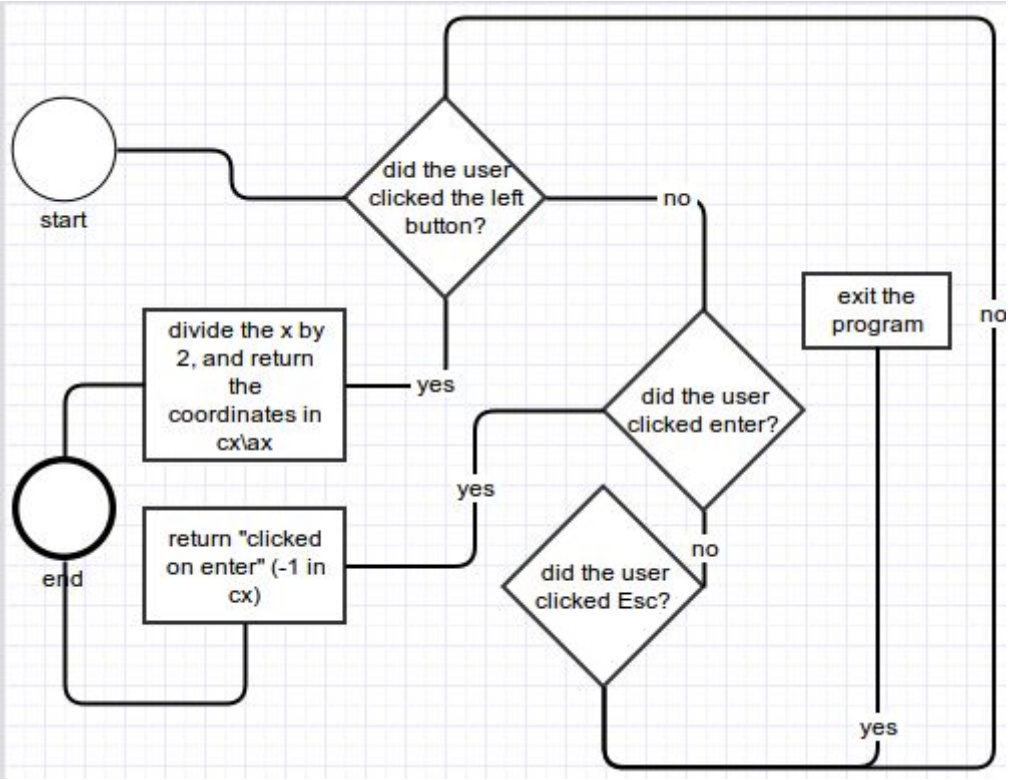


startMenu

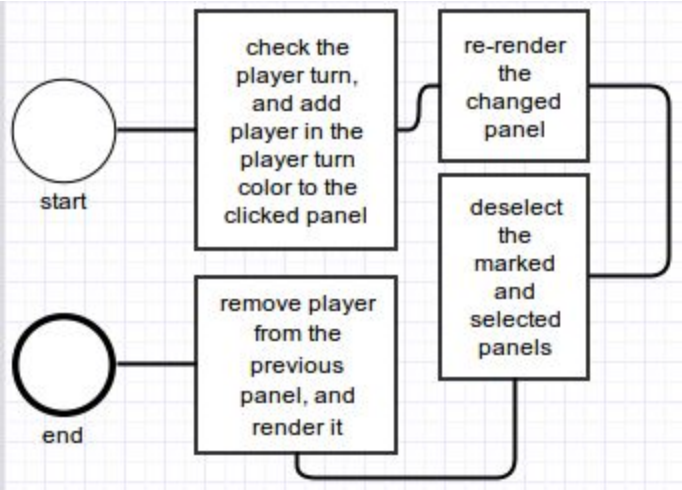

start
call renderMenu
bring the cursor back after renderMenu disabled it
get the input
re-render the options rectangle
did enter was clicked?
yes
no
insert into the PvWhat variable 0/1. if its 0, its PvP, and if 1, PvAI
end
did the player clicked on an arrow?
no
yes
is the clicking X greater then 100?
yes
no
change the current option to PvAI
change the current option to PvP

Logic.asm

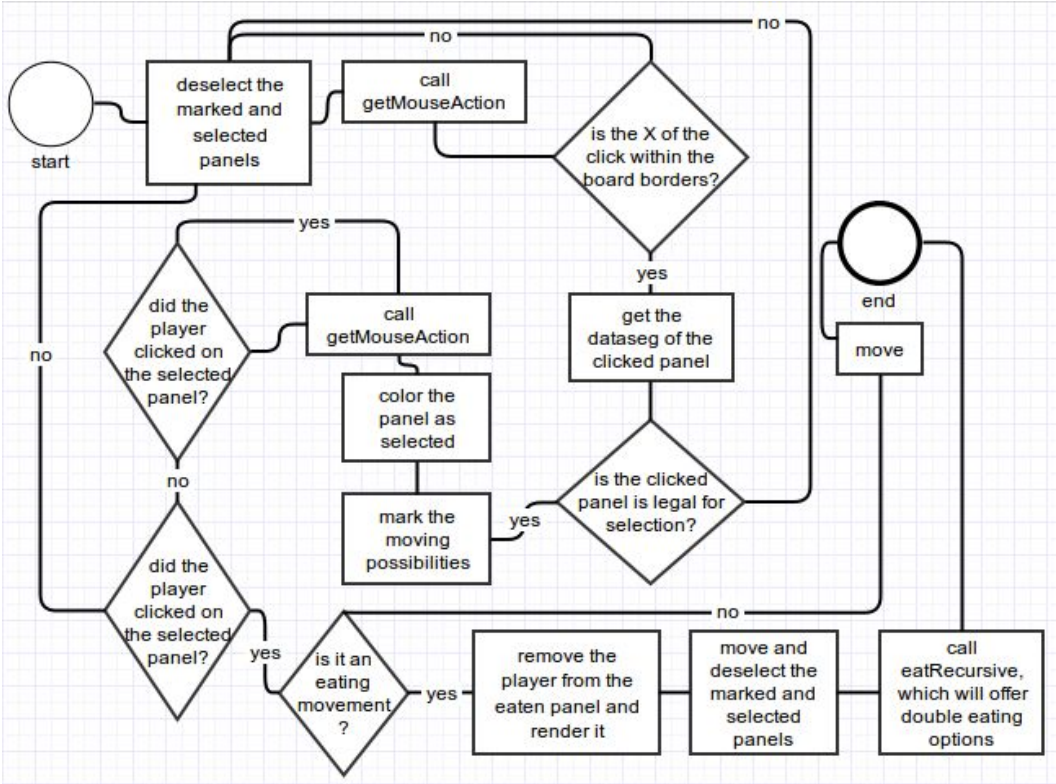
getMouseAction



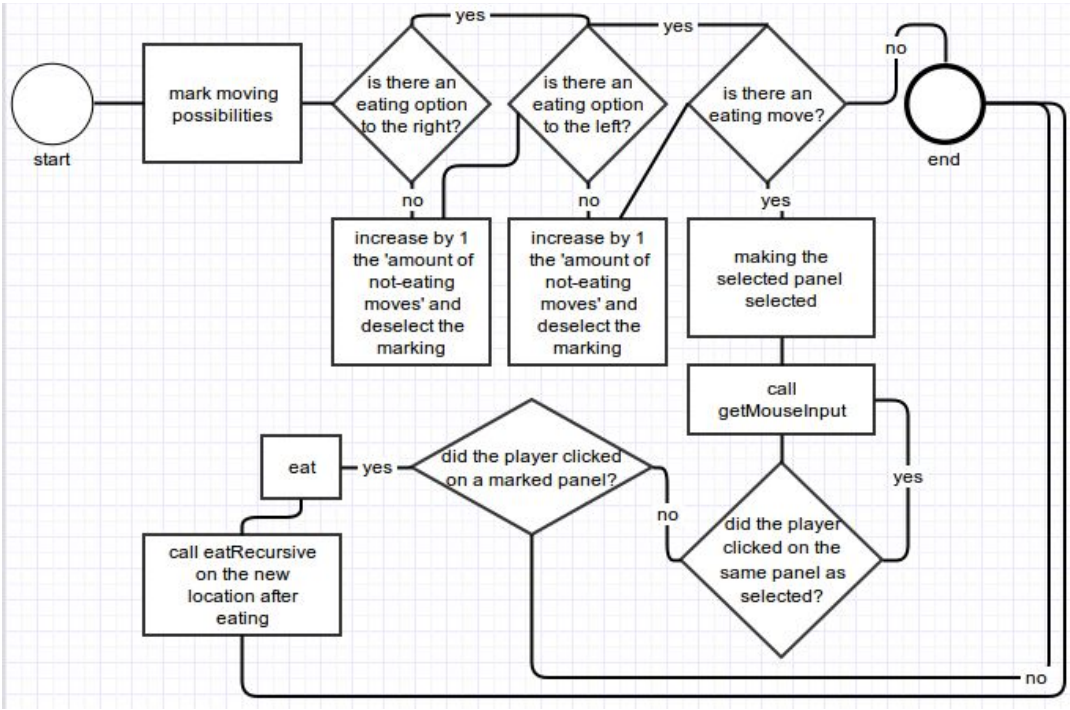
move



nextTurn

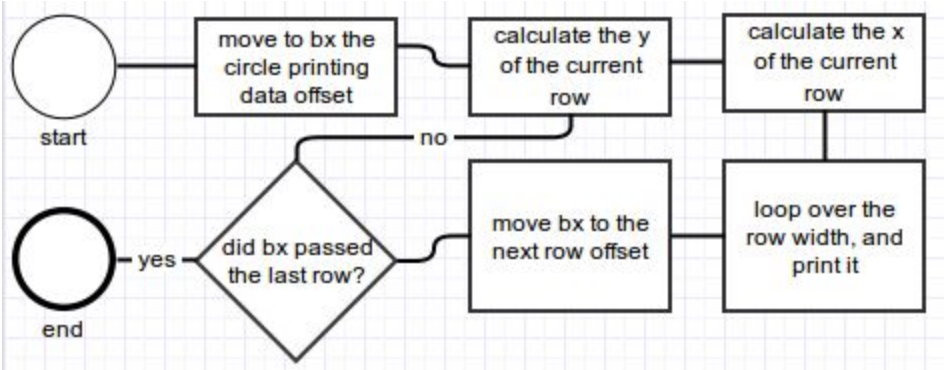


eatRecursive



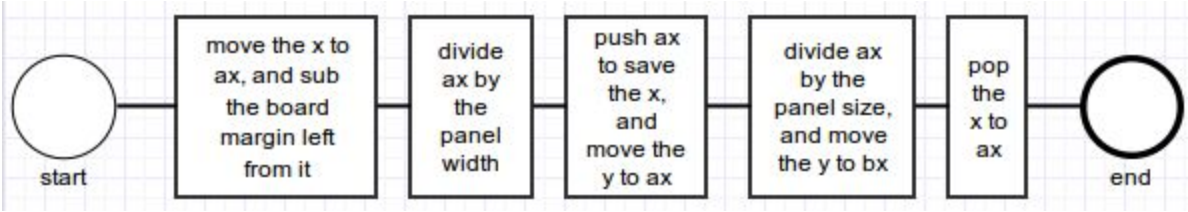
Graphics.asm

drawCircle



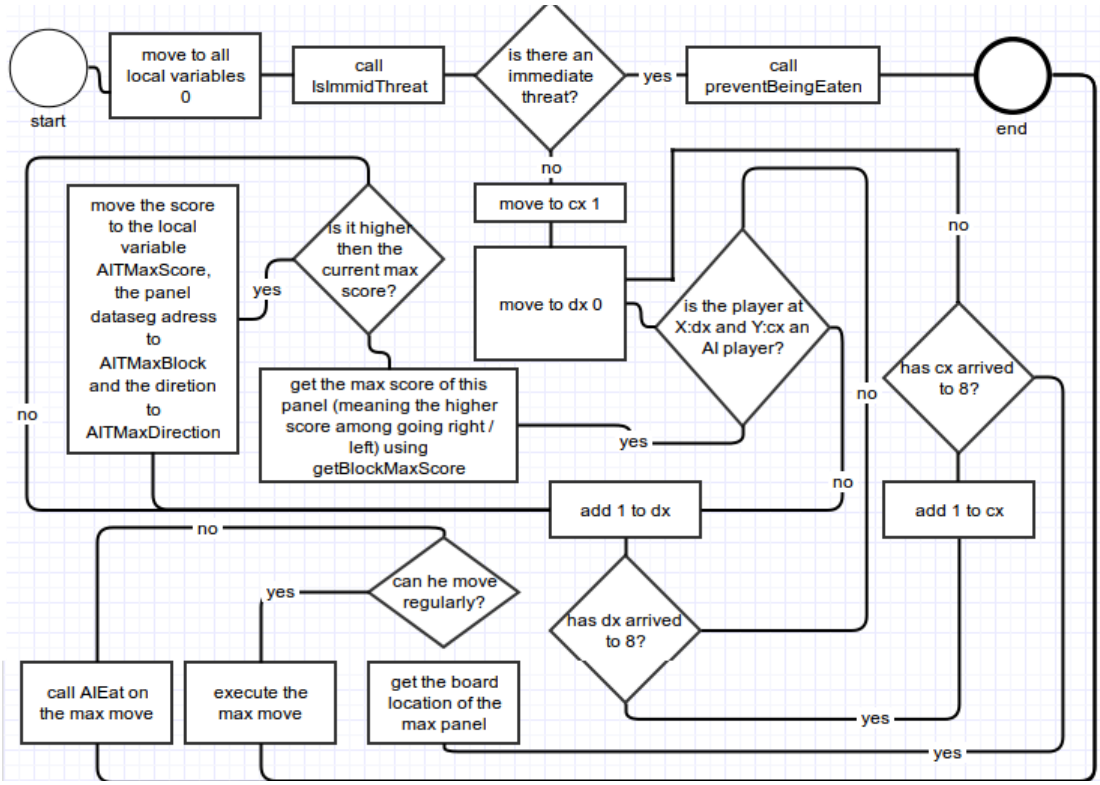
Convert.asm

pixelLocationToBoardLocation

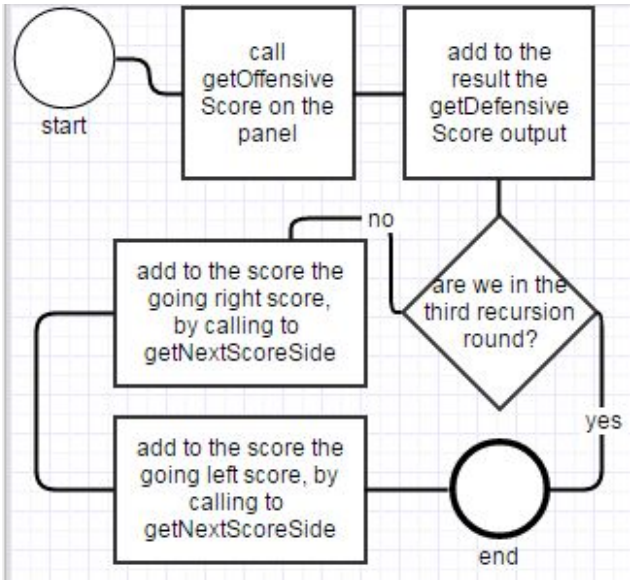


AI.asm

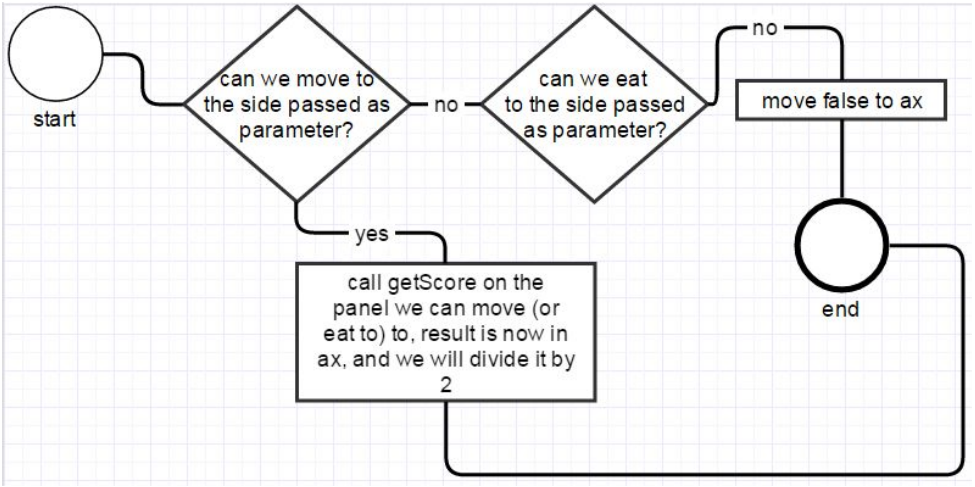
AITurn



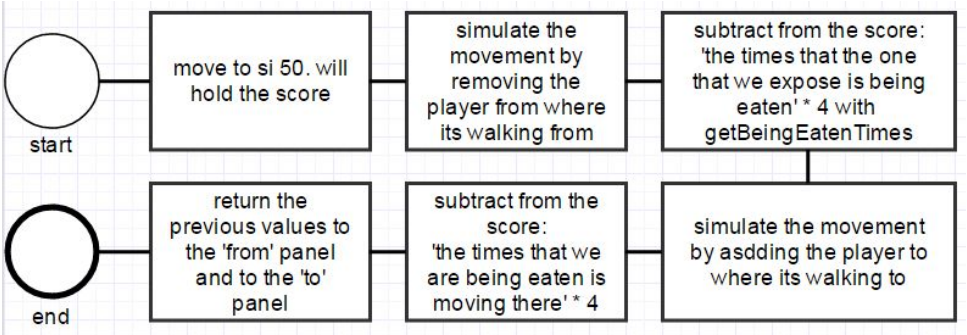
getScore



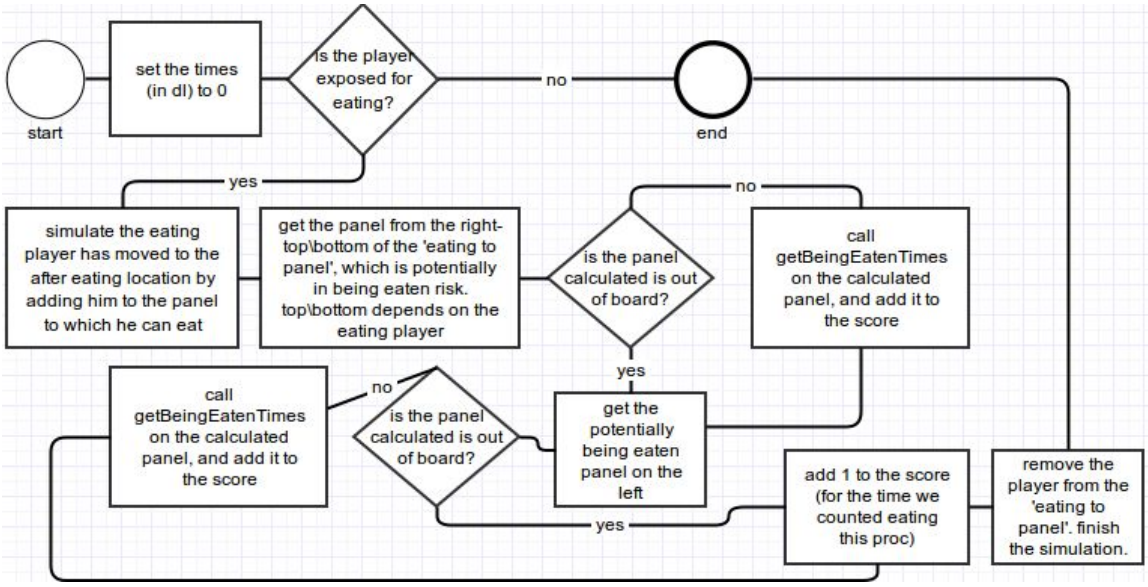
getNextScoreSide



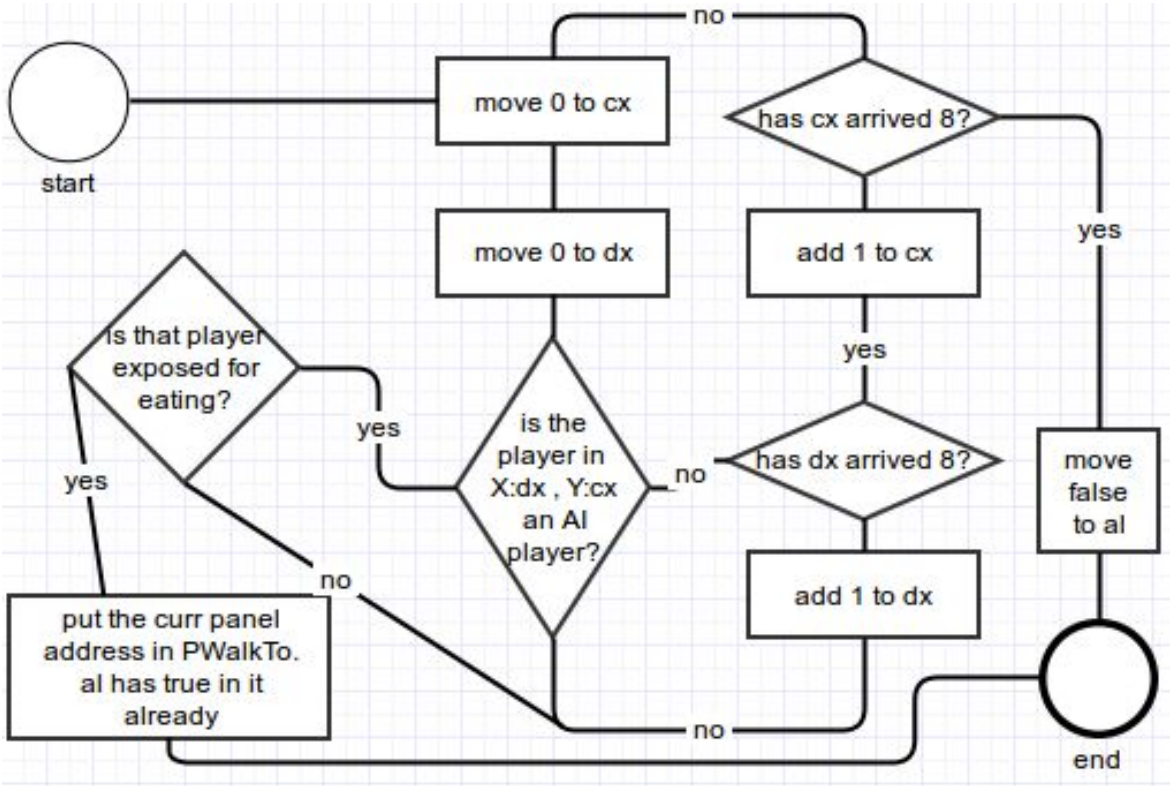
getDefensiveScore



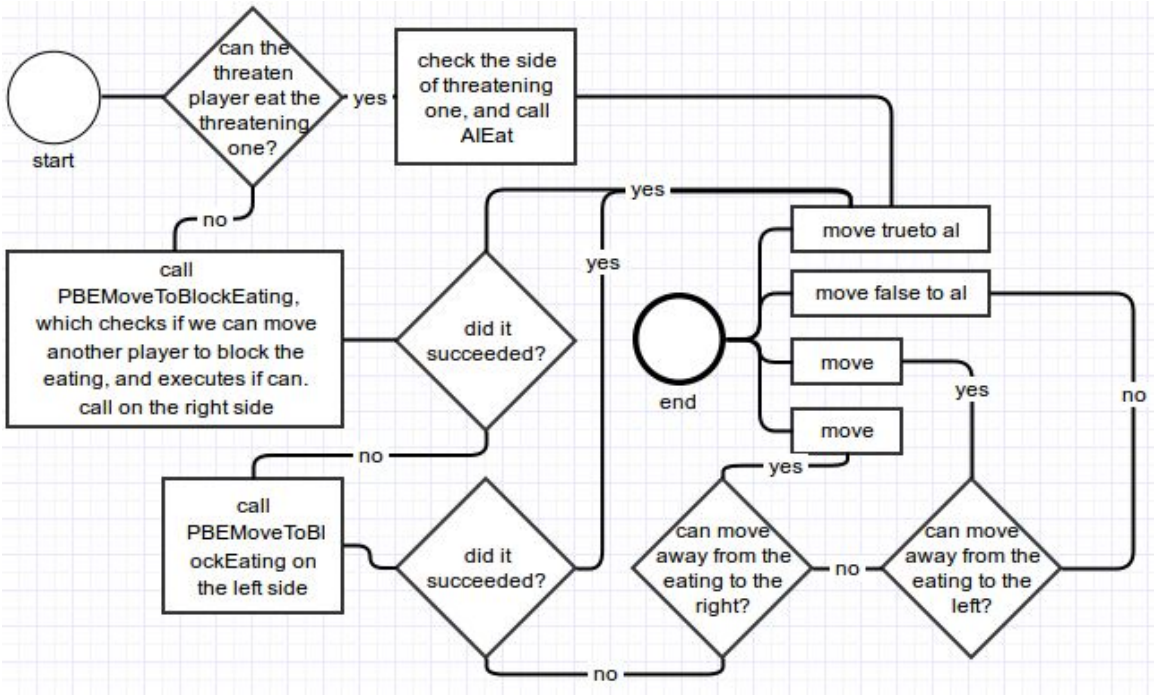
getBeingEatenTimes



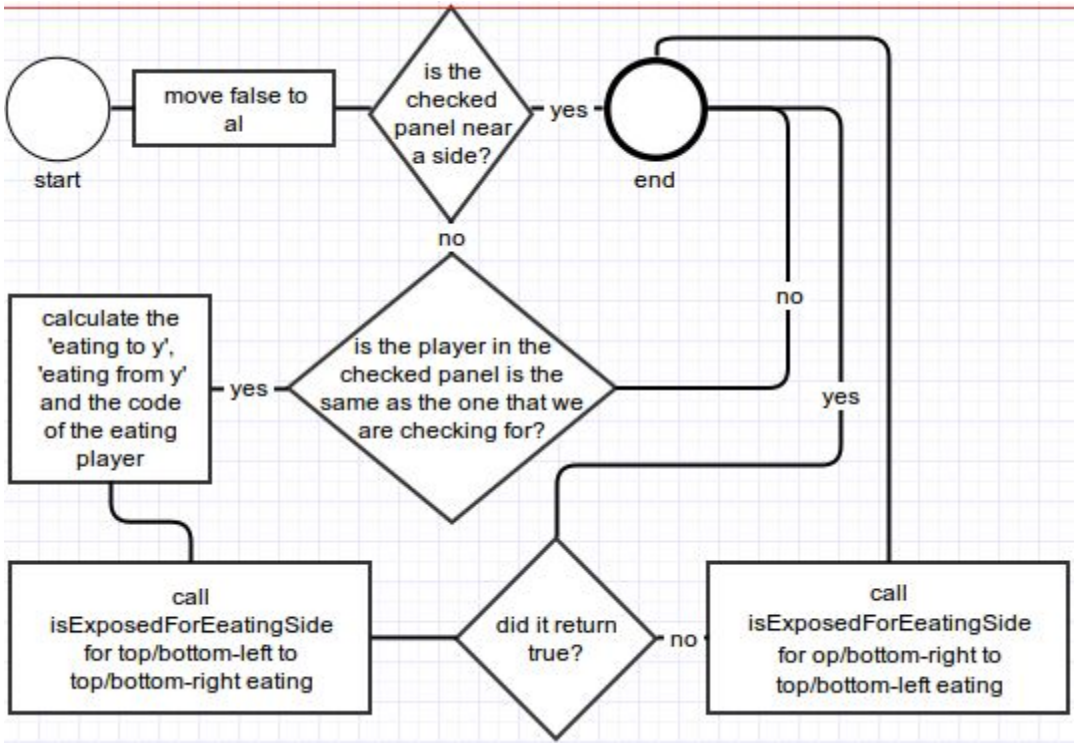
isImmidThreat



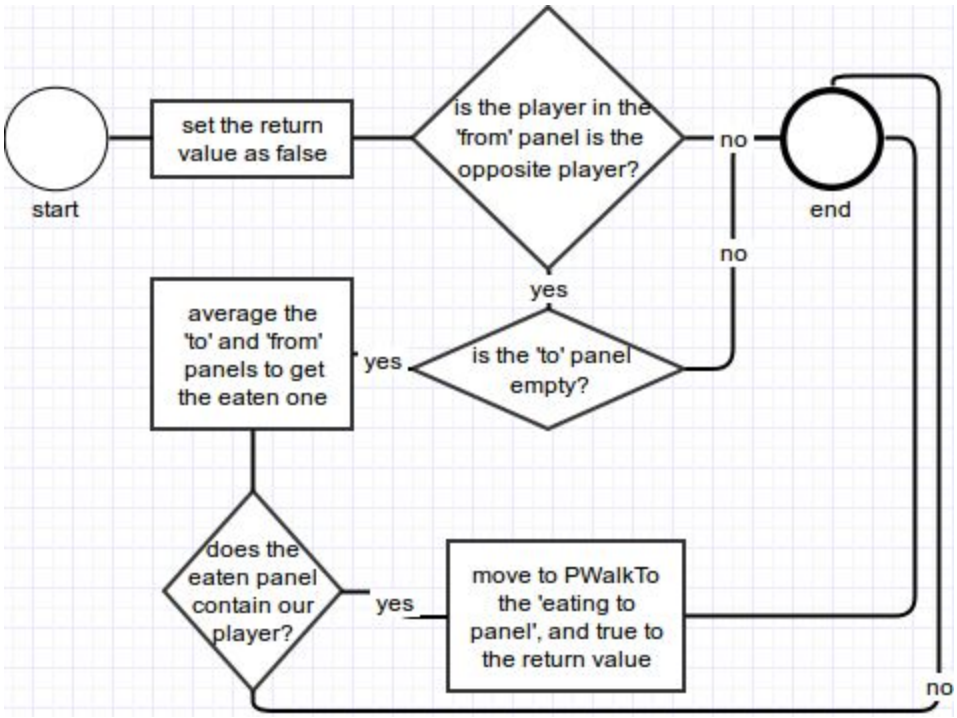
preventBeingEaten



isExposedForEating



isExposedForEatingSide



תיעוד הפרוצדורות

בחלק זה ארשום את כל פרוצדורות התוכנית, כשלכל אחת שם, טענת כניסה, טענת יציאה והערות לגבי תפקודה (אם הוא לא ברור).

select.asm

Procedure name	Procedure arguments	Procedure return value	Notes
markMovingPossiblities	none	void	
markRightOrLeft	whichDirection(0 for right mark 1 for left), table x, table y	void	
getMarkY	selected table y	the y in ax, or 9 if shouldn't mark	
mark	whichDirection(0 for right mark 1 for left),table x, table y	void	
setMarkingIndicators	marking side, dataseg address	void	
isLegalForSelection	dataseg location	true or false in ax	
deselect	none	void	
deselectSide	none	whichDirection(0 for right mark 1 for left)	

renderer.asm

Procedure name	Procedure arguments	Procedure return value	Notes
renderBoard	none	void	
renderPanel	the panel dataseg location (board location...)	void	

menu.asm

Procedure name	Procedure arguments	Procedure return value	Notes
startMenu	none	void	
renderMenu	none	void	
renderWord	color, offset, pixels-x, pixels-y	void	
renderAsGameMode	the option (0 for player, 1 for AI)	void	

main.asm

Procedure name	Procedure arguments	Procedure return value	Notes
nextTurnSet	none	void	
didGameOver	none	true/false in al	a game is over when there are no more possible moves / no more players from a specific side

logic.asm

Procedure name	Procedure arguments	Procedure return value	Notes
nextTurn	none	void	
eatRecursive	none	void	
getEatenPanel	the moving-to panel address	the eaten panel address in bx, or false if not eaten	
move	dataseg address of the clicked panel	void	
isMarkedPanel	dataseg address	true / false in ax. Whether this panel is marked	
getMouseAction	none	the mouse click X in cx,	If clicked enter, will

		and mouse click Y in dx	return -1 in cx. if clicked esc, will exit the program
getPlayer	the panel dataseg location	the player code in ax (PLAYER_1_CODE, PLAYER_2_CODE, NO_PLAYER_CODE).	
finishProgram	none	void	

graphics.asm

Procedure name	Procedure arguments	Procedure return value	Notes
drawTriangle	color, facingDirection(0 for right mark 1 for left), pixels-x, pixels-y	void	the height and width are 15
drawCircle	pixels-x, pixels-y, color	void	
drawSquare	height, width,pixels-x, pixels-y, color	void	
drawPixel	pixels-x, pixels-y, color	void	

convert.asm

Procedure name	Procedure arguments	Procedure return value	Notes
datasegLocationToPixel Location	dataseg location	the x in ax, and the y in bx.	working according to this formula: datasegLocation = the location in the dataseg tableY = datasegLocation / 8 tableX = datasegLocation - tableY*8 pixelsX = tableX * panelSize + marginLeft

			pixelsY = tableY * panelSize where the table locations are cells numbers - first cell is 0,0 and last cell is 7,7
tableLocationToPixelLocation	the tableX and tableY	the pixel-x in ax, and the pixel-y in bx.	
datasegLocationToTableLocation	dataseg location	the table x in cx, and the table y in ax.	
pixelLocationToBoardLocation	pixelX,pixelY	the board x in ax, and board y in bx	
boardLocationToDatasegLocation	boardX,boardY	the dataseg location in bx	It should be called tableLocationToDatasegLocation
pixelLocationToDatasegLocation	pixelX,pixelY	the dataseg location in bx	

AI.asm

Procedure name	Procedure arguments	Procedure return value	Notes
AITurn	none	void	
getBlockMaxScore	block x, block y	the score of the better side in ax, side direction in bx	
getScore	recursiveRound(will stop at 1), fromX, fromY, toX, toY	score in dx	returns the score of this move + next turn move / 2 + next next turn move / 4
getNextScoreSide	side(0 for right, 1 for left), recursiveRound(not decreased, GNSS will dec)(will stop at 1), toX, toY	score of the side in ax, already divided by 2	

getOffensiveScore	movingToX, movingToY	score in dl, when default 50 and as higher, as better	
getDefensiveScore	movingFromX, movingFromY, movingToX, movingToY	score in dl, when default 50, as higher, as better	
AIEat	eatingPanelDatasegAdress, direction(0 for right, 1 for left)	void	
getBeingEatenTimes	beingEatenPlayerCode, panelX, panelY	times in dl	
IsImmediateThreat	none	true/false in al, and where the threat is in PWalkTo	
preventBeingEaten	where the threat is in PWalkTo	true/false in al, saying whether we acted	
PBEMoveToBlockEating	datasegment address of the threatened panel, side from which to check movement (0:right of the panel, 1:left of it)	true/false in al, representing whether we moved or not	
IsExposedForEating	whoWeAreCheckingFor(the player that we are checking if can be eaten, PLAYER_1_CODE / AI_CODE), panelX, panelY	true/false in al, and to where in PWalkTo	
IsExposedForEatingSide	badGuysCode(PLAYER_1_CODE/AI_CODE), eatingToX, eatingToY, fromX, fromY	true/false in al, and to where in PWalkTo	
canEat	panelX, panelY	true/false in al, and to where in PWalkTo	
canMove	panelX, panelY, toLeftOrRight(0 for right mark 1 for left)	true/false in al, and to where in PWalkTo	

קוד מקור

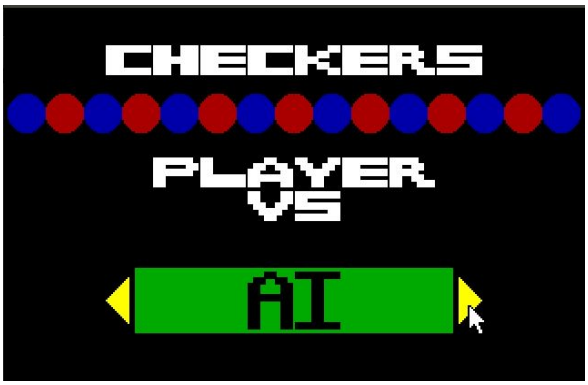
select.asm

```
; markMovingPossiblities PROCEDURE HEADER
;     parameters: none
;     return: void

proc markMovingPossiblities
  push cx
  push ax
  push bx

  push [selectedPanel]
  call datasegLocationToTableLocation ; tableX => cx, tableY => ax

  xor bx, bx ; will store 2 if two directions have no eating
option (not exactly, but thats the concept. I'll increase bx after
the
           ; first check (if we should), and if its 1 after the
second check and we should increase again, but I won't bother
increasing.)
           ; in short: only if have no eating option to both sides,
put 1 in recursiveEatingNoMove

  mov byte ptr [recursiveEatingNoMove],0
  push 0
  push cx ; table x
  push ax ; table y
  call markRightOrLeft

  cmp byte ptr [recursiveEatingNoMove], 1
     jne MMPDontInc
     inc bx
  MMPDontInc:

  mov byte ptr [recursiveEatingNoMove],0
  push 1
  push cx ; table x
  push ax ; table y
  call markRightOrLeft

  cmp byte ptr [recursiveEatingNoMove], 1
     jne MMPHaveEatingOptions
```

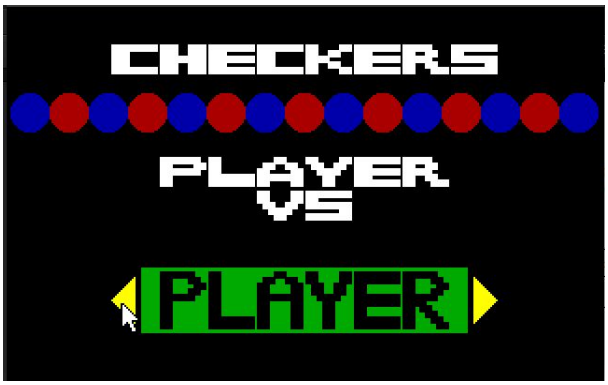
דוגמאות הרצה

תפריט

שחקן נגד בינה מלאכותית

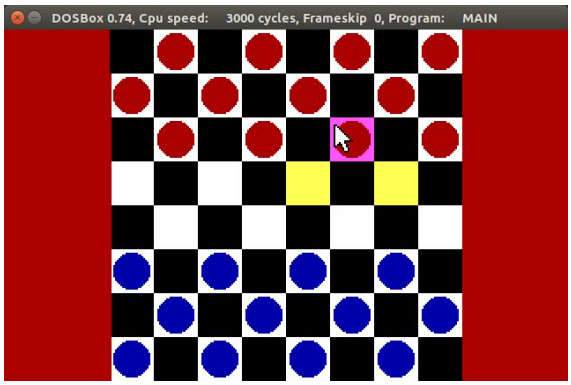


שחקן נגד שחקן

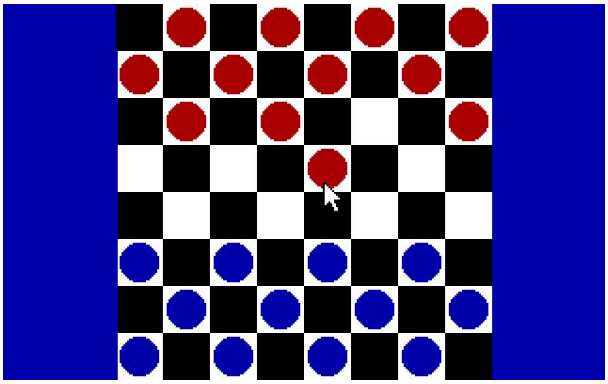


תזוזה

בחירת שחקן להליכה

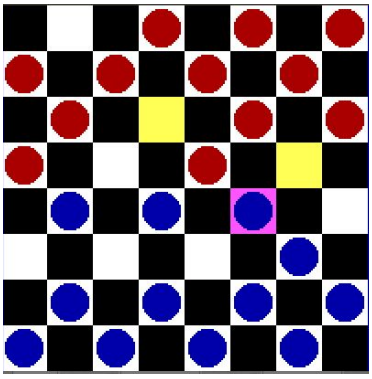


בחירת משבצת אליהה רוצים ללכת

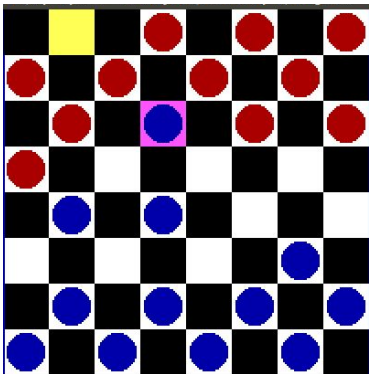


אכילה

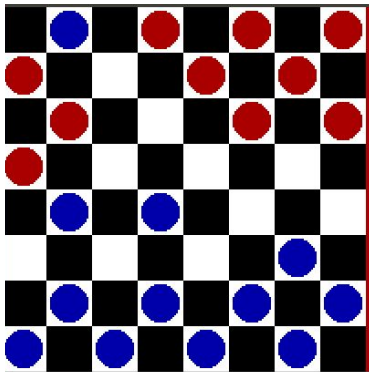
1: הכחול יכול לאכול את האדום



2: לאחר שהכחול אכל, יש לו אפשרות לאכילה כפולה

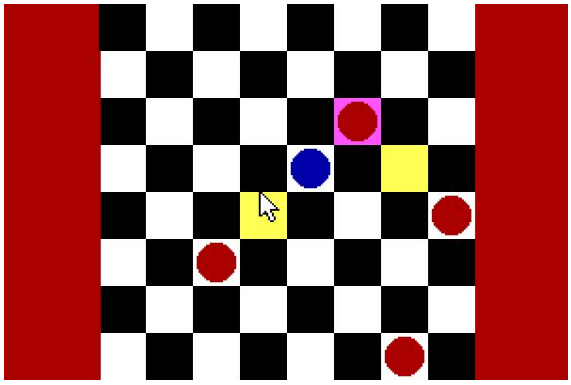


3: הוא מימש אותה, ומאחר ואין לו עוד אפשרות אכילה, הועבר התור



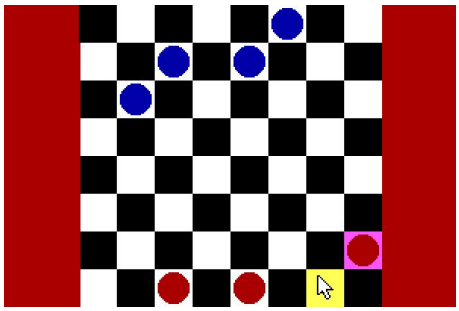
סיום משחק

נצחון של האדום



סוף המשחק

אין מהלכים אפשריים לאדום



סוף במשחק

סיכום אישי

פרויקט הדמקה היה הפרויקט הגדול ביותר שעבדתי עליו עד כה בעצמי, ולמדתי ממנו רבות. ראשית, ככל שהתקדמתי בפרויקט נוכחתי לגלות את החשיבות שבתכנון מראש ותיעוד, וככל שהתקדמתי גם יותר ויותר מימשתי את אלו.
בנוסף, היתרונות שבתכנות פרוצדורלי נכון ובreusable code בלטו במיוחד עם התקדמותי, ועזרו לי בפרויקט במיוחד, מה שעזר לי להבין את משמעותם.

כיוון שהצטרפתי למגמה באמצע השנה, היה עלי לחזור במהרה על כל החומר ולעשות את הפרויקט, מה שהיווה אתגר נוסף.
מעבר לאתגר התכנות, גם האתגר האלגוריתמי - שבב"מ ובמשחק בכלל - היה אתגר שנהנתי להתעסק בו, ושתרם לי רבות אף הוא.
העובדה שהשפה בה אני מתכנת היא אסמבלי בלטה תחילה, אך ככל שהתקדמתי בפרויקט והתרגלתי אליה, השפה ומגבלותיה הפכו לאתגר משני וזניח בשבילי, ונהיה לי קל לפתור בה בעיות כמעט כמו בשפה עילית.

אני מרגיש שמהפרויקט הזה אני יוצא עם יותר תובנות והבנה יותר טובה של קונספטים שתוארו לעיל ואחרים, וכמובן עם הנסיון הרב שרכשתי במהלכו.