תיק פרוייקט-

הרץ במבוך

מגיש: שירן גרצמן - ת"ז XXXX

מורה מלווה: XXXX

כיתה:י' X - תיכון XXXX

תוכן עניינים

מבוא ---------------------------------- 3

נושא העבודה -------------------------- 4

אופן הפעלה ---------------------------- 5

גרסאות -------------------------------- 6

תיעוד והסבר --------------------------- 7

תרשימי זרימה ------------------------ 10

רשימת פעולות ------------------------- 12

דוגמאות הרצה ------------------------ 14

סיכום אישי --------------------------- 16

מבוא

שם העבודה: הרץ במבוך

שם קובץ ההרצה: 3Dmaze  
אין קבצים נלווים.

סביבת פיתוח: Notepad++, Python 3.8 IDLE  
הפרוייק עצמו נכתב בNpp- וטבלאות עזר וחישובים שנעשו מראש נעשו בפייתון 3.8 בסביבת הפיתוח הדיפולטית.

סביבת הרצה: DOSBox

סביבת עבודה: TASM ו TLink

Text

Description automatically generatedכדי להגיע מקוד המקור לקובץ ההרצה:   
- נריץ דרך DOSBox את התוכנית TASM עם הקובץ 3Dmaze.asm כארגומנט. כעת יש לנו קובץ בשם 3Dmaze.obj. - כעת נריץ TLink עם קובץ זה כארגומנט. לאחר הרצת הלינקר יהיה לנו את קובץ ההרצה הרצוי אותו ניתן להריץ דרך הDOSBox. לפני הרצת הקובץ יש להגדיר בDOSBox:  
cycles = max. מצ"ב תמונה שמסכמת את התהליך:

נושא העבודה

נושא התכנית שאני מגיש הוא משחק מבוך שניתן לשחק ממבט "מלמעלה למטה" (top down view - דו מימדי: כמו מבוך על נייר) או ממבט ראשון (תלת מימדי: מאפשר למשחק לתפוס את הסביבה כאילו שהוא נמצא בתוך המבוך).

המשחק מבוסס על שלבים, כאשר בכל שלב יש להגיע מנקודת ההתחלה במבוך לנקודת הסיום על מנת להתקדם לשלב הבא.

מטרת הפרוייקט - להוכיח שגם בשפה מסורבלת וישנה, שאין בה אפילו מספרים לא שלמים כמו אסמבלי 8086, אפשר להכין תכנית עם ביצועים טובים שמשלבת גרפיקה תלת מימדית ומשחק מהנה ופשוט כגון מבוך.

אופן ההפעלה

כדי להריץ את התוכנית לאחר שיצרנו את קובץ ההרצה (מפורט במבוא) יש לכתוב בDOSBox את שם התוכנית: 3Dmaze

בכל שלב במשחק ניתן יהיה לשנות את מצב המשחק בלחיצה על הכפתורים t אוr כדי לעבור למצב תלת מימדי או דו מימדי בהתאמה.

בכל שלב המשתמש יוצב בנקודת ההתחלה של המבוך, ועליו למצוא את נקודת הסיום ולהגיע אליה. נקודת הסיום היא הקיר הירוק (קיים קיר ירוק 1 בלבד בכל שלב). בכל שלב מופיע מבוך שונה. כאשר המשתמש מגיע לנקודת הסיום הוא מועבר לשלב הבא.

ניתן לנוע בתוך המבוך באמצעות המקשים:  
 "W" - ללכת קדימה בכיוון שאליו מביטה הדמות.  
 "S" - ללכת בכיוון הנגדי לזה שהדמות מביטה.  
 "D" - להסתובב במקום ימינה  
 "A" - להסתובב במקום שמאלה.

בנוסף בכל רגע במשחק לחיצה על "Q" תסגור את התכנית ותחזיר שליטה לDOS.  
במהלך המשחק ניתן לשנות את מצב המשחק (את נקודת המבט) על ידי לחיצה על "t" או "r" כדי לעבור למצב תלת מימדי או דו מימדי בהתאמה.

במהלך המשחק אסור למשתמש לגעת בקירות למעט הקיר הירוק. כל נגיע בקיר (למעט הקיר הירוק) תחזיר את המשתמש לתחילת המבוך.

גרסאות מערכת

הגרסה ה"יציבה" הראשונה של התוכנית הייתה תוכנית דו מימדית ממבט על, שדומה לגרסא הנוכחית ממבט על, אך לא היו בה שלבים, אלא מבוך אחד קבוע וגרפיקה ברמה נמוכה.  
הגרסה השנייה הייתה דומה לראשונה, אך המשחק היה ממבט הדמות. גם בגרסה זאת הגרפיקה הייתה ברמה נמוכה.

כעת, בגרסא הנוכחית:

ניתן לעבור בין מצבי משחק (דו מימדי ותלת מימדי) כאשר לשני המצבים ישנה גרפיקה טובה, וקיים מספר שלבים, כאשר בכל שלב - מבוך ייחודי.

בגרסאות הבאות הייתי כולל:  
-סאונד להליכה,  
-כפתור להליכה מהירה,  
-מגוון מבוכים נוספים.

תיעוד והסבר הפתרון

לולאה ראשית

אחרי ההכנה הראשונית התכנית מבצעת לולאה אין סופית, בה היא מחכה לקלט ופועלת לפיו. ללא קלט, הפלט שעל המסך לא ישתנה.

כדי לבדוק קלט, כל איטרצייה, הפעולה הראשית Update קוראת לפעולה GetInput. פעולה זו משתמשת בפסיקה  
 int 21h כדי לבדוק (בצורה חצי-אסינכרונית) האם יש קלט. אם כן – הפעולה תעדכן את את הנתונים שמאחורי הקלעים (מצב משחק, מיקום והכיוון אליו השחקן מסתכל) ותקרא לפעולה flipDisplay שתעדכן את הפלט שעל המסך בהתאם לנתונים החדשים. הסבר דרך הפעולה של flipDisplay יופיע כתרשים זרימה בהמשך, בסעיף המיועד לכך.

שימוש בטריגונומטריה

על מנת לעמוד במקום ולהסתובב, וכדי ללכת קדימה לא משנה לאן אנו מסתכלים, יש צורך בשימוש בטריגונומטריה. הצורה בה מימשתי זאת פשוטה: לכל אובייקט שצריך לזוז לכיוון מסויים נחשב דלתא (delta) לזווית שלו. דלתא היא זוג מספרים שמייצגים x, y שכל פעם שנוסיף אותם למיקום של האובייקט – הוא יזוז לכיוון אליו חישבנו את הדלתא. את הדלתא של זווית כללית a נחשב בעזרת הנוסחא:  
 deltay = sin(a), deltax = cos(a).   
כדי לחשב סינוס וקוסינוס הכנתי מראש טבלת סינוסים שלכל ערך בין 0-2048 תחזיר את הסינוס של הזווית שלו. מטבלת סיונוס קל לחשב גם קוסינוס  
בזכות השימוש בדלתות ומבנה הזכרון הייחודי שעליו מפורט בהמשך ניתן להסתובב במקום (פשוט לשנות את משתנה הזווית) וללכת ולכיוון אליו אנו מסתכלים (לחשב את הדלתא ולהוסיף אותה כמה פעמים שנרצה). השימוש האינטנסיבי ביותר בדלתא בתוכנית קורא בפעולה castRay. פעולה זו מקבלת זווית (יחסית לזווית של השחקן), ובעזרת דלתא היא "יורה קרן" לכיוון זה וכאשר הקרן פוגעת בקיר היא מחזירה נתונים אודות הפגיעה (המרחק שהקרן עברה, צבע הקיר).  
בזכות פעולה זו ניתן לצייר סביבה תלת מימדית, שכן הציור מתבצע בעזרת נתונים על קרן שמייצגת את העמודה השמאלית ביותר, ועוד קרן שמייצגת את העמודה שמימינה לה וכו...

שמירת נתונים:

מפני שאין באסמבלי 8086 floating point (מספרים לא שלמים) אך עם זאת היה צורך בשימוש בפונקציות טריגונומטריות (שאמורות להחזיר מספר כמעט תמיד לא שלם בין1- ל 1), ומפני שהיה צורך בשימוש בזוויות ומיקומים בדיוק של יותר מ360 מעלות ו320 פיקסלים בהתאמה, מימשתי שיטה, שבה אני מאכסן נתונים ככפל בסקלר של הנתון "האמיתי" (הסקלר תמיד חזקה של 2 כדי לחשב מהר). לדוגמא אם ארצה לשמור את המיקום על ציר ה x: **130.3**, אשמור בזיכרון מספר שקרוב מאוד ל ( הוא סקלר קבוע לכל הנתונים של מיקום). כל פעם שארצה להשתמש במשתנה זה אחלק בסקלר () ותתתקבל תוצאה מעוגלת של 130. שיטה זו עוזרת מפני שבעזרת אני יכול להוסיף מספרים קטנים, לדוגמה 0.4, עוד פעם ועוד פעם, ויהיו צעדים הדרגתיים למספרים השלמים שייצאו בעיגול. לא אוכל לעשות זאת עם מספרים ששמורים כ"ערכים האמיתיים", שכן 130.3 כבר יהיה שמור מעוגל ל 130, ואם נוסיף 0.6 יתעגל 130.6 ל-130. כשאוסיף עוד 0.6 עדיין ישאר 130.6 שיתעגל ל-130, למרות שכבר אמור להיות 131. השיטה שאותה פיתחתי מאפשרת לפתור בעיה זאת.

תרשימי זרימה

Diagram

Description automatically generatedתרשים זרימה לפעולה הראשית :Update

תרשים זרימה לפעולה שמציירת למסך – flipDisplay

Diagram

Description automatically generated

רשימת פעולות

|  |  |  |  |
| --- | --- | --- | --- |
| תיאור | טענת יציאה | טענת כניסה | שם פעולה |
| Changes program from text mode to graphic mode | Opens a graphic display | - | SetGraphic |
| The infinite program loop | - | - | Update |
| Seeks for input and acts according to it | Changes player position and rotation, calls flipDisplay | - | GetInput |
| Called by flipDisplay | Draws 2d maze | [map] = reference to current maze | Init |
| Called to get info to later draw the environment in first person | si = ray length  cx = distance left  al = wall color | si = rayAngle  [map] = reference to current maze | castRay |
| Used to calculate deltas. Uses a premade values table. si: 0-2047 | ax = sin(si) | si = angle | get\_sin |
| Used to calculate deltas. Uses get\_sin  si: 0-2048 | ax = cosin(si) | si = angle | get\_cos |
| Calculates player's delta based on rotation | [PlayerDeltaX] [PlayerDeltaY] | [PlayerRotation] | UpdateDelta |
| Calculates Delta for rays when casted | [tayDeltaX] [rayDeltaY] | bx = ray rotation | UpdateDeltaRay |
| Used when drawing frames in first person | One vertical line on the screen | cx = x  dx = y  si = length  al = color | DrawVerticalLine |
| used by init when drawing in 2d | Draws one tile (square) on screen | Ax = color  Dx = y  Cx = x | DrawTile |
| Called by flipDisplay on 2d mode | Draw a short ray showing where the player faces | [map] = reference to current maze | guideLine |
| Checks if player touches a wall | [PlayerPositionX]  [PlayerPositionY]  [map] | [PlayerPositionX]  [PlayerPositionY]  [map] | collision |
| Called by collision if the wall isn’t green, resets the player to maze start | [PlayerPositionX]  [PlayerPositionY]  [playerRotation]  Calls flipDisplay |  | resetPosition |

A picture containing text, clipart, vector graphics

Description automatically generatedArrow, funnel chart

Description automatically generatedדוגמאות הרצה

Logo

Description automatically generated with low confidence

התמונה השנייה והשלישית נלקחו באותו המקום בשני המצבים – דו ותלת מימד

סיכום אישי

אני מאוד מרוצה מהתהליך שעברתי בכתיבת בפרוייקט. קודם כל, התמודדתי לבד עם בעיות שהייתי צריך למצוא אליהן פתרון שהוא לא בהכרח פשוט, ותחושת הסיפוק אחרי סשן דיבאגינג שלוקח שעות הוא אדיר. הרגע הזה שמבינים "הצלחתי, פתרתי את הבעיה". זה רגע שאין לא אף תחליף. בנוסף לסיפוק שמגיע מפתרון הבעיות, למדתי ידע רב שבאמת יוכל לשמש אותי בעתיד. שימוש בדלתות (delta), אחסון מספרים בעזרת כפל בסקלר, ושמירת נתונים מראש בטבלה כדי לחסוך בזמן חישוב הם רק חלק מהדברים שלמדתי במהלך הפרוייקט, ובגלל שהגעתי אליהם בעצמי ולא לימדו אותי אותם – אני בטוח שאני אזכור דברים אלו לכל החיים. אני חושב שפרוייקטים כאלו מאוד תורמים גם לידע וגם לפיתוח האישי ואני מקווה שיהיו בעתיד עוד פרוייקטים דומים.