פרוייקט באסמבלי לסייבר

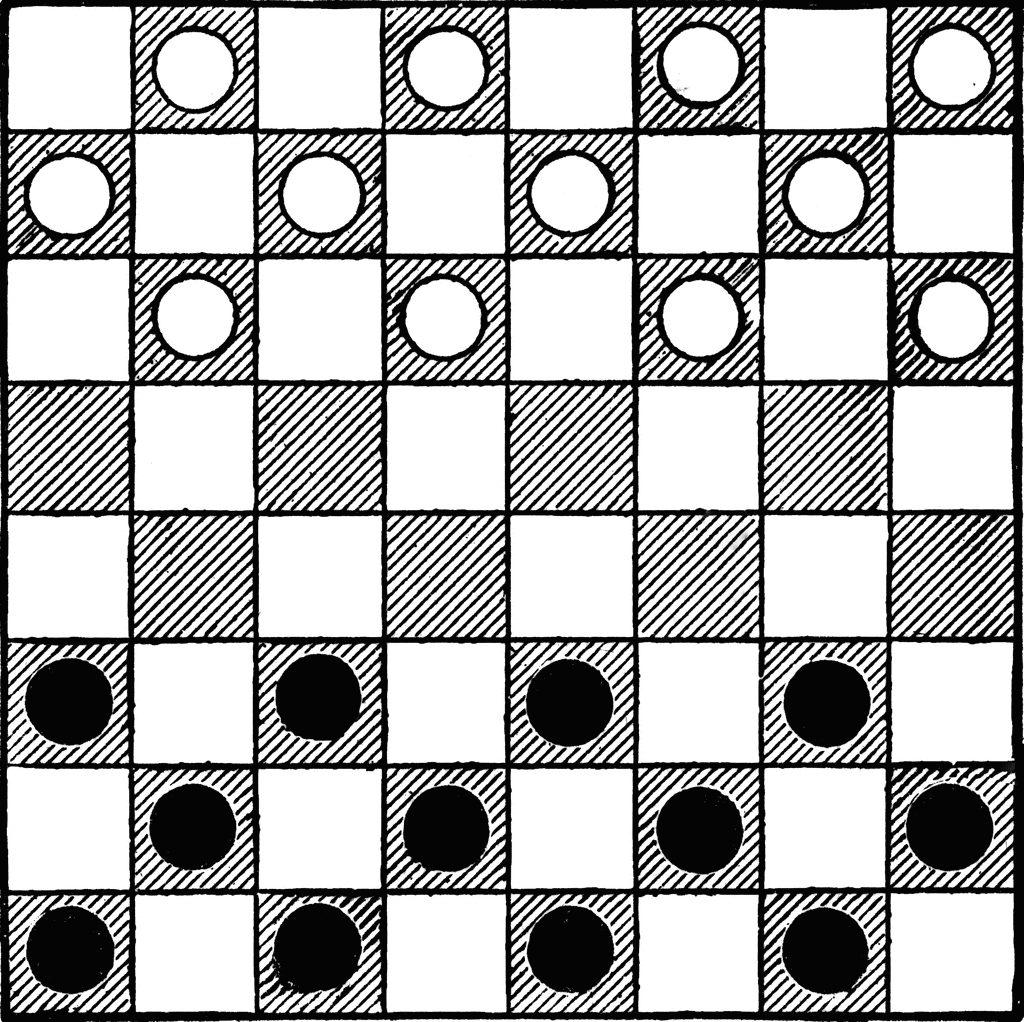
מאת: אסף ארבל י'4

נושא: משחק דמקה

בית ספר: תיכון הדרים, הוד השרון

תאריך הגשה: 29.6.2016

דמקה



מידע על המשחק: דמקה

דמקה הינו משחק אסטרטגיה לשני שחקנים. לוח המשחק מורכב מ 24 חיילי משחק, 12 לבנים ו12 שחורים.

במהלך המשחק, יש תורות בהם על השחקן שתורו להזיז את אחד מחייליו.

כל חייל יכול לזוז באלכסון עם כיוון תנועת הצבע (שחור-למעלה, לבן-למטה), ואם נחסם על ידי חייל אויב הוא יכול לקפוץ מעליו ובכך להעלים את החייל של היריב מהמשחק. מהלך זה נקרא "אכילה" והוא מזכה את השחקן שאכל בנקודה.

המשחק נגמר כאשר אחד השחקנים מגיע ל12 נקודות והוא מוכרז כמנצח.

הערה: אם חייל כלשהו מגיע לסוף הלוח הוא אינו יכול להתקדם. קחו זאת בחשבון באסטרטגיה שתכינו.

סביבת העבודה: TASM

מדוע בחרתי בנושא זה לפרויקט

בחרתי במשחק הדמקה לפרויקט שלי משום שאני מאוד אוהב את המשחק וחשבתי שזה יהיה לי אתגר לתכנת אותו בשפת אסמבלי. במהלך העבודה נתקלתי בהרבה מכשולים, כמו שימוש בעכבר, צביעת ריבועים, קריאה לפרוצדורות, שימוש באוגר BP ועוד רבים. כדי להתגבר על אותם מכשולים נעזרתי בספר הלימוד שצוות "גבהים" כתב עבור תלמידי הקורס. הספר כתוב בצורה טובה מאוד העוזרת להבין בקלות נושאים לא פשוטים מתחום האסמבלי.

תוך כדי קריאת הספר גיליתי אפשרויות רבות להרחיב את הפרויקט שלי, כמו הוספת צלילים ושימוש במשתנים מקומיים, שעזרו לי בכתיבת התוכנית בצורה יותר יעילה ופשוטה.

הוראות הפעלה

כדי להפעיל את המשחק יש להיכנס לדוסבוקס, לעבור לתיקיה בה checkers.exe שמור ולכתוב בחלון הדוסבוקס: “checkers”.

לאחר מכן תודפס הודעה ועל כל שחקן להקליד את שמו, קודם השחקן הלבן ואז השחור.

אחרי זה יתחיל המשחק. התור הראשון יהיה של הלבן. בסיום תור הלבן יגיע תור השחור וככה יחזור על עצמו עד שלא נותרו חיילים לצבע מסוים.

בסיום המשחק תודפס הודעת הניצחון ויש ללחוץ על כל לחצן כדי לצאת מהמשחק.

אלגוריתם הפרויקט

בתוכנית הראשית, מתבצעת הדפסת הודעות, השמורות בזיכרון בתור מערכים, המבקשות מכל משתתף לכתוב את שמו. ולאחר מכן התוכנית קולטת תווים שיכנסו למערך ריק בזיכרון בתור השמות של השחקנים, עד שיוקלד ENTER או שיוקלדו 10 תווים. לאחר מכן התוכנית עוברת למצב גרפי ומדפיסה את שמות שני השחקנים.

אחרי זה היא קוראת לפרוצדורה שמדפיסה את לוח הדמקה, שמשתמשת בפרוצדה משנית, paintSquare, שמדפיסה ריבועים בלי הפינות ועם מסגרת לבנה/שחורה. התוכנית קוראת אחרי זה לפרוצדורה המדפיסה את 24 חיילי המשחק במקומותיהם ההתחלתיים על הלוח. פרוצדורה זאת משתמשת גם כן בפרוצדורת paintSquare, ובלולאה המדפיסה כמה ריבועים כאלה עם רווחים.

אחר כך התוכנית מדפיסה חץ צהוב ליד השם הלבן, כדי לסמל שתורו כעת. החץ זז בין שני המשתתפים לסירוגין בכל פעם שמתחלף תור כדי להצהיר למשתתפים תור איזה שחקן. בכל תור, התוכנית מגדילה את המשתנה whitesEaten ב-1 אם בתור השחור נאכל חייל לבן, או blacksEaten אם השחקן הלבן אכל חיילים של היריב.

לאחר כל תור התוכנית משווה בין המשתנים blacksEaten או whitesEaten ל-12, שהוא מספר החיילים שיש לכל צד במשחק. אם ערך אחד המשתנים 12, התוכנית מפסיקה את רצף התורות, מוחקת את כל מה שמודפס ומצוייר על המסך, מדפיסה הודעת ניצחון לשחקן המנצח ומחכה ללחיצה על המקלדת. אם לא הגיע אחד המשתנים ל12, היא מעדכנת את המספר על המסך וממשיכה במשחק.

פרוצדורות

paintSquare: פרוצדורה זו מקבלת ארבעה פרמטרים:

1. צבע לריבוע
2. אורך ריבוע כולל שוליים
3. X התחלתי
4. Y התחלתי

ומדפיסה ריבוע בצבע שהוכנס עם שוליים לבנים, או שחורים במקרה שהצבע הוא לבן.

paintBoard: פרוצדורה זו לא מקבלת פרמטרים משום שמתמשים בה רק פעם אחת והיא קבועה. היא מדפיסה ריבוע שחור גדול כגודל כל הלוח ובתוכה מדפיסה 32 ריבועים כחולים בגודל 25 (כולל הקצוות) עם מרווחים של 25 בין כל אחד כדי שייראה כאילו יש ריבועים שחורים באותו גודל בין כולם. התוכנית בעצם מבצעת לולאה בתוך לולאה, כשבכל פעם משנה את הX והY ההתחלתיים של כל ריבוע.

paintWhiteChecker: פרוצדורה זו מקבלת 2 פרמטרים:

1. X התחלתי
2. Y התחלתי

והיא מדפיסה ריבוע לבן בגודל 17 במיקום שקיבלה.

paintBlackChecker: פרוצדורה זו היא זהה לפרוצדורה paintWhiteChecker, עם שינוי בצבע של הריבוע המודפס לשחור. הערה: הצבע של החיילים השחורים הוא גוון של אפור אבל לא שחור בדיוק, וזאת על מנת שהתוכנית לא תתבלבל עם לחיצה על המרווחים השחורים בין הריבועים בלוח. במהלך החוברת אתייחס לצבע זה כשחור.

paintPlayers: פרוצדורה זו מדפיסה בלולאות את חיילי המשחק במקומם ההתחלתי והיא אינה מקבלת פרמטרים. היא משתמשת בפרוצדורות paintWhiteChecker & paintBlackChecker.

click: פרוצדורה זו מקבלת קוד BIOS של צבע, ומחכה ללחיצה. לאחר שנלחץ העכבר היא בודקת האם הצבע שנלחץ שווה לצבע שקיבלה ואם לא היא ממשיכה לחכות. אם היא קיבלה '-' היא לא בודקת את הצבע הנלחץ אלא רק מחכה ללחיצה. לאחר הלחיצה ולפני בדיקת הצבע היא מנקה את באפר הלחיצות באמצעות לולאה. בסיום הפעולה cx וdx מחזיקים את הX והY של הלחיצה.

getCurrentTile: פעולה זו מקבלת X וY בCX וDX בהתאמה. באמצעות חישובים, היא מחזירה אליהם את הX והY של הריבוע לוח שהוא נמצא בו.

blackMovement & whiteMovement: פעולה זו לוקחת את מיקום הלחיצה על הדמקה ששמור במשתנים המקומיים MouseX ו MouseY, בודקת מה אפשרויות התנועה שלהם, ומחזירה את המיקומים של ריבועי הלוח האפשריים למשתנים המקומיים lredX, lredY, rRedX, rRedY.

whiteTurn: פעולה זו מחכה ללחיצת עכבר על חייל לבן, (באמצעות click) בודקת מה אפשרויות התזוזה שלו, (באמצעות whiteMovement) ומחכה ללחיצה שניה. לאחר הלחיצה השניה, היא בודקת האם הצבע הנלחץ הוא אדום. אם לא, היא חוזרת לתחילת הפעולה ומוחקת את הריבועים האדומים, כדי לאפשר לשחקן לבחור חייל אחר. אם לחץ על אדום, היא בודקת האם הלחיצה הייתה משמאל לחייל או מימין לו, ומה המרחק בין הלחיצות. אם המרחק הוא כזה של אכילה, התוכנית מעבירה את החייל אחרי החייל השחור, מוחקת את החייל השחור שנאכל, מוסיפה למשתנה הגלובלי blacksEaten אחד, ומסיימת את הפעולה. אם הלחיצה היא לא במרחק של אכילה, התוכנית מעבירה אותו למקום הנלחץ ולא מוסיפה למשתנה כלום.

blackTurn: פרוצדורה זו היא כמו הפרוצדורה whiteTurn, עם הבדלים בפעולות שנקראות (למשל: במקום whiteMovement, blackMovement) ובמשתנים (במקום blacksEaten, whitesEaten).

stringInputToMemory: פעולה זו מקבלת שני פרמטרים:

1. ההיסט של המערך בזיכרון אליו לכתוב.
2. המיקום של הסמן במסך בו לכתוב את ההד של הקלט (השורה בBYTE הגבוה, והטור בBYTE הנמוך).

הפעולה קוראת עד 10 תווים מהמשתמש, או עד שהוקלד ENTER, ומכניסה אותם לתוך מערך שהיסטו בזיכרון ידוע לה. לאחר התו האחרון שנקלט לזיכרון היא מציבה '$'.

printString: פרוצדורה זו מקבלת שלושה פרמטרים:

1. היסט המערך בזיכרון.
2. מיקום הסמן במסך בו להתחיל את ההדפסה.
3. הצבע בו המחרוזת תודפס, אם במצב גרפי.

פרוצדורה זו מדפיסה כל תו שבמערך בצבע שנקלט, עד שמגיעה לסימן '$' שמודיע לה שהיא סיימה את עבודתה כעת.

printNumber: פרוצדורה זו מקבלת שני פרמטרים:

1. היסט המספר בזיכרון.
2. מיקום הסמן על המסך בו להדפיס את המספר.

היא מבצעת חישובים על המספר, לוקחת כל ספרה בנפרד אם המספר דו ספרתי ומוסיפה לה את ערך האסקי של 0 כדי להגיע לערך האסקי של הספרה.

תיאור המשתנים בתוכנית

|  |  |  |
| --- | --- | --- |
| שם המשתנה | טיפוס | תיאור המשתנה |
| whiteName | Byte | מערך שמוקצים לו 11 בתים ואליו יוכנסו 10 תווי השם של השחקן הלבן ובסיומו '$'. |
| blackName | Byte | מערך שמוקצים לו 11 בתים ואליו יוכנסו 10 תווי השם של השחקן השחור ובסיומו '$'. |
| whitesEaten | Byte | מספר שמוגדר ל0 בהתחלה ויגדל ב1 בכל פעם שהשחקן השחור יאכל חייל לבן. ערך זה מוצג על המסך ליד שמו של השחקן השחור בכל חלקי המשחק. ומתעדכן בסיום כל תור. |
| blacksEaten | Byte | בית זה מכיל את מספר החיילים שהשחקן הלבן אכל. מספר זה מוצג ליד שמו של השחקן הלבן בכל חלקי המשחק. |
| turnSign | Byte | זהו מערך של שני בתים: '►' ו'$'. זהו הסימן שיסמל תור איזה שחקן בכל פעם שמתחלף תור. הוא נע לסירוגין בין שם השחקן הלבן לשם השחקן השחור בסיום כל תור. |
| getNames | Byte | זוהי מחרוזת שכתובה בה הודעה שמבקשת מהשחקנים לכתוב את שמם. |
| getWhiteName | Byte | זוהי מחרוזת שכתובה בה הודעה שמצהירה שאחריה השחקן הלבן מתבקש לכתוב את שמו. |
| getBlackName | Byte | אותו דבר רק של השחקן השחור. |
| isThe | Byte | מחרוזת שלפניה יהיה כתוב שמו של המנצח ובה כתוב "הוא המנצח!" באנגלית. |
| pressAny | Byte | מחרוזת שמבקשת מהמשתמש ללחוץ על כל מקש כדי לסיים את המשחק. |

קוד התוכנית

JUMPS

IDEAL

MODEL small

STACK 100h

DATASEG

whiteName db 11 dup (?)

blackName db 11 dup (?)

whitesEaten db 0

blacksEaten db 0

turnSign db 16,'$'

getNames db 'Type the names of each player (until 10 chars). Press ENTER to finish:$'

getWhiteName db 'White player:$'

getBlackName db 'Black player:$'

isThe db 'is the winner!$'

pressAny db 'Press any key to finish$'

CODESEG

proc paintSquare

;this procedure needs us to push 4 parameters before we call it in that order: color, length, x and y.

push bp

mov bp, sp

len equ [word ptr bp+8]

initX equ [word ptr bp+6]

initY equ [word ptr bp+4]

sub sp, 4 ;create place for two local variables

locX equ [word ptr bp-2]

locY equ [word ptr bp-4] ;the variables we are going to use for our pixels' (x;y)

push cx

push dx

push ax

push bx

mov dx, initY

mov locY, dx ;put the initial y in our local y to paint the first pixel

;outline

cmp [word ptr bp+10],15 ;check if the color of the inside is white

jne black

mov ax, 0c00h

jmp notBlack

black:

mov ax, 0c1fh ;the color white 1fh for the border and 0c for printing the pixels of the outline

notBlack:

mov cx, initX

mov locX, cx ;put the beginning of the line in locX

inc locX ;skip the first pixel so it will be black

mov cx, len ;the number of pixels in each line

sub cx,2 ;so the last and first pixels will remain black

drawFirstLine:

push cx ;for the loop

mov cx, locX ;the x of the upcoming pixel

int 10h

inc locX ;increase locX for the next pixel

pop cx

loop drawFirstLine

mov dx, initY

mov cx, len

sub cx,2 ;the first and last pixels of the collumn won't be painted

drawFirstCol:

push cx

inc dx ;go down one line

mov cx, initX

int 10h ;the color remains the same from before

pop cx

loop drawFirstCol

mov dx, initY

inc dx ;the place is one pixel below the first line and at the last collumn

mov cx, initX

add cx, len

dec cx

mov locX, cx ;the last pixel in the line

sub cx, initX

dec cx ;don't need to paint the first and last pixels

drawLastCol:

push cx

mov cx, locX

int 10h ;the color remains white

inc dx ;go down one line

pop cx

loop drawLastCol

mov dx, initY

add dx, len

dec dx ;the last line

mov cx, initX

inc cx

mov locX, cx ;the first pixel in the line after the one not painted

mov cx, len

sub cx, 2 ;for the loop

drawLastLine:

push cx

mov cx, locX

int 10h

inc locX ;for the next pixel x

pop cx

loop drawLastLine

fill:

;the filling

mov ax, [bp+10] ;the color is in [bp+10]

mov ah, 0ch

mov dx, initY

inc dx

mov locY, dx ;the y for the filling begins one pixel below the outline

mov cx, len

sub cx, 2

drawFillSquare:

push cx ;for the loop of the number of lines

mov cx, initX

inc cx

mov locX, cx ;the x for the filling begins one pixel after the outline

mov cx, len

sub cx, 2 ;for the loop of the pixels in each line

drawFillLine:

push cx

mov cx, locX ;the x of the upcoming pixel is put in cx

mov dx, locY ;the y of the upcoming pixel is put in dx

int 10h

inc locX ;increase locX for the next pixel

pop cx

loop drawFillLine

inc locY ;increase locY for the next line

pop cx ;the one we pushed before we drew the line

loop drawFillSquare

pop bx

pop ax

pop dx

pop cx

add sp, 4

pop bp

ret 8

endp paintSquare

proc paintBoard

push bp

mov bp, sp

sub sp, 4

beginY equ [word ptr bp-2]

beginX equ [word ptr bp-4]

push cx

bigSquare:

push 0 ;color

push 193 ;length. not 192 because the last square has an extra line at the bottom - his outline

push 120 ;x

push 4 ;y

call paintSquare

mov beginX, 120

mov beginY,4

mov cx, 8

cols8:

;this loop paints 4 collumns of blue squares. they begin at the top of the board

push cx

mov cx, 4

paint4Squares:

;this loop paints a collumn of 4 blue squares that begins at the top of the board

push 3 ;color

push 25 ;length

push beginX ;x

push beginY ;y

call paintSquare

add beginY, 48

loop paint4Squares

add beginX, 24

cmp beginY, 196

je changeToBottom

mov beginY,4

jmp continue

changeToBottom:

mov beginY, 28

continue:

pop cx

loop cols8

pop cx

add sp, 4

pop bp

ret

endp paintBoard

proc paintWhiteChecker

push bp

mov bp, sp

push 15 ;white

push 17 ;length

push [bp+6] ;x

push [bp+4] ;y

call paintSquare

pop bp

ret 4

endp paintWhiteChecker

proc paintBlackChecker

push bp

mov bp, sp

push 19 ;black of checkers

push 17 ;length

push [bp+6] ;x

push [bp+4] ;y

call paintSquare

pop bp

ret 4

endp paintBlackChecker

proc paintPlayers

push bp

mov bp, sp

sub sp, 4

beginX equ [word ptr bp-2]

beginY equ [word ptr bp-4]

mov beginX, 124

mov beginY, 8 ;4 pixels next to the border of the tile

mov cx, 3

x3Lines:

push cx

mov cx, 4

x4Playrs:

push cx

push beginX ;x

push beginY ;y

call paintWhiteChecker

add beginX, 48

pop cx

loop x4Playrs

cmp beginX, 316

je nowSecond

mov beginX,124

jmp continue2

nowSecond:

mov beginX, 148

continue2:

add beginY, 24

pop cx

loop x3Lines ;now there are three lines of white checkers at the top

mov beginX, 148

mov beginY, 176

mov cx,3

x3blackLines:

push cx

mov cx, 4

x4Checkers:

push cx

push beginX ;x

push beginY ;y

call paintBlackChecker

add beginX, 48

pop cx

loop x4Checkers

cmp beginX, 340

je nowMiddle

mov beginX, 148

jmp contin2

nowMiddle:

mov beginX, 124

contin2:

sub beginY,24

pop cx

loop x3blackLines

add sp, 4

pop bp

ret

endp paintPlayers

proc click

;this procedure waits for a click on a color pushed returns the x and y of the click in cx, dx, respectively

push bp

mov bp, sp

push ax

push bx

Mouse:

mov ax, 3

int 33h

cmp bx, 0

je Mouse

push cx

push dx

;clear the buffer of clicks

clear:

mov ax, 3

int 33h

cmp bx, 0

jne clear

pop dx

pop cx

shr cx, 1

mov bx, 0

dec dx

mov ax, [word ptr bp+4]

cmp ax, '-' ;this means we don't need to check the color clicked

je stop

mov ah, 0dh

int 10h ;check the color of the place clicked

cmp al, [byte ptr bp+4] ;work only if the color is of the one pushed

jne Mouse

stop:

pop bx

pop ax

pop bp

ret 2

endp click

proc blackMovement

;this procedure puts the options of movement for a black checker in the lred and rRed variables

push ax

mov cx, mouseX

mov dx, mouseY

sub cx, 24

sub dx, 24 ;the square left of the checker

mov ah, 0dh

int 10h

cmp al, 15 ;white checker

je checkAboveLeft

cmp al, 3

jne checkRight

call getCurrentTile

mov lredX, cx

mov lredY, dx

jmp checkRight

checkAboveLeft:

mov cx, mouseX

mov dx, mouseY

sub cx, 48

sub dx, 48

mov ah, 0dh

int 10h

cmp al, 3

jne checkRight

call getCurrentTile

mov lredX, cx

mov lredY, dx

checkRight:

mov cx, mouseX

mov dx, mouseY

add cx, 24

sub dx, 24

mov ah, 0dh

int 10h

cmp al, 15

je checkAboveRight

cmp al, 3

jne endOfMove

call getCurrentTile

mov rRedX, cx

mov rRedY, dx

jmp endOfMove

checkAboveRight:

mov cx, mouseX

mov dx, mouseY

add cx, 48

sub dx, 48

int 10h

cmp al, 3

jne endOfMove

call getCurrentTile

mov rRedX, cx

mov rRedY, dx

endOfMove:

pop ax

ret

endp blackMovement

proc whiteMovement

;this procedure puts the options of movement for a white checker in the lred and rRed variables

push ax

mov cx, mouseX

mov dx, mouseY

sub cx, 24

add dx, 24 ;the square left of the checker

mov ah, 0dh

int 10h

cmp al, 19 ;check if there is a black checker

je checkBelowLeft

cmp al, 3 ;check if it is empty

jne checktheRight

call getCurrentTile

mov lredX, cx

mov lredY, dx

jmp checktheRight

checkBelowLeft:

mov cx, mouseX

mov dx, mouseY

sub cx, 48

add dx, 48

mov ah, 0dh

int 10h

cmp al, 3

jne checktheRight

call getCurrentTile

mov lredX, cx

mov lredY, dx

checktheRight:

mov cx, mouseX

mov dx, mouseY

add cx, 24

add dx, 24

mov ah, 0dh

int 10h

cmp al, 19

je checkBelowRight

cmp al, 3

jne endtheMovement

call getCurrentTile

mov rRedX, cx

mov rRedY, dx

jmp endtheMovement

checkBelowRight:

mov cx, mouseX

mov dx, mouseY

add cx, 48

add dx, 48

int 10h

cmp al, 3

jne endtheMovement

call getCurrentTile

mov rRedX, cx

mov rRedY, dx

endtheMovement:

pop ax

ret

endp whiteMovement

proc whiteTurn

push bp

mov bp, sp

sub sp, 14

mouseX equ [word ptr bp-2] ;the x of the click that tells which white checker to move

mouseY equ [word ptr bp-4] ;the y of the click that tells which white checker to move

rRedX equ [word ptr bp-6] ;the x of the right red square

rRedY equ [word ptr bp-8] ;the y of the right red square

lRedX equ [word ptr bp-10] ;the x of the left red square

lRedY equ [word ptr bp-12] ;the y of the left red square

mouseX2 equ [word ptr bp-14] ;the x of the click of the red square

push dx

push cx

push bx

push ax

startOfTurn:

mov rRedX, 0

mov lRedX, 0 ;when we will check if they are 0, we want them to be 0

xor di, di ;di is the flag that tells the program wether the second click wasn't on red. normally it is 0

mov ax, 1

int 33h ;show mouse

push 15 ;color of white checker

call click

jmp redSquares

;this is a dead area that the program won't get to unless it jumps to one of the checkpoints

checkpoint: ;this one is for if the player pressed on a checker and he has regrets

jmp startOfTurn ;jump to the start

redSquares:

mov ax, 2

int 33h

mov mouseX, cx

mov mouseY, dx

call whiteMovement

cmp lredX, 0

je paintRightRed

;paint the left one red

push 4

push 25

push lRedX

push lredY

call paintSquare

paintRightRed:

cmp rRedX, 0

je checkLeftAgain

push 4

push 25

push rRedX

push rRedY

call paintSquare

jmp secondClick

checkLeftAgain:

cmp lredX, 0

je checkpoint

secondClick:

mov ax, 1

int 33h

push '-'

call click

mov ah, 0dh

int 10h ;check the color of the place clicked

cmp al, 4

je ok

mov di, 0ffh ;di is a flag that will tell in the future wether to go back to the beginning or not

jmp returntheLeft

ok:

mov mouseX2, cx

mov ax, 2

int 33h ;erase the mouse, so the mouse won't interfere with the erasing of the checker

mov cx, mouseX

mov dx, mouseY

call getCurrentTile ;erase the checker moved

push 3 ;color

push 25 ;length

push cx

push dx

call paintSquare

returntheLeft:

mov ax, 2

int 33h ;also here because we might not need to erase the checker

cmp lRedX, 0 ;this means left wasn't red in the first place

je returntheRight

push 3 ;color

push 25 ;length

push lRedX

push lRedY

call paintSquare

returntheRight:

cmp rRedX, 0 ;this means the right one wasn't red

je whereToMove

push 3 ;color

push 25 ;length

push rRedX

push rRedY

call paintSquare

whereToMove:

cmp di, 0 ;di will not be 0 only if it was changed beforehand

jne checkpoint

mov cx, mouseX2

cmp cx, mouseX ;check where the new click is relatively to the first click

ja movetoRight

jb movetoLeft

movetoRight:

mov cx, rRedX

sub cx, mouseX

cmp cl, 29 ;29 is the smallest distance between a board corner and a checker that can eat

jb noFeast

mov cx, rRedX

sub cx, 24

mov dx, rRedY

sub dx, 24

push 3

push 25

push cx

push dx

call paintSquare ;eat the black!

inc [blacksEaten]

noFeast:

mov cx, rRedX

mov dx, rRedY

add dx, 4

add cx, 4

push cx

push dx

call paintWhiteChecker

jmp endd

movetoLeft:

mov cx, mouseX

sub cx, lredX ;check if the space between the checker and the place moved is over 1 square

cmp cx, 48

jb noEat

mov cx, lredX

add cx, 24

mov dx, lredY

sub dx, 24

push 3

push 25

push cx

push dx

call paintSquare

inc [blacksEaten]

noEat:

mov cx, lredX

mov dx, lredY

add cx, 4

add dx, 4

push cx

push dx

call paintWhiteChecker

endd:

mov ax, 1

int 33h ;return the mouse

push offset blacksEaten

push 50dh

call printNumber

pop ax

pop bx

pop cx

pop dx

add sp, 14

pop bp

ret

endp whiteTurn

proc blackTurn

push bp

mov bp, sp

sub sp, 14 ;room for 7 word variables- mouseX and mouseY, and 4 for the red squares xs and ys and the second mouse click x

mouseX equ [word ptr bp-2] ;the x of the click that tells which black checker to move

mouseY equ [word ptr bp-4] ;the y of the click that tells which black checker to move

rRedX equ [word ptr bp-6] ;the x of the right red square

rRedY equ [word ptr bp-8] ;the y of the right red square

lRedX equ [word ptr bp-10] ;the x of the left red square

lRedY equ [word ptr bp-12] ;the y of the left red square

mouseX2 equ [word ptr bp-14] ;the x of the click of the red square

push ax

push bx

push cx

push dx

beginning:

mov rRedX, 0

mov lRedX, 0 ;when we will check if they are 0, we want them to be 0

xor di, di ;di is the flag that tells the program wether the second click wasn't on red. normally it is 0

mov ax, 1

int 33h ;show mouse

push 19 ;color of black checker

call click

jmp new

;this is a dead area that the program won't get to unless it jumps to one of the checkpoints

checkpoint2: ;this one is for if the player pressed on a player and regrets his decision

jmp beginning ;jump to the start

new:

mov mouseX, cx

mov mouseY, dx

call blackMovement

cmp lredX, 0

je paintRight

push 4

push 25

push lRedX

push lredY

call paintSquare

paintRight:

cmp rRedX, 0

je checkLeftAgen

push 4

push 25

push rRedX

push rRedY

call paintSquare

jmp waitForSecond

checkLeftAgen:

cmp lredX, 0

je checkpoint2

waitForSecond:

push '-'

call click

mov ah, 0dh

int 10h ;check the color of the place clicked

cmp al, 4

je skip

mov di, 0ffh ;di is a flag that tells wether to go back to the beginning or not

jmp returnLeft

skip:

mov mouseX2, cx

mov ax, 2

int 33h ;erase the mouse, so the mouse won't interfere with the erasing of the checker

mov cx, mouseX

mov dx, mouseY

call getCurrentTile ;erase the checker moved

push 3 ;color

push 25 ;length

push cx

push dx

call paintSquare

returnLeft:

mov ax, 2

int 33h ;also here because we might not need to erase the checker

cmp lRedX, 0 ;this means left wasn't red in the first place

je returnRight

push 3 ;color

push 25 ;length

push lRedX

push lRedY

call paintSquare

returnRight:

cmp rRedX, 0 ;this means the right one wasn't red

je whereToMove?

push 3 ;color

push 25 ;length

push rRedX

push rRedY

call paintSquare

whereToMove?:

cmp di, 0 ;di will not be 0 only if we changed it

jne checkpoint2

mov cx, mouseX2

cmp cx, mouseX ;check where the new click is relatively to the first click

ja moveRight

jb moveLeft

moveRight:

mov cx, rRedX

sub cx, mouseX

cmp cl, 29 ;29 is the smallest distance between a board corner and a checker that can eat

jb notHungry

mov cx, rRedX

sub cx, 24

mov dx, rRedY

add dx, 24

push 3

push 25

push cx

push dx

call paintSquare

inc [whitesEaten]

notHungry:

mov cx, rRedX

mov dx, rRedY

add dx, 4

add cx, 4

push cx

push dx

call paintBlackChecker

jmp end1

moveLeft:

mov cx, mouseX

sub cx, lredX ;check if the space between the checker and the place moved is over 1 square

cmp cx, 48

jb notEat

mov cx, lredX

add cx, 24

mov dx, lredY

add dx, 24

push 3

push 25

push cx

push dx

call paintSquare

inc [whitesEaten]

notEat:

mov cx, lredX

mov dx, lredY

add cx, 4

add dx, 4

push cx

push dx

call paintBlackChecker

end1:

mov ax, 1

int 33h ;return the mouse

push offset whitesEaten

push 060dh

call printNumber

pop dx

pop cx

pop bx

pop ax

add sp, 14

pop bp

ret

endp blackTurn

proc stringInputToMemory

;in this procedure you need to push the offset of the array in the data segment and the cursor location of where to print the echo

push bp

mov bp, sp

push si

push dx

push bx

push ax

mov dx, [bp+4] ;the cursor location

mov ah, 2

int 10h

mov bx, 0

mov si, [bp+6] ;the offset

jmp getChar

backspace:

cmp bx, 0

jz getChar

dec bx

mov dl, 32 ;space

mov ah, 2

int 21h

mov dl, 8

mov ah, 2

int 21h

getChar:

cmp bx, 10

je proceed

mov ah,1

int 21h

cmp al, 8 ;backspace

je backspace

cmp al, 13 ;enter

je proceed

mov [byte si+bx], al

inc bx

jmp getChar

proceed:

mov [byte ptr si+bx], '$'

pop ax

pop bx

pop dx

pop si

pop bp

ret 4

endp stringInputToMemory

proc printString

;this procedure prints a string (that ends with '$') who's offset is pushed and after it the row (h) and collumn (l)

;and lastly the color

push bp

mov bp, sp

push si

push ax

off equ [bp+8]

curs equ [bp+6]

color equ [bp+4]

mov si, off

mov bx, color

mov dx, curs

mov ah, 2

int 10h

begin:

cmp [byte ptr si], '$'

je outro

mov al, [si]

mov ah, 0eh

int 10h

inc si

jmp begin

outro:

pop ax

pop si

pop bp

ret 6

endp printString

proc printNumber

;this one prints a number. you need to push the offset, row (h) and collumn (l)

push bp

mov bp, sp

push si

push dx

push bx

push ax

ofset equ [word ptr bp+6]

cursor equ [word ptr bp+4]

mov dx, cursor

mov ah, 2

int 10h

xor dx, dx

mov si, ofset

xor ah, ah

mov al, [si] ;character

cmp al, 10

jb oneDigit

xor ah, ah

mov al, [si]

mov bx, 10

div bx

add al, '0'

mov ah, 0eh

mov bl, 0ah

int 10h

mov ax, dx

oneDigit:

add al, '0'

mov bl, 0ah

mov ah, 0eh

int 10h

pop ax

pop bx

pop dx

pop si

pop bp

ret 4

endp printNumber

proc getCurrentTile

;this procedure uses cx and dx and returns the x,y of the beginning of the current tile

push ax

push bx

;cx

mov ax, cx

mov bx, 24

push dx

push dx ;once for the division and once for the substraction

xor dx, dx

div bx

sub cx, dx

;dx

xor dx, dx

pop ax

div bx

mov bx, dx

pop dx

sub dx, bx

add dx, 4 ;the board squares are is 4 pixels below the 24 lines

pop bx

pop ax

ret

endp getCurrentTile

start:

mov ax, @data

mov ds, ax

mov ax, 2

int 10h

push offset getNames

push 200h

push 5

call printString

push offset getWhiteName

push 300h

push 5

call printString

push offset whiteName

push 400h

call stringInputToMemory

push offset getBlackName

push 600h

push 5

call printString

push offset blackName

push 700h

call stringInputToMemory

mov ax, 13h

int 10h ;change to graphic mode

push offset whiteName

push 501h

push 15 ;white

call printString

push offset blackName

push 601h

push 7 ;dark/light gray

call printString

push offset blacksEaten

push 50dh

call printNumber

push offset whitesEaten

push 60dh

call printNumber

call paintBoard

call paintPlayers

Game:

push offset turnSign

push 500h

push 14 ;yellow

call printString

mov dx, 600h

mov ah, 2

int 10h

mov ax, 0e20h ;space

int 10h

call whiteTurn

cmp [blacksEaten], 12

jz whiteWins

push offset turnSign

push 600h

push 14 ;yellow

call printString

mov dx, 500h

mov ah, 2

int 10h

mov ax, 0e20h ;space

int 10h

call blackTurn

cmp [whitesEaten], 12

jz blackWins

jmp game

blackWins:

mov ax, 13

int 10h ;text mode

push offset blackName

push 40ah

push 2

call printString

push offset isThe

push 50ah

push 4

call printString

;3 sec delay

mov cx, 2dh

mov dx, 0c6c0h

mov ah, 86

int 21h

push offset pressAny

push 600h

push 3

call printString

mov ah, 0

int 16h

mov ax, 2

int 10h ;text mode

jmp exit

whiteWins:

mov ax, 13

int 10h ;text mode

push offset whiteName

push 40ah

push 15

call printString

push offset isThe

push 50ah

push 4

call printString

;3 sec delay

mov cx, 2dh

mov dx, 0c6c0h

mov ah, 86

int 21h

push offset pressAny

push 600h

push 3

call printString

mov ah, 0

int 16h

mov ax, 2

int 10h ;text mode

exit:

mov ax, 4c00h

int 21h

end start