# תוכן הגליון



**קראתם את העיתון?**

**אל תשכחו למלא את טופס המשוב**

**הנמצא בעמוד האחרון**

**ולשלוח אותו אל מינהלת מל"מ.**

**כתובתנו:**

**מרכז המורים הארצי למדעי המחשב**

**הפקולטה לחינוך למדע וטכנולוגיה, טכניון, חיפה 3200003**

**טלפון 04-8292880 , פקס 04-8293004**

# מאמר מערכת

## קוראים יקרים

לפניכם הגליון השני של "הבטים בהוראת מדעי המחשב" לשנת תשע"ו. אתם מוזמנים לקרוא אותו באתר ללא תשלום ולאחר מכן למלא את טופס המשוב ולשלוח אותו למינהלת מל"מ. שליחת דפי המשוב היא התנאי להמשך קיומו של העיתון.

## מה בגיליון הפעם?

הפעם בחרנו להתמקד ב**סביבות למידה ויזואליות למתחילים בלימוד תכנות**. בשנים האחרונות פותחו סביבות תכנותיות רבות שכולן נועדו לתלמידים מתחילים והמפתחים שלהם מצהירים כי הן אמורות להקל על הוראת תכנות למתחילים. המאמר יבחן מספר סביבות כאלה וישים דגש על 3 סביבות פופולריות שפותחו בעשור האחרון: סביבת Alice, סביבת Scratch, וסביבת Greenfoot.

המאמר של אודי מלכה, עוסק ב**הבנת הנקרא במדעי המחשב.** למדעי המחשב יש תדמית של מקצוע בו לטקסט חשיבות מעטה ושעיקר המיומנויות הנדרשות מהתלמידים מתמקדות בטכניקה ובלוגיקה. אולם כל מורה יודע שהמצב אינו כזה. המאמר בוחן את הסוגיה תוך בדיקה של דוגמאות לפתרון הבעיה מתחום המתמטיקה. במאמר מוצעת אסטרטגיה לפתרון שאלה מרובת מלל במדעי המחשב תוך מתן דוגמא ליישום האסטרטגיה. האסטרטגיה נבנתה תוך שימוש באלמנטים מתוך אסטרטגיות שפותחו במקור עבור תלמידים לקויי למידה.

תוכלו למצוא בגיליון גם קול קורא לכנס המורים הארצי. הכנס יתקיים במכללה ע"ש הרמלין בנתניה בתאריך 27.12.16.

מומלץ לעקוב אחר ההודעות השוטפות על פעילויות המרכז הארצי באתר האינטרנט שלנו http://cse.proj.ac.il.
אתם מוזמנים להמשיך לגלוש באתר, לקרוא ולהוריד קבצים.

נשמח גם לקבל מכם חומרים מפרי עטכם בכל נושא שיכול לעניין את קוראי העיתון.

**קריאה מהנה וחופשה מוצלחת,**

**ממערכת העיתון**

**ומצוות המרכז הארצי**

# קול קורא להגשת הצעות לכנס המורים הארצי תשע"ז

הכנס יתקיים בתאריך 27.12.16 במכללה ע"ש הרמלין בנתניה

**הקדמה**

המטרה העיקרית של כנס המורים השנתי היא לספק הזדמנות למפגש חברתי ומקצועי, לאפשר למורים לחלוק את המומחיות המקצועית שלהם עם עמיתים למקצוע, להתחבט בצוותא בשאלות מקצועיות, ולעודד שיתוף פעולה חברתי ומקצועי בתוך הקהילייה של מורי מדעי המחשב והנדסת תכנה.

בנוסף להרצאות מוזמנות ותערוכת חומרי לימוד יינתן בכנס מקום להצגת הצעות שיוגשו על ידי מורים. ההצגה תוכל להתבצע במסגרת הרצאות קצרות, סדנאות, דיוני שולחן עגול ופוסטרים בכל אחד מנושאי הכנס.

**פרטים נוספים על הכנס יפורסמו באתר.**

אנו מזמינים את כל העוסקים בהוראת מדעי המחשב והנדסת תכנה להגיש הצעות לכנס. ניתן להגיש הצעה בכל אחד מהנושאים המופיעים בהמשך. **המועד האחרון להגשת הצעות 1.10.16**

מחבר של הצעה שהתקבלה חייב להציגה באופן אישי. אם אינו יכול לעשות זאת, עליו ליצור קשר עם הועדה המארגנת כדי לתאם מחליף. שימו לב: רק המחבר הראשון ישובץ כך שמושבו לא יתנגש עם מושבים אחרים שלו. המציג יהיה חייב להירשם לכנס. הצעה שלא תעמוד בהנחיות ההגשה, תידחה על הסף.

**נושאי הכנס**

ניתן להגיש הצעות לכנס בכל אחד מהנושאים הבאים: (אפשר לקבל רעיונות גם באתרי הכנסים הקודמים)

1. רעיונות הוראה.
2. מחקר בתחום למידה/הוראה של מדעי המחשב והנדסת תכנה.
3. תכנון לימודים של יחידה או תחום תוכן, תכנון כיתתי או בית ספרי, מסגרות למידה מיוחדות (למשל, חוגי הורים ותלמידים).
4. שיטות הוראה: שיטות הוראה חלופיות, סגנונות הוראה.
5. הערכת הישגים של תלמידים: שיטות הערכה חלופיות, הערכה עצמית של לומדים, הקשר בין הערכה ללמידה והוראה.
6. תהליכים קוגניטיביים: יכולת ואינטליגנציה, שונות של לומדים, תהליכי חשיבה של תלמידים.
7. סביבות למידה: סביבות שיתופיות, בינתחומיות, מתוקשבות.
8. הוראת מדעי המחשב והנדסת תכנה לאוכלוסיות מיוחדות (למשל, מחוננים).
9. הוראת מדעי המחשב והנדסת תכנה במוסדות על-תיכוניים (למשל, מכללות להכשרת מורים).
10. הכשרת מורים: הכשרת פרחי הוראה, השתלמויות מורים.

**הגשת הצעות לכנס**

ניתן להגיש הצעות להרצאה (20-30 דקות), סדנה (30-60 דקות), שולחן עגול (30-60 דקות), או פוסטר בכל אחד מנושאי הכנס. ההנחיות להגשת ההצעות מופיעות בהמשך. כל ההצעות שתתקבלנה תפורסמנה בחוברת הכנס.

**מה צריך להגיש? (עד לתאריך 1.10.16)**

- **ההצעה:** תוגש בעברית ותהיה בהיקף של עד 1,000 מילים (להוציא מקורות ביבליוגרפיים), בגופן (פונט) David או Times New Roman, בגודל 12, על נייר A4 (11״ x 8.5״), רווח של שורה וחצי. בשורה הראשונה יופיע שם ההצעה ממורכז ומודגש. בשורה השנייה, שם/שמות המחבר(ים). שם המרצה המציג יסומן בקו. בשורה שלישית כתובת דוא״ל.
- **טופס נלווה להגשת הצעה** – יש להגיש לכל הצעה בנפרד.
- **אופן ההגשה:** רצוי לשלוח את ההצעה והטופס הנלווה בדואר אלקטרוני לכתובת

lapidot@tx.technion.ac.il

אם אין לכם גישה לדוא״ל,
ניתן לשלוח את ההצעה והטופס הנלווה
לפקס 04-8293004

**טופס נלווה להגשת הצעה לכנס**

נבקשך למלא טופס זה ולצרף אותו אל ההצעה.

**תאריך אחרון לקבלת החומר: 1.10.16**

שם המרצה:

כתובת:

טלפון:

טלפון נייד:

דוא״ל:

מקום עבודה:

כותרת ההצעה:

מחברים נוספים:

ההצעה מוגשת להצגה במסגרת הבאה:
(נא לסמן בתיבה המתאימה)

[ ] מושב הרצאות קצרות [ ] סדנה

[ ] דיון שולחן עגול [ ] הצגת פוסטר

נא לסמן את התחום המתאים ביותר להצעה:

| | תחום | עזרים (נא לסמן כל מה שדרוש): |
|---|---|---|
| 1 | רעיונות הוראה | [ ] לוח |
| 2 | מחקר למידה/הוראה מדהמ״ח והנדסת תכנה | [ ] מטול שקפים |
| 3 | תכנון לימודים | [ ] מחשב |
| 4 | שיטות הוראה | [ ] מקרן למחשב |
| 5 | הערכת השגים של תלמידים | [ ] אחר (פרט) |
| 6 | תהליכים קוגניטיביים | |
| 7 | סביבות למידה | |
| 8 | אוכלוסיות מיוחדות | |
| 9 | מוסדות על תיכוניים | |
| 10 | הכשרת מורים | |

במידה וההצעה תתקבל, אני מתחייב/ת להציגה בכנס.

_____ _____

תאריך חתימה

# סביבות למידה ויזואליות למתחילים בלימוד תכנות

**ד"ר תמי לפידות**

**חינוך מדעי טכנולוגי, הטכניון**

בשנים האחרונות פותחו סביבות תכנותיות רבות שכולן נועדו לתלמידים מתחילים והמפתחים שלהם מצהירים כי הן אמורות להקל על הוראת תכנות למתחילים. המאמר יבחן מספר סביבות כאלה וישים דגש על 3 סביבות פופולריות שפותחו בעשור האחרון: סביבת Alice, סביבת Scratch, וסביבת Greenfoot.

## סוגים של סביבות למידה משולבות

בספרות המקצועית ניתן למצוא מיונים שונים של סביבות למידה משולבות.

כך, למשל, רונגס ועמיתיו (Rongas et al., 2004) מציעים חלוקה לארבע קטגוריות: סביבות פיתוח משולבות IDE, סביבות למידה וירטואליות, ויזואליזציות[1], ומערכות להגשה ניהול ובדיקת תרגילים. במסגרת החלוקה הזו, הם מתייחסים לתכנות כאל תהליך של פתרון בעיות ומתייחסים באופן שונה לדרישות שיש לכלים האלה מצד מתחילים או מומחים. הם טוענים כי IDE הם כלים המתאימים למומחים (כך, למשל, היא סביבת אקליפס) ואילו סביבות למידה וירטואליות מתאימות למתחילים (כמו, למשל, סביבת Bluej).

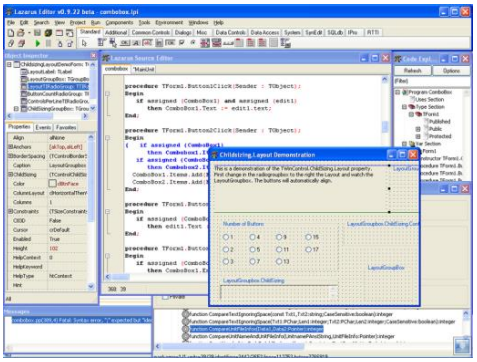
קלר ופאוש ערכו מיון מקיף של למעלה מ- 50 מערכות (Kelleher and Pausch 2005) ויצרו טקסונומיה[2] היררכית המאורגנת לפי מטרות המערכת (הוראה או שימוש בתכנות) ולפי ההיבט התכנותי הראשי שאותו מנסה המערכת להפוך לפשוט יותר.

IDE הם ראשי התיבות של סביבת פיתוח משולבת (Integrated Development Environment).

באופן כללי, סביבת פיתוח משולבת היא אוסף של כלי תכנות המשולבים באותה סביבה. לדוגמה, ברוב שפות התכנות יש כלים שונים כמו עורך טקסט, קומפיילר, ו-debugger שכל אחד מהם הוא תוכנית נפרדת אבל הם כולם מופעלים באותו מקום.

סביבת פיתוח משולבת מסייעת למתכנתים בפיתוח תוכנה, מקלה על תהליך התכנות והופכת אותו לפחות מסורבל.

קיימות סביבות פיתוח משולבות שהן טקסטואליות בעיקר (כמו טורבו פסקל) וסביבות פיתוח ויזואליות או גרפיות (כמו ויז'ואל סטודיו, או דלפי) שבהן ניתן לבצע חלק מכתיבת הקוד באמצעות שימוש בעזרים חזותיים.



תמונת מסך של GPL/LGPL Lazarus project

לאזרוס היא סביבת פיתוח גרפית של קוד פתוח לפסקל

1. סביבות העושות שימוש בעזרים חזותיים
2. מיון וחלוקה לקטגוריות

הסביבות (Fincher and Utting, 2010) נקראות ILE בגלל שהמטרות שלהם (גם בפיתוח וגם בשימוש המצופה מהם) שונות מסביבות תכנות אחרות (שמכונות IDE).

Initial Learning Environments - ILE

Integrated Development Environment – IDE

בשנים האחרונות היו הרבה ניסיונות לפתח סביבות למידה התחלתיות ILE. בין אם דרך מניפולציות ישירות על עצמים (למשל סביבת BlueJ) ובין אם דרך ויזואליזציה של נתונים ומבנים (למשל סביבת JGrasp). המערכות האלה מספקות למתחילים סביבה הרבה יותר כללית וגישה פורמלית לתכנות.

פינצ'ר ואאוטינג (שם) קראו למאמר "מכונות למען חשיבה" בכוונה. כפי שבית הוא "מכונה" שגרים בה, כך אנחנו מאכלסים את הסביבה שבה אנו חושבים. סביבה שמתוכננת לתמוך במחייה עושה את הדברים לקלים יותר, נוחים ומהנים.

כל הסביבות שנתייחס אליהן בהמשך פותחו במטרה ליצור סביבה אטרקטיבית ומעניינת שתסחוף תלמידים להתעניין בחשיבה של computational thinking בשלב מוקדם ככל האפשר.

בנוסף להתמקדות שלהן בשלבי למידה התחלתיים, לסביבות האלה יש מכנה משותף – הם תוצרים של המיומנויות שהם רוצים ללמד. הרציונל והגישה מוכלים בכל הסביבה. כל אחד מהם ייחודי ותפור לפי צרכי הלומד. אבל לכל אחד יש תיאוריה שונה וייחודית על הוראה ולמידה של תכנות.

סביבת אליס: מבוססת על מוטיבציה של הלומדים. אם הלומדים יכולים להיות מעורבים בתכנות אזי המוטיבציה תבוא מתוך הרצון שלהם להמשיך במשימת התכנות.

סביבת סקרצ': התוצר יותר חברתי. אם הלומדים יכולים לשתף ולהציג את הישגיהם אזי הם ירצו להמשיך ללמוד. למידה עם סקרצ' היא בעיקר בסביבה לא פורמלית.

סביבת גרינפוט: הלומדים יתעניינו בבניית משחקים, אבל הם יעשו זאת תוך כדי עבודה בשפה אמיתית ג'אווה. מבוסס על גישה פדגוגית של עצמים תחילה. אם לומדים לתכנת נכון אז ילמדו טוב ותפקיד המורה הוא להציג להם עבודה מתוכננת היטב שהם יוכלו לעבוד איתה).

Sally Fincher

Ian Utting

## שושלות משפחתיות של פיתוח

כל אחת מהסביבות שיתוארו בהמשך ממוקמת במקום אחר לפי גזדיאל (Guzdial, 2004) – סקרצ' היא מצאצאי לוגו (גם בגישה וגם בתחושה), אליס הוא עולם זוטא rule-based, וגרינפוט בקבוצה של השפות האמיתיות.

גזדיאל (שם) מתאר שלוש שושלות משפחתיות (genealogies) של פיתוח, שלכל אחת מהן יש גישה שונה ביחס לשאלה "מה הופך את התכנות לקשה עבור מתחילים?" – לוגו וצאצאיה, סביבות מבוססות חוקים (rule-based), ושפות רגילות.

Mark Guzdial

שושלת לוגו

משפחת לוגו – פותחה באמצע שנות ה- 60 על ידי Wally Feuzeig and Danny Bobrow יחד עם Seymour Papert ותוכננה כ"ליספ ללא סוגריים". כאשר פותח, לא ידעו שתכנות קשה ולכן המפתחים שאלו "למה חשוב שתלמידים יתכנתו?" התשובה שלהם הסתמכה על הגישה הפיאז'טנית ללמידה. המטרה היתה שתלמידים יחשבו על החשיבה של עצמם בעזרת התכנות. בהמשך שאלו "עבור איזה

משימות ירצו התלמידים לתכנת?" או "איזה תחומים ירצו ללמוד דרך תכנות?"

smalltalk-72 פותח על ידי Alan Kay, Dan Ingalls, Adele Goldberg והרחיב את לוגו בכמה כיוונים – קיי חשב שהכח החישובי של לוגו אינו מספיק ולכן פיתח שפה מונחית עצמים. סביבת העבודה של לוגו (command line) לא הספיקה ולכן המציא את desktop. למרות שנוסה בהצלחה עם מתחילים, הגרסאות הבאות יועדו למומחים והעבודה עם מתחילים נעלמה מהשטח ל- 15 שנים עד שפותח Squeak.

משפחת הסביבות מבוססות-חוקים

הבונים של שפות מבוססות חוקים rule-based יצאו מנקודת הנחה שסוגי תכנות מסוימים הם קשים למתחילים ויהיה להם יותר קל לתאר את מצב העולם על-ידי חוקים או יחסים. הסביבה הראשונה שפיתחו היתה כשפה בת של lisp ובעקבותיה באו הרבה סביבות תכנות למתחילים (פרולוג, לוגו ו- smalltalk-72).

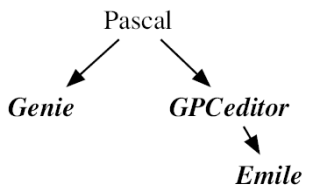
מספר חוקרים פיתחו סביבות תכנות לא-טקסטואליות המבוססות על rule-based programming (במקום על תכנות פרוצדורלי או אימפרטיבי). תלמידים העובדים בסביבות האלה מתארים מצבים בעולם במקום להגיד למחשב איך לפעול בעולם הנתון. שפה כזו היא פרולוג.

שפות "אמיתיות"

הבונים של "השפות האמיתיות" האמינו שהמתחילים זקוקים לסביבה מתווכת שתסייע להם. הבעיה היא לא בשפה אלא בגישה הפדגוגית. ולכן, הם ניסו לא לשנות את השפה אלא לספק למתחילים כלים שיסייעו להם.

במענה לשאלה "מה קשה בתכנות" אפשר לנקוט בגישה אחרת. שימוש בשפות "חינוכיות" פוגמת במוטיבציה של התלמידים כי זו לא שפה אמיתית. ולכן הבעיה היא בסביבה וצריך לבנות אותה כך שתתמוך בלומדים המתחילים.

למשל, לפסקל נבנו הסביבות התומכות הבאות:




לסיכום, גזדיאל טוען שבמהלך 40 שנים של פיתוח סביבות ללומדים מתחילים רק התחלנו לגרד את השאלה מה קשה בתכנות. ולכן, אולי הבעיה לא היתה בסביבות אלא בתכנית הלימודים וצריך לחשוב על דרכי הוראה אחרות.

## סביבת Greenfoot

סביבת Greenfoot פותחה על-ידי Michael Kölling וצוות BlueJ. היא פורסמה בשנת 2006.

הסביבה עושה שימוש בגרפיקה דו-ממדית, אבל התוכניות הן בג'אווה. פותח למצבי למידה יותר פורמליים (למשל, בבית הספר), עם תמיכה של מורה. יש ספר לימוד ותמיכה למורים דרך "החדר הירוק" (The Greenroom).

אתר

http://www.greenfoot.org/

באתר גרינפוט הסביבה מוצגת כך – גרינפוט הוא שילוב בין IDE לסביבת עבודה של משימות ג'אווה שמטפלות ב- grid דו ממדי המתאימות למתכנתים מתחילים. בעוד גרינפוט תומכת בשפת ג'אווה במלואה, היא שימושית במיוחד לתרגילי תכנות שמעורב בהם אלמנט חזותי. דוגמאות פופולריות למשימות: חידת 8 המלכות, קארל הרובוט, משחק החיים (Life), גרפיקת הצב.

הסביבה אחראית לשני דברים עיקריים: שיהיה קל לייצר ייצוגים גרפיים לעצמים, שליטה על הביצוע של לולאת סימולציה.

Michael Kölling

גרינפוט היא סביבת פיתוח משולבת (IDE) חינוכית שנועדה ללימוד תכנות למתחילים בני 14 ומעלה (מתאים גם לתלמידי מכללה ואוניברסיטה). הסביבה משלבת גרפיקה ופלט אינטראקטיבי עם תכנות בג'אווה.

תיאור הסביבה:

פרויקט בסביבה הוא תסריט (scenario). "שחקנים" (actors) הם עצמים שמוצגים בעולם ויש להם התנהגות שמאפשרת את ביצוע התסריט.

השפה שנעשה בה שימוש היא שפת ג'אווה סטנדרטית אבל הסביבה תומכת בשימוש פשוט בשפה. לדוגמה, אין צורך לכתוב שיטה ראשית סטטית ואפשר להתחיל לתכנת עם שיטות פשוטות של שורה אחת. מיד לאחר שכותבים מחלקה, ניתן לבצע קומפילציה ולעבוד אינטראקטיבית עם העצמים (אין צורך להשלים פרויקט שלם).

לכל השחקנים בעולם יש מחלקה act (יורשת מהמחלקה Actor superclass). לחיצה על כפתור act (אחד מלחצני הבקרה) מביאה לביצוע של השיטה על ידי כל אחד מהעצמים בעולם. לחיצה על כפתור Run גורמת להפעלה חוזרת של act עד שעוצרים את הביצוע. כך מתבצעים התסריטים בגרינפוט.

המתכנת מגדיר את ההתנהגות של כל שחקן בתוך השיטה act והסביבה מבטיחה שכל שחקן יפעל בהתאם והאנימציה תתבצע אוטומטית. לכן, תלמידים יכולים להתרכז בתכנות ההתנהגות ההגיונית של השחקנים ולא צריכים להתעסק בכתיבת קוד גרפי.

אחת המטרות של גרינפוט היא תכנון שימחיש באופן ברור תכנים חשובים של תכנות מונחה עצמים. תרגיל ראשון טיפוסי יהיה להפעיל תסריט שכבר קיים בסביבה (הוכן מראש על-ידי המורה).

מאפיינים נוספים: אם לשיטה יש פרמטרים, נפתח חלון דיאלוג. ניתן ליצור הרבה עצמים מאותה מחלקה. ניתן לבדוק את המצב של כל שחקן בעזרת כלי Inspect (בעזרת חלון שמראה את כל המשתנים הפנימיים וערכיהם) וזה ממחיש את העצמאות של העצמים ואת הקשר שלהם עם המחלקה שלהם (לכל העצמים של המחלקה יש אותם סוגים של שדות).

מבחינה פדגוגית, האינטראקציות האלה חשובות כי הן מאפשרות למורה להדגים מושגים מרכזיים בדרך שניתנת להבנה בקלות.

המושגים האלה כוללים:

- תוכנית מורכבת מאוסף מחלקות.
- בעזרת מחלקות ניתן ליצור עצמים.
- ניתן ליצור מספר עצמים מאותה מחלקה.
- לכל העצמים של אותה מחלקה יש אותן שיטות ואותם שדות.
- לכל עצם יש את הערכים שלו. לכל עצם יש מצב עצמאי.
- אנחנו מתקשרים עם עצמים על ידי קריאה לאחת מהשיטות שלהם.
- לשיטות יכולים להיות פרמטרים. שיטות יכולות גם להחזיר ערך.
- לפרמטרים ולערכים מוחזרים יש טיפוס.

כל המושגים האלה הם מרכזיים בהבנת תכנות מונחה עצמים והם קשים להוראה. אחת הסיבות לקושי בהוראתם היא שהמושגים האלה מופשטים. גרינפוט הופכת אותם לקונקרטיים על-ידי תרגילים מוחשיים, והמחשה ויזואלית שלהם. זה מאפשר את הוראת המושגים המרכזיים תחילה, לפני הצורך להתעסק בתחביר.

נביא דוגמאות לקוד טיפוסי בגרינפוט.

כאשר יוצרים מחלקה חדשה של שחקנים, כל מחלקה מקבלת דמות (ניתן לבחירה מתוך ספריה של דמויות או מתוך תמונות של המשתמש). עצמים שנוצרים ממחלקה מסוימת מתחילים עם הדמות של המחלקה

שלהם. לכל שחקן יש 3 אלמנטים של מצב שמוצגים אוטומטית על המסך – מיקום position, כיוון rotation, ודמות image. כל אחד מהם ניתן לשינוי דרך קריאה לשיטות כדי ליצור אפקט של אנימציה.

למשל setRotation(90);

מסובב את העצם ב- 90 מעלות.

השיטה getRotation()

מחזירה את הכיוון הנוכחי 0-359 במעלות. כך שניתן בקלות לגרום לשחקן להסתובב סביב הציר שלו בעזרת setRotation( getRotation()+2 );

כאשר ההוראה הזו מוכנסת לגוף השיטה act, ביצוע התסריט יגרום לביצוע חוזר ולכן השחקן יסתחרר סביב עצמו.

כאשר יוצרים שחקנים חדשים, נוצרת באופן אוטומטי תבנית של מחלקה שלמה אבל פשוטה שנראית כך

```
import greenfoot.*;
/**
 * Write a description of class MyActor here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class MyActor extends Actor
{
  /**
  * Act - do whatever the actor wants to do.
  */
  public void act()
  {
     // Add your action code here.
  }
}
```

מטרות הפיתוח שנביא בעמוד הבא השפיעו על פיתוח גרינפוט.

כאשר חושבים על קלות השימוש, השיקולים הכי חשובים אינם קשורים לאלמנטים שיכנסו לסביבה אלא דווקא למה שלא יכנס לסביבה.

ניתן להשיג קלות בשימוש רק על ידי הגבלת התחום של המערכת. במקרה של גרינפוט, למרות שהיא עושה שימוש בשפה קיימת (ג'אווה) הוחלט לא להכניס לסביבה כלים שנהוג למצוא ב- IDE אחרים כמו ניהול גרסה, או בדיקת יחידות. המטרה היא לאפשר ללומד להכיר את כל ממשק המשתמש במספר קטן של ימים. גם בתכנון ה- API הוחלט לא להכניס את הכל. גרינפוט מנסה ליצור איזון בין גמישות (לתת למשתמש מספיק כח כדי ליצור תסריטים מעניינים) לבין פשטות (הגבלת מספר השיטות ב- API לאוסף שניתן ללמוד אותו בקלות).

במבט ראשון גרינפוט נראית כמו עולם-זוטא סטנדרטי (כמו גרפיקת הצב של לוגו או קארל הרובוט), בנוי לתוך IDE עם מעגל חוזר של עריכה-קומפילציה-הרצה (edit/compile/execute). אבל זה הרבה יותר גמיש ממה שנראה. ניתן לחשוב על גרינפוט כמסגרת-על של עולם-זוטא ( a micro-world meta framework) שמאפשר יצירה מהירה וקלה של עולמות זוטא. קל לבנות את גרפיקת הצב או קארל <u>בתוך</u> גרינפוט, אבל גרינפוט יכולה להגיע ליותר מהמגבלות של הסביבות האלה.

סביבות מסורתיות משלבות יכולות תכנות (כמו זוז, הסתובב) עם תסריט ספציפי (למשל רובוט שאוסף זמזמים). כתוצאה מכך, אם תלמיד מתחיל עם רובוט וזמזמים אז הוא ישאר באותה סביבה גם אחרי מספר שבועות. אם תלמיד מתעייף מהרובוט או לא מעוניין בו מלכתחילה, אין לו אפשרות להחליף סביבה. גרינפוט מפרידה בין שני היבטים: יכולת התכנות מופרדת מתסריט ספציפי, מה שמאפשר יישום של מגוון רחב של דוגמאות. זה תומך במטרות של הנאה והנעה וגמישות. כתוצאה מכך ניתן ליצור הרבה תסריטים כמו משחקים, סימולציות, מוזיקה, ועוד.

בתכנון הסביבה היו שתי מטרות-על שכל אחת מהן משליכה על מספר מטרות מפורטות יותר:

תמיכה בלמידה

מנקודת המבט של התלמיד,
המטרה היא להפוך את התכנות למהנה,
יצירתי ומשביע רצון.

בעקבות המטרה הראשונה, הוגדרו תת-המטרות הבאות:

1. שימוש נוח: המערכת חייבת להיות נוחה בשימוש, וצריכה לאפשר לתלמידים להשיג את מטרתם בלי להיתקע עם משימות אדמיניסטרטיביות.
2. גילוי עצמי: המערכת צריכה לאפשר לתלמידים לגלות בקלות את כל מה שהם זקוקים לו לצורך השגת המטרות שלהם.
3. סביבה שובת-לב: המערכת צריכה לאפשר עבודה אטרקטיבית ושובת לב, לאפשר שימוש נוח בגרפיקה, אנימציה וקול.
4. גמישות: המערכת צריכה להיות גמישה מספיק כדי לספק מגוון רחב של תסריטים שיתאימו לכל גיל ולכל תחום עניין.
5. משוב מיידי: המערכת צריכה לספק משוב מיידי. כלומר, היא צריכה לאפשר פיתוח/עבודה בצעדים קטנים עם הזדמנויות תכופות לביצוע, צפייה ומשוב חזותי.
6. נגישות: המערכת צריכה להיות נגישה ולהתאים לעבודה על מערכות שונות, בעיקר כדי לאפשר לתלמידים לעבוד עליה בבית (הדרישה הזו נתפסת כחלק מרכזי בלמידה יצירתית וחקרנית).
7. שיתוף ואינטראקציה: המערכת צריכה לתמוך בעבודה משותפת ובתקשורת בין לומדים, שכן מאפיינים כאלה של עבודה שיתופית יכולים להוות גורם מניע משמעותי ביצירתיות.
8. הרחבות: יכול להועיל אם המערכת תהיה ניתנת להרחבות, כך שתוכל לתקשר עם מערכות אחרות (למשל, אמצעי חומרה נפוצים של משחקים, שרתי אינטרנט, ומסדי נתונים).

תמיכה בהוראה

מנקודת המבט של המורה,
המטרה היא שהסביבה תסייע בהוראת מושגי
תכנות אוניברסליים חשובים.

בעקבות המטרה השנייה, הוגדרו תת-המטרות הבאות:

1. ויזואליזציה: מושגים חשובים של תכנות [במקרה שלנו מונחה עצמים] צריכים להיות מוצגים בבירור בצורה חזותית.
2. אינטראקציה: אינטראקציות בסביבה צריכות להמחיש עקרונות תכנות כמו instantiation או קריאה לשיטות.
3. מודל מנטלי עקבי: כל הייצוגים של עקרונות בסביבה צריכים לשקף באופן נכון את המודל התכנותי כדי שתלמידים יוכלו להגיע למסקנות מתאימות בעקבות ההתנסויות שלהם.
4. עקרונות חשובים יותר מתחביר: תחביר לא יכול להוות מכשול בשלבים הראשונים של העבודה ואסור שתחביר יפגע בסיכוי ההצלחה של מתחילים.
5. הימנעות מעומס קוגניטיבי: מערכות פיתוח מודרניות רבות הן מורכבות ומקשות על מתחילים. התיאוריה של עומס קוגניטיבי מלמדת אותנו שהיכולת של עיבוד מנטלי היא מוגבלת ושמספר האתגרים הקוגניטיביים שאנחנו עומדים מולם במקביל משפיע מאד על יכולת הלמידה שלנו. כדי לשמור על העומס הקוגניטיבי ברמה סבירה צריך להימנע ממורכבות בלתי רלבנטית (כלומר, מורכבות שאינה הכרחית או פנימית למשימת הלימוד) וכל המשימות האחרות צריכות להיות אוטומטיות ונסתרות.
6. תמיכה במורים: יש צורך במסגרת שתתמוך במורים, כמו מערכת שיתוף פעולה של דיונים והחלפת חומרים.


סימולציה של פסנתר


סימולציה של מעלית

לאפשרות הזו לכסות מספר גדול של דוגמאות או הקשרים עם אותה מסגרת יש מספר יתרונות. ניתן לתפור דוגמאות עבור גילאים שונים או תחומי עניין שונים. אם תלמידים מעוניינים במשחקים הם יכולים להתחיל עם בניית משחק. אם תלמיד אחר מעוניין במשהו יישומי, הוא יכול לעבוד עם סימולציות. תלמידים שאוהבים מדע יכולים לבחור סימולציות מתחום הביולוגיה, כימיה או פיסיקה – והמורה יכול להתאים בקלות דוגמאות לפי תחומי העניין של התלמידים.

גישת הוראה טיפוסית להתחיל בסביבת גרינפוט היא לתת לתלמידים תסריט מוכן חלקית ולתת להם להשלים או לשנות אותו. תלמידים מתחילים בדרך כלל עם הרצה והתנסות עם החומרים הנתונים ואז יצירת הרחבה ותוספות.
חשוב שלמורים יהיה קל לבנות וליצור תסריטים. תסריט התחלתי טיפוסי כולל 40-70 שורות קוד. תסריטים מתקדמים ומתוחכמים יותר כוללים כמה מאות שורות קוד. למשל, מימוש מלא של משחק האסטרואידים כולל 620 שורות קוד. כמובן שניתן להשתמש גם באוסף התסריטים המוכנים, אבל אם מורה רוצה ניתן לעשות זאת בקלות. הכנת תסריטים מוכנים חשובה רק בשלבי למידה ראשונים כי אחר כך תלמידים כבר יבנו בעצמם.

קהל היעד של גרינפוט הוא תלמידים בני 14 ומעלה. בעוד שהפלט של גרינפוט הוא ויזואלי, התכנות הוא טקסטואלי (בניגוד לסקרצ' או אליס) ולכן הוא דורש יותר מיומנות. ניתן להשתמש בגרינפוט להתחלה או אחרי אליס/סקרצ'. המושגים שנלמדו בסביבת אליס או סקרצ' יעברו בקלות לגרינפוט, אבל כאן אפשר ללמוד גם מושגים נוספים.
המטרה של שימוש בגרינפוט עם מתחילים דומה לזו שבאליס או סקרצ'. כלומר, יצירת עניין והתלהבות בתכנות והעברת עקרונות תכנות בסיסיים. אבל גרינפוט מאפשר גם העברת עקרונות מתקדמים של פיתוח תוכנה כמו תכנון נכון של מחלקות, הכמסה, רמות הפשטה.

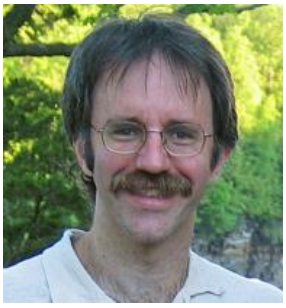
השפה שנעשה בה שימוש היא ג'אווה. יש לזה השפעה על הרבה מאפיינים, חלקם תורמים, וחלקם יוצרים מגבלות. ראשית, שימוש בשפה טקסטואלית יותר קשה ללומדים צעירים ולכן מומלץ לא להשתמש בגרינפוט לפני גיל 14. ההיבט השני מתייחס לג'אווה – התחביר שלה מקשה וגם הודעות השגיאה גרועות. אבל עדיין יותר קל לעשות את זה בגרינפוט ולא ישירות בג'אווה. מצד שני, יש יתרונות לשימוש בשפה אמיתית כך שמתאים לשימוש גם בגילאי אוניברסיטה, וניתן ליישם רעיונות יותר מתקדמים.

נעשה הרבה שימוש בגרינפוט בבתי ספר תיכוניים, מכללות ואוניברסיטאות. הקורסים נעים בין יום אחד לבין סמסטר שלם. בזמן כתיבת המאמר היו מעל 800 חברים בחדר הירוק שרובם מורים שמלמדים את הסביבה. כשני שליש מהם מורים בתיכון והיתר במכללות ואוניברסיטה.
יש הרבה משותף לגרינפוט עם אליס וסקרצ', כמו עידוד לחקירה יצירתית, התנסות, גילוי, אינטראקציה חברתית. בכל השלוש יש שימוש בגרפיקה וצליל כגורמים מוטיבציוניים, תוך כדי הוראת עקרונות תכנות.

לפי הטענה של מחברי גרינפוט, יש הבדל בגיל של קבוצת היעד. סקרצ' מתאים לגילאים 8-16, אליס לגילאים 12-19, גרינפוט לגילאים 14+.

## סביבת Alice

Randy Pausch



Stephen Cooper

סביבת אליס פותחה על ידי Randy Pausch באוניברסיטת קרנגי-מלון ופורסמה בשנת 2000. הסביבה מאפשרת למידה חוויתית של מושגי תכנות בסביבה ויזואלית תלת ממדית, תוך כדי יצירת סיפור, משחק, או סרטון וידאו.

עצמים תלת ממדיים (אנשים, בעלי חיים או כלי רכב) מאכלסים עולם וירטואלי והתלמידים יוצרים תוכנית שמבוססת על אנימציה של העצמים האלה. התכנות מתבצע בעזרת אריחים גרפיים (graphic tiles) אותם גוררים על המסך. אליס מאפשרת לתלמידים לקבל משוב חזותי מיידי (איך התוכניות שלהם רצות) ומאפשרת להם להבין בקלות את היחסים בין הוראות בתכנית לבין התנהגות העצמים. על-ידי תיפעול העצמים תלמידים רוכשים ניסיון עם כל המבנים התכנותיים שנילמדים בקורס מבוא לתכנות.

אתר

http://www.alice.org/


המאמר של קופר (Cooper 2010) בודק את ההחלטות שהתקבלו בפיתוח אליס.

חשוב להבחין בין 2 סוגי החלטות: פיתוח הכלי ופיתוח הפדגוגיה (הוראה בעזרת הכלי). היה יותר חשוב להם בפיתוח הסביבה להחליט מה המספר המינימלי של שיטות שיספקו כדי שיהיה ניתן לבצע תנועת עצמים ופחות עניין אותם לחשוב על הוראות שיטפלו במשתנים.

מאפייני הפיתוח

1. ויזואליזציה של תכנית: המטרה היתה להפוך כמה שיותר מהמצב state למשהו ויזואלי עבור התלמיד. במובנים רבים, השיקול הפדגוגי החשוב ביותר של פיתוח אליס היה לבחור במערכת של ויזואליזציה של תוכנית. במערכות מסורתיות של ויזואליזציה של אלגוריתמים, המורה בונה אלגוריתם של אנימציה ותלמידים מתבוננים באנימציה המוכנה ואולי מכניסים כמה שינויים. רבים טוענים שתלמידים צריכים לתכנת ולא רק להסתכל איך המורה מתכנת או איך התוכנית המוכנה עובדת. בניגוד לתוכניות ויזואליזציה של אלגוריתמים, במקרה של ויזואליזציה של תוכניות מאפשרים לתלמידים ליצור אנימציות בעצמם. רמת המעורבות שלהם יותר גבוהה. ככלי לויזואליזציה של תוכניות, אליס מושפעת מאד מהגרפיקה של הצב של פפרט ומקארל הרובוט של פטיס. כמו עם צבים ורובוטים, באליס תלמידים רואים את הביצוע של התוכנית שלהם ויכולים לבדוק את ההשפעה הישירה של כל הוראה בתכנית.

2. אנימציה של שינוי מצב: התלמידים רואים את ביצוע התוכנית שלהם, הוראה אחרי הוראה ויכולים לראות את הקשר הישיר בין כל הוראה לביצוע שלה. בזכות עובדה זאת הם יכולים למצוא בקלות טעויות. מורים מדווחים שתלמידים יודעים בדיוק באיזו הוראה יש שגיאה בזכות האנימציה. במילים אחרות, במקום לעקוב אחרי ערכים של משתנים תלמידים עוקבים אחרי אנימציה של צורות.

3. מתן התנסות תכנותית רצינית לתלמידים כהכנה למדעי המחשב: מחקרים רבים מצביעים על קשיים של מתחילים. לכן, מטרה נוספת בפיתוח

של אליס היתה לספק סביבה עשירה שתאפשר לרכוש התנסות תכנותית משמעותית לתלמידים שימשיכו לקורס CS1.

4. הפרדה בין הבטים אימפרטיביים והבטים פונקציונליים: אחד ההיבטים המעניינים של אלגול היה ההפרדה בין הבטים אימפרטיביים ופונקציונליים. שיטות עושות רק אחת מהשתיים – או שהן שינוי מצב או שהן מחשבות ערך. אף פעם הן לא עושות את שני הדברים האלה ביחד. מבחינה פדגוגית, ההפרדה הזו מאפשרת לתלמידים להחליט מה הם רוצים להשיג בשיטה שהם כותבים ולבצע בהתאם. האם הם רוצים לחשב את מספר ההיפוכים שכדור צריך לבצע כאשר הוא מתקדם, או האם הם רוצים שארנב יקפוץ ויגיע לברוקולי. אליס 2 מיישמת את ההפרדה הזו. יש בה שיטות עם ערך מוחזר void, ופונקציות (שיטות עם ערך מוחזר לא void). הוראות אנימציה שיוצרות שינוי מצב לא יכולות להיכלל בתוך פונקציות. בנוסף, פונקציות לא יכולות לקרוא לשיטות.

5. שינוי מצב דרך הפעלת שיטות פרימיטיביות ולא דרך מניפולציה ישירה של מצב דרך משתנים: כאשר תלמיד רוצה להקפיץ ארנב הוא צריך רק להפעיל את השיטה bunny.hop() . ברור שברקע יש שינוי בערכיהם של המשתנים x,y,z שמייצגים את מיקומו של הארנב בחלל, אבל התלמיד לא צריך לדעת אפילו על קיומם של המשתנים האלה. בנוסף, אומצה הגישה הרווחת בתכנות פונקציונלי, שם עובדים עם פונקציות תחילה ורק בשלב מאוחר יותר (אם בכלל) מציגים משתנים. הואיל ומשתנים קשים לעבודה הוחלט להימנע משימוש בהם מבחינת התלמיד לפחות בהתחלה. לסיכום - אליס מספקת שיטות לשינוי מצב בלי התעסקות עם משתנים.

6. עורך Drag and drop מונע שגיאות תחביר. הצוות שפיתח את אליס חושב ששגיאות תחביר לא חשובות ואילו שגיאות סמנטיות הם הקשיים היותר חשובים ולכן צריך להתמקד בהם.

7. Object reification: אם מתחילים במחלקות כמו חשבון בנק זה יכול להיות קשה לתלמידים כיון שהם לא מכירים את הנושא. לכן, באליס קל יותר לראות את העצמים והמחלקות.

8. פתרון בעיות: סביבת אליס מאפשרת להתמקד בפתרון בעיות ולא בלימוד של שפה מסוימת.

9. תכנות בתוך הקשר: סביבת אליס משתמשת באנימציה, סיפור storytelling ובניית משחקים כהקשרים שבתוכם עושים את התכנות.

10. סיפור storytelling דרך יצירת אנימציות. בנוסף להקשר שבתוכו מתכנתים, סיפור סיפורים הוא מטפורה חזקה עבור תלמידים. הם מכירים ורגילים לסיפורים, זה חשוב להם ומהווה חלק מהתרבות שלהם.

11. דגש על תכנון. הוראה עם אליס שמה דגש על תכנון. אנימטורים עובדים עם storyboards ותכנות באליס עושה שימוש דומה בלוחות סיפור. הם משמשים מעין דיאגרמה לשינוי מצבים ומתאימים לתכנון אנימציה.

12. מעבר לג'אווה ושפות מונחות עצמים אחרות. קהל היעד ההתחלתי היה סטודנטים ולכן חשבו מראש על מעבר לשפות כמו ג'אווה. אליס 2 מספקת שתי תמיכות: א. כוללת את המבנים הטיפוסיים (עצמים, מחלקות, לולאות while ו- for, הוראות תנאי), ב. היכולת לעבור מאליס לשפה דמויית ג'אווה כך שהתלמידים יראו איך הקוד הזה נראה גם אם לא יכתבו קוד בעצמם.

13. קהל יעד: בפיתוח של אליס 2 קהל היעד היה סטודנטים במכללה לפני CS1. באותה תקופה (שנת 2000) הקורס CS0 עסק בשימושי מחשב. הניסוי הראשון של אליס היה במכללה. רק בשלב מאוחר יותר נכנסו לתיכונים וחטיבות בינים. הצוות שפיתח את אליס חושב שהכי טוב להשתמש באליס בגילאים 12-18 ותחילת מכללה. יש מעל 2,700 מורים שעובדים עם אליס וכמעט מחצית מהם דיווחו שהם מלמדים במערכת החינוך הרגילה.

נקודות חוזק וחולשה של סביבת אליס

1. מוטיבציה של תלמידים: תלמידים אוהבים לעבוד בסביבת אליס. מורים מדווחים שהתלמידים לא רוצים לעזוב את המעבדה.
2. משיכת קהל יעד רחב: אליס מושכת קהל רחב ומגוון כי גם בנות וגם בנים אוהבים לספר סיפורים, גם אם יש לסיפורים שלהם אופי שונה.
3. תלמידים מבינים את הנושא של העברת פרמטרים.
4. אינטואיציות של תלמידים ביחס לעצמים: תלמידים מקבלים תחושה טובה של מהו עצם. הם רואים עצם כישות מרובת מצבים (שניתנים לשינוי), תקשורת בין עצמים מתרחשת דרך הפעלת שיטות.
5. קל להשתמש.
6. תלמידים משתמשים יותר מידי בשיטת trial and error ניסוי וטעייה כאסטרטגיה של ניפוי שגיאות. ניתן להתמודד עם זה אם דורשים מהם לתכנן לפני כן.
7. בניית סצנה מול יצירה דינמית של עצם: יותר נוח להציב עצמים בעולם בגלל הגרפיקה התלת ממדית ופחות נוח לגרום לכך שהתכנית תציב את העצמים.
8. היררכית מחלקות רדודה: היררכית המחלקות באליס 2 היא בעומק של רמה אחת. כל המחלקות יורשות מ- Object. לא ניתן להעביר פרמטר מטיפוס פינגוין.
9. חוסר טיפול בחלק מהמושגים של תכנות מונחה עצמים: אין באליס 2 בנאים constructors ברורים והיא לא תומכת בפולימורפיזם.
10. קשה לעבוד עם מערכים ויזואליים ויש תמיכה רק במבנים של מימד אחד.

למרות שאליס תוכננה להיות סביבה לסטודנטים מתקשים במדעי המחשב, הרי שכיום היא נפוצה הרבה יותר, כולל בבתי ספר.

## סביבת Scratch

הסביבה פותחה על ידי קבוצת Lifelong Kindergarten Group הפועלת ב- Media Lab של MIT. היא פורסמה בשנת 2007.


סביבת Scratch מאפשרת למזג סוגים שונים של סרטוני מדיה (גרפיקה, תמונות, מוסיקה וצלילים) בדרכים יצירתיות וכך גם ניתן לה שמה. מקור השם Scratch הוא בפעולת הגירוד בה משתמשים די. ג'יי. שמסובבים בידיהם תקליטים אחורה וקדימה על מנת למזג יחד קטעי מוסיקה בדרך יצירתית.

Scratch נבנתה על בסיס שפת התכנות Squeak, בהשראת העבודה על Logo ו- Squeak Etoys אך במטרה להיות פשוטה יותר ואינטראקטיבית.

סקרצ' היא סביבת תכנות ויזואלית שמאפשרת למשתמשים (בגיל 8-16) ללמוד מדעי המחשב תוך כדי עבודה על פרויקטים בעלי משמעות אישית כמו אנימציה של סיפורים או משחקים. למשל, סיפורים עם אנימציה, משחקים, מהדורות חדשות, כרטיסי ברכה, דו"ח של ספר, וידיאו של מוזיקה, פרויקטים של מדע, או סימולציות.

התכנות בסקרצ' מתבצע בעזרת אבני בניין צבעוניות. בעזרת התכנות אפשר לשלוט בעצמים גרפיים דו-ממדיים שנקראים sprites הנעים על הבמה.

התכנון המקורי של סקרצ' הושפע על ידי הצרכים ותחומי ההתעניינות של ילדים בגילאי 8-16. לפעילויות של תיפעול מדיה סקרצ' הוסיפה תכנות וזה עודד את התלמידים ללמוד דרך חקירה ושיתוף פעולה. במקור, נעשה שימוש בסקרצ' רק בסביבות לא פורמליות כמו מרכזים קהילתיים, או חוגים

לאחר הלימודים. אבל לאחרונה יש שימוש בסביבה גם בבתי ספר.

הפרויקט של פיתוח סקרצ' התחיל ב- 2003 והסביבה פורסמה יחד עם האתר ב- 2007. מאז התבצעו מעל 2 מליון הורדות של התוכנה מהאתר ומעל מיליון פרויקטים הוטענו לאתר.

סקרצ' נבנתה על בסיס רעיונות הקונסטרקשיוניזם constructionist של לוגו ו- Etoys.

כדי לאפשר חופש פעולה מלא, סקרצ' מאפשרת לייבא או ליצור כל סוג של מדיה (תמונות, קולות, מוזיקה). האתר של סקרצ' מהווה מסגרת חברתית למשתמשים לשיתוף פרויקטים, קבלת משוב.

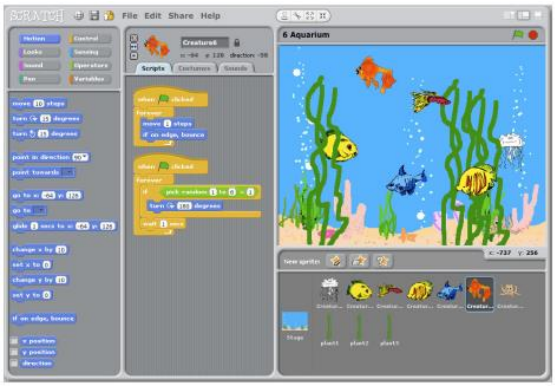
על פי המאמר של רזניק ומלוני (Resnick and Maloney, 2010) מטרת המפתחים של סקרצ' היתה להציג תכנות לתלמידים שאין להם רקע בתחום. המטרה הזו הכתיבה חלק גדול מהיבטי הפיתוח של הסביבה. חלק מההחלטות ברור מאליו, כמו הבחירה בשפה של בלוקים ויזואליים ובאוסף מינימלי של פקודות. החלטות אחרות פחות ברורות מאליהן, כמו הטיפול בשגיאות.

John Maloney

Mitchel Resnick

סקרצ' מעודדת לימוד עצמי

1. ממשק משתמש של חלון יחיד כדי לאפשר ניווט קל בסביבה. כל המרכיבים הנחוצים נמצאים על המסך. בחלון יש 4 חלקים עיקריים – השמאלי הוא לוח הפקודות עם כפתורים לבחירת קטגוריה, אמצעי לתסריטים של ה- sprite הנוכחי, החלק הגדול מימין למעלה הוא הבמה עליה מתרחש האקשן ומימין למטה אוסף ה- sprite שיש בפרויקט כדי שיהיה קל לעבור מאחד לשני.



ההוראות מחולקות ל- 8 קטגוריות, כל אחת בצבע אחר כדי להקל על המשתמש: בקרה, חיישנים, מספרים, משתנים, תנועה, מראה, צלילים ועט. הן מופיעות לפי סדר קושי ומה שקל יותר להבין נמצא בהתחלה.

2. הסביבה חיה ומעודדת שעשוע: זה מאפיין מפתח של סקרצ'. אין שלב קומפילציה או הפרדה בין עריכה להרצה. אפשר ללחוץ על פקודה או חלק מהתכנית בכל שלב כדי לראות מה יתבצע. אפשר גם לשנות פרמטרים או להוסיף בלוקים לתסריט תוך כדי ריצה. ולכן קל יותר לנסות דברים חדשים, לדבג, לשפר פרויקטים. עובדה זו גם מעודדת למידה hands-on ותומכת בגישת מלמטה-למעלה לכתיבת תסריטים (כאשר בודקים קטעים קטנים של קוד לפני שמשלבים אותם עם יחידות נוספות).

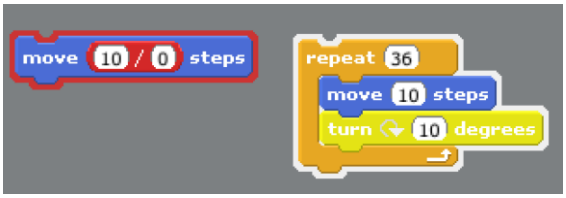
בלוקים של פונקציה מראים את הערך המוחזר בתוך בלון שיחה talk bubble.


כל בלוק מגיע עם ערכים התחלתיים שיקלו על התלמידים להבין ויש מסכי עזרה לכל הוראה.

לא צריך ליצור תסריט שלם לפני שמריצים ובניפוי שגיאות אפשר לפרק תסריט ארוך לחלקים ולבדוק כל חלק לחוד.

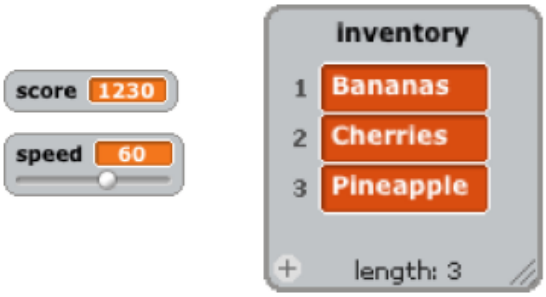
3. ויזואליות של הביצוע: סקרצ' מספק משוב ויזואלי. כאשר מתבצע תסריט, הוא מוקף גבול

לבן בוהק. אם יש שגיאה, הגבול נצבע אדום והבלוק שגרם לשגיאה מואר באדום.




4. אין הודעות שגיאה בדיוק כמו כשמשחקים באבני לגו – אין הודעות שגיאה. הבלוקים ניתנים לחיבור רק בדרכים מסוימות והמבנה שלהם. כך גם סקרצ'. אין שגיאות תחביר כי לא פשוט לא ניתן לבנות תסריט בדרך שגויה (הבלוקים לא יתחברו). כל הבלוקים מנסים להבין ולבצע משהו הגיוני גם כאשר יש שגיאה. למשל, בלוק set size תוחם את התחום של המספרים.

5. נתונים קונקרטיים: ברוב השפות הטקסטואליות, המשתנים בלתי נראים, מופשטים וקשים להבנה. כמו בוקסר, גם סקרצ' הפכה משתנים למשהו שניתן לראות ולעבוד איתם ולכן יותר קל להבין. בסביבת סקרצ' משתנה יופיע על הבמה כמוניטור. זה מאפשר לראות השפעה של הוראות כמו change x by 1 ומסייע להם לבנות מודל מנטלי של משתנים. אבל מוניטורים גם מסייעים לפרויקט (ניקוד במשחק למשל).

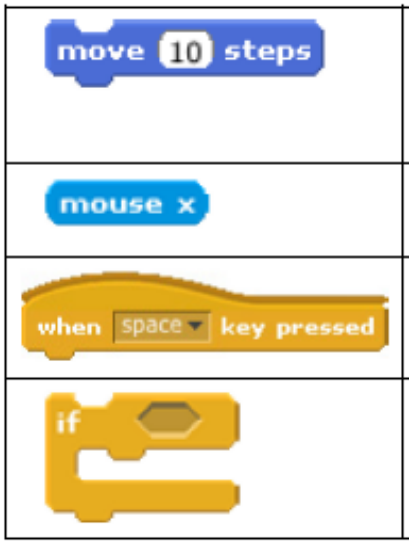



פרטים קטנים בתכנון יכולים להשפיע רבות. בגרסאות מוקדמות של סקרצ' לא ראו על הבמה משתנה חדש שרק נוצר. לאחר מכן שינו ועכשיו רואים כל משתנה – כך שהתלמידים הרבה יותר משתמשים במשתנים בפרויקטים.

6. אוסף מינימלי של פקודות: עשו מאמץ לקיים מינימום של פקודות למרות שהסביבה תומכת במגוון רחב של סוגי פרויקטים. הואיל וכל פקודה תופסת מקום על המסך ואם היו יותר הוראות היה קשה למצוא אותן. אחת מהדרכים להפחית במספר הבלוקים היא לאסוף יחד כאלה שהם רלבנטיים כמו drop-down menu של פונקציות מתימטיות.

שפת התכנות

1. תחביר: התסריט נוצר על ידי חיבור בלוקים. מבנה הבלוקים מרמז איך מחברים ביניהם. אם לא מתאימים, המערכת לא תיתן לחבר אותם (מה שמונע שגיאות תחביר). הדקדוק הגרפי ממלא את תפקיד התחביר. יש 4 סוגי בלוקים: פקודה, פונקציה, טריגר, מבנה בקרה.




כאשר מצרפים יחד בלוקים של הוראות (מחסנית stack), הבליטות והחריצים מתאימים זה לזה כמו חתיכות פאזל. המבנה של בלוקים של מבני בקרה מאפשר לעבוד איתם בקלות. בניגוד לשפות טקסטואליות, מבנה בקרה הוא יחידה שלא ניתן לחלק אותה. (וזה מקל על התלמידים)

בלוקים של trigger מקשרים בין אירועים events (כמו לחיצת עכבר או מקש) למחסניות שמטפלות באותם אירועים. למשל, כל המחסניות שמתחילות בטריגר של דגל ירוק מורצות כאשר המשתמש לוחץ על כפתור ההתחלה.

לחלק מהבלוקים יש חריצים המיועדים לפרמטרים. צורת החריץ מראה מאיזה טיפוס יהיה הפרמטר – מספר, מחרוזת, בוליאני וכדומה.

2. טיפוסי נתונים

לסקרצ' יש 3 טיפוסי נתונים – מספר, מחרוזת, בוליאני.

באופן כללי, המשתנים של סקרצ' יכולים להכיל מספר או מחרוזת. משתנה יכול להכיל נתון מכל סוג. סקרצ' יודעת לבצע המרה אוטומטית בין שני הטיפוסים, למשל להמיר את המחרוזת "123" למספר.

3. מודל העצם של סקרצ' sprites

ספרייטים הם עצמים – הם מהווים הפשטה של מצב (משתנים) והתנהגות (תסריטים). יחד עם זאת, לסקרצ' אין מחלקות או הורשה. זו שפה מבוססת עצמים (object-based) אבל לא מכוונת עצמים (object-oriented).

פקודות פועלות רק על ה- sprite שבו הם מופיעים. עצם לא יכול להפעיל תזוזה על עצם אחר למשל. למעשה, כל פקודה נמסרת או מתקבלת רק על ידי הספרייט שבו היא מופיעה.

לכל ספרייט יש סט עצמאי של תסריטים. לתכנון הזה יש tradeoff (עסקת חליפין). מצד אחד, קל להבין כי התסריטים מספקים את כל המידע על התנהגות הספרייט ולא צריך לחפש הורשה כדי להבין מה עושה הספרייט. מצד שני, יותר קשה לעבוד עם מספר ספרייטים שמתנהגים דומה (כמו אבני משחק). בפועל, התלמידים יוצרים ספרייט אחד ואז משתמשים בכלי stamp ליצירת עותקים. ואם צריך, משנים את התסריט.

4. תקשורת ושיתוף פעולה בין ספרייטים

ספרייטים לא יכולים לקרוא לתסריט של ספרייט אחר. המנגנון שהם משתמשים בו הוא מנגנון שידור. כל ספרייט יכול לשדר הודעה (מחרוזת). שידור מפעיל את כל התסריטים של כל הספרייטים שמתחילים בבלוק טריגר מתאים מהסוג

"when I receive <msg>"

יש שני יתרונות לזה שהספרייטים לא שולטים אחד על השני באופן ישיר. ראשית, כשמנסים להבין מדוע ספרייט עושה משהו (נניח הולך בדרך מסוימת), מרווח האפשרויות מוגבל לתסריטים שלו עצמו. שנית וחשוב יותר, הואיל וספרייטים הם עצמאיים, ניתן להעביר אותם בקלות בין פרויקטים שונים.

5. פרוצדורות

בגרסאות קודמות של הסביבה היה אפשר ליצור פרוצדורות. אבל בבדיקה בשטח התברר שתלמידים מתבלבלים בין פרוצדורות לשידורים (בשני המקרים מדובר על קישור בין שם לאוסף הוראות). כדי לשמור על פשטות ומינימליזם החליטו לבטל את הפרוצדורות והתברר שהתלמידים מסתדרים בלעדיהן.

אבל הפשטה פרוצדורלית היא רעיון רב-עצמה (powerful idea) של מדעי המחשב ולכן צוות הפיתוח(*) שוקל להחזיר פרוצדורות כך שמשתמשים יוכלו להגדיר את הבלוקים שלהם בעצמם (בלוקים של פקודות).

השוואה לאליס וגרינפוט

שלושתן מיועדות להציג תכנות למתחילים, ללא נסיון קודם, ולכן יש להן מטרות פיתוח דומות. למשל, בכל השלוש יש תמיכה בגרפיקה עשירה וצליל ושלושתן מאפשרות למשתמשים לבנות פרויקטים שמקושרים לתחומי העניין שלהם.

אליס וגרינפוט מיועדות לתלמידים מבוגרים יותר, מציגות תכנות class-based object-oriented ומדגישות את השפה ג'אווה או את העקרונות של ג'אווה. ולכן הן מתאימות להכנה לקראת בחינות AP (המקבילה האמריקאית לבחינות הבגרות). גרינפוט מאפשרת גם להרחיב את המערכת וללמוד תכנות מתקדם יותר.

אליס היא היחידה מהשלוש שתומכת בגרפיקה תלת ממדית.

(*)המאמר של רזניק ומלוני נכתב בשנת 2010

סקרצ' פונה לתלמידים צעירים יותר, מתמקדת בלמידה עצמית ומדגישה tinkerability.
בעוד שכל השלוש מאפשרות לייבא מדיה, לסקרצ' יש גם כלים לציור והקלטה.
הגרפיקה הדו ממדית של סקרצ' וגרינפוט קלה יותר ליצירה ועריכה מאשר המודל התלת ממדי של אליס.
סקרצ' מעודדת מגוון גדול יותר של פרויקטים מטיפוסים שונים וגם נוחה יותר בשיתוף פרויקטים דרך האתר.
סקרצ' משמשת לעיתים כמבוא לאליס או גרינפוט.
המעבר לגרינפוט חלק כי בשניהם גרפיקה דו ממדית וגם קל לעבור מהבמה והספייטים של סקרצ' לעולם של השחקנים בגרינפוט.

מסקנות
סקרצ' שואפת לעזור למשתמש לבנות אינטואיציות על תכנות תוך כדי בניית פרויקטים שמתאימים לתחומי העניין שלהם.
מבנה הבלוקים מונע התעסקות עם שגיאות תחביר ומאפשר להתמקד בבעיות מעניינות מייד מההתחלה.
סקרצ' מדגישה פשטות.
המערכת תמיד פעילה, בלי כפתור run/edit ואפשר להפעיל הכל בלחיצת כפתור.
יש המחשות ויזואליות קונקרטיות למשתנים ורשימות.
ניתן להתחיל לתכנת תוך 15 דקות אבל יש מספיק עומק ומגוון לשמור את התלמידים מעורבים.

קבוצת מדעי המחשב במחלקה להוראת המדעים במכון ויצמן עוסקת בפיתוח חומרי לימודים ל- Scratch ובמחקר על הלמידה של מדעי המחשב דרך הסביבה.
פרטים נוספים
http://stwww.weizmann.ac.il/g-cs/scratch/index.html

## דיון השוואתי בין שלוש הסביבות

הדיון על שלוש הסביבות (גרינפוט אליס וסקרצ') התחיל בכנס SIGCSE 2010 והמשיך בהכנת העיתון המיוחד המוקדש לשלושת הסביבות (ראה מקורות בסוף המאמר).
השאלות המובאות בהמשך הוצגו על ידי Ian Utting והשתתפו בדיון Stephen Cooper, Michael Kölling, Mitchel Resnick, John Maloney.

נושאים הקשורים בגיל
כפי שהוצגו הסביבות, ברור שכל אחת מהן מיועדת לגיל אחר: גילאים 8-16 לסקרצ', גילאי 14+ לגרינפוט ומבוא למדעי המחשב לאליס. האם בגלל שזה הגיל המתאים ללימוד תכנות?
קולינג: אין גיל "מתאים" ללימוד רעיונות תכנותיים. זה כמו לימוד שפה טבעית – ככל שמתחילים מוקדם יותר זה טוב יותר. אבל יש גבולות של התפתחות קוגניטיבית ללימוד רעיונות או טכניקות מסוימים. בגרינפוט משתמשים בתחביר ג'אווה. זה מציב גבול ולא מאפשר לילדים צעירים להצטרף. הניסיון שלהם מראה שבני 13 יכולים להסתדר ובני 10 לא יכולים. לכן, מגבלת הגיל היא side effect של ההחלטה התכנונית (ג'אווה) לעבוד עם שפה אמיתית.
מלוני ורזניק: בפיתוח סקרצ' הם רצו "להנמיך את הרצפה" כך שילדים יוכלו להתחיל תכנות מוקדם ככל האפשר. לפי החזון שלהם, לימוד תכנות הוא כמו לימוד כתיבה. בשני המקרים, ילדים צריכים להתחיל מייד כאשר הם מעוניינים בכך. זה הגיוני שילדים יתחילו עם צורות פשוטות של התבטאות וילמדו באופן הדרגתי דרכים מעודנות ומתוחכמות לבטא את עצמם עם הזמן.
קופר: אין גיל מושלם לאליס. יש כאלה שמשתמשים באליס רק כמנגנון לבניית משחקים תלת ממדיים. אבל הדגש שלנו הוא על שימוש באנימציה כסביבה מלהיבה ללימוד של פתרון בעיות ותכנות, כאשר התכנות הוא יישום לאלגוריתם (או תכנון) של פתרון הבעייה. לכן, הם חושבים שאליס הכי מתאים לאותם תלמידים שעוסקים בפתרון בעיות ויש להם מספיק רקע (מלימוד של פתרון בעיות במתימטיקה למשל) כדי להתמודד עם פתרון בעיות. הוא עבד עם ילדות מה"צופים" בנות 10 אחרי כיתה ד (סדנאות של

שעתיים). אבל הדגש שם לא היה על פתרון בעיות מורכבות אלא על המנגנון של שימוש באליס כדי לספר סיפור. יש כאלה שמשתמשים באליס גם במכללות.

נושאים הקשורים במגדר

יש דאגה גוברת בגלל חוסר האיזון המגדרי במדעי המחשב. בעיקר ייצוג מופחת של נשים. האם אתם מאמינים שהמערכות האלה יכולות לעזור? בפרט, האם המערכת שלכם תורמת משהו לעניין הזה?

קולינג: אין לנו ראיות/הוכחה אבל אנחנו מקווים שכן וזו אחת המטרות. אבל לא כיוונו לנושאים "נשיים" אלא אנו מקווים לסייע דרך הדגשת האינדיבידואליות בעבודה בסביבה. הניסיון מראה שבממוצע, נשים מתעניינות בדברים אחרים מגברים. הן יותר מכוונות לענייני חברה, מתעניינות בהקשרים שעושים שינוי בחברה. בעוד בנים שמחים אם סתם נותנים להם לשחק. אבל זה לא תקף לכל בן או בת – זו הכללה. הם מנסים להביא את התלמידים במהירות למצב שתלמידים יעשו מה שמעניין אותם. וכך הם מייחסים תכנות לעבודה שבה יש להם מוטיבציה ועניין. גרינפוט לבדה לא מספיקה לשם כך וצריך מורה טוב שמציג אתגרים טובים.

מלוני ורזניק: במחקר על סקרצ' שערך המרכז הארצי לנשים וטכנולוגית המידע, הם קוראים לסקרצ' בשם promising practice להגדלת המגוון המגדרי בטכנולוגיות המידע כי סקרצ' מאפשר למידה פעילה, הוא ויזואלי, מאפשר לבטא יצירתיות, המשוב מיידי וברור, וכן יש שחרור מבעיות תחביר לעיסוק בתהליכים ורעיונות. יש עדויות שסקרצ' עוזר, למשל בהרוורד, שימוש בסקרצ' בקורס מבוא לתכנות סייע להפחתה חדה בנשירה מהקורס ובקבלת ציונים נכשלים וגם בעלייה משמעותית במספר הנשים

קופר: פרסמו מספר מחקרים על שימוש באליס. מחקר אחד הראה שיפור משמעותי אצל סטודנטים "בסכנה" (חשיפה לאליס, לפני או תוך כדי CS1, הביאה לביצועים טובים יותר ב- CS1 ופחות נשירה – הרבה המשיכו לקורס CS2). שני המחקרים האחרים עקבו אחרי לימוד אליס במכללה והיו תוצאות טובות אצל נשים. אחרים דיווחו גם כן על הצלחה. למשל, באוניברסיטת הוואי בנו קורס שמשלב הסטוריה ותרבות מקומית עם תכנות באליס.

דברים משותפים בין הסביבות

מקריאת המאמרים שלכם מעניין שיש דברים דומים במטרות הפיתוח והתהליכים. איזה דברים משותפים כאלה הם לדעתכם החשובים ביותר?

קולינג: מעורבות ומתן כח ללומד.

מלוני: מסכים. כל המערכות שואפות למעורבות הלומד על ידי כך שהם כותבים תוכניות על מה שמעניין אותם בניגוד להוראת תכנות שגרתית (דוגמאות יכולות להיות יצירת מספרים ראשוניים או מיון רשימת מספרים). בנוסף, כל המערכות מתמקדות בפתרון בעיות עם התנסות פעילה hands-on, כולן מיועדות להכרות עם תכנות לפני (או תוך כדי) CS1, כולן שואפות לעודד תלמידים "בסכנה" וכולן מנסות להגיע לקהל רחב.

קופר: מסכים.

מלוני: כל המערכות ממחישות עצמים, מספקות פקודות ברמה גבוהה כמו move ומסתירות פרטים ברמה נמוכה.

קולינג: זה מתקשר לאחד הדברים החשובים ביותר שאנחנו מנסים ללמד – אל תסתפק במה שנותנים לך, אתה לא רק משתמש בשפה אלא גם מתכנן designer. הכח הזה של הפשטה הוא אחד הדברים המלהיבים במדעי המחשב.

קופר: מעניין לציין שברוקס מתייחס לזה כהתפתחות של הנדסת תוכנה (Brooks במאמר No Silver Bullet). הוא מתייחס לכך ששפה עלית מספקת תוספת שכבות של הפשטה, אבל אני רואה אנלוגיה למה שאנחנו מדברים עליו כאן.

מלוני: כל המערכות שואפות להפחית קשיים אבל בדרכים שונות. אליס וסקרצ' מורידות קשיי תחביר. גרינפוט מספקת את השיטה הראשית וספריה קטנה אך חזקה של מחלקות שמגנה על המשתמש מפני העושר והמורכבות של ג'אווה.

תהליכי פיתוח

שלושת הכלים שאנחנו מדברים עליהם הם חלק ממסורת של גרפיקת הצב של פפרט, וקארל הרובוט של פאטיס. באיזו מידה הם השפיעו על העבודה שלכם והיו חלק מתהליך התכנון?

קולינג: חלק מרעיונות התכנון היו בהתחלה ואחרים הגיעו תוך כדי עבודה. מאד מעניין ששלוש הסביבות

הגיעו בסופו של דבר לרעיונות דומים וגם למכניזמים ופתרונות דומים. זה מראה שהרוח, המטרה והפילוסופיה המנחה היו דומים מלכתחילה. אחד הרעיונות החשובים ביותר ששאבנו מהמערכות הישנות הוא הויזואליזציה.

מלוני: מסכים. בסקרצ' היו הרבה אנשים שהתווכחו על החלטות התכנון. היה גרעין של 4 אנשים, שלכולם היה נסיון במערכות קודמות ( Logo, StarLogo, Etoys, Crickets, Mindstorms). למרות כל הנסיון, עדיין היו הרבה בעיות ודיונים.

קופר: מסכים. כולנו מרויחים מהמערכות הקודמות. בצוות אליס, כולנו הסכמנו על חשיבות הויזואליזציה ועל הצורך לאפשר לתלמיד להיות הכותב / המחבר של הסיפור (ולא סתם צופה פסיבי או אפילו אקטיבי של אנימציות ע"י המורה).

חשיפה לרעיונות

חלק קריטי בפיתוח כלים לקהל של לא-מומחים הוא בבחירת הרעיונות – מה לכלול ועל מה לוותר. מה הניע אתכם לעשות את ההחלטות שבחרתם?

קולינג: כל הסביבות מנסות להסיר או להחביא מורכבויות, לאפשר למשתמש לעבוד עם מבנים בסיסיים. פרט מעניין הוא הדרגה של למידה מכוונת שיש במערכת: סקרצ' מנסה להתחמק מזה לגמרי ומאפשרת למשתמשים ללמוד לבד דרך משחקים. גרינפוט משחקת קצת יותר תפקיד של מורה (מכריח אותך לעשות דברים שאינם הכרחיים בגלל שאנחנו חושבים שזה טוב בשבילך). למשל, קומפילציה. סקרצ' מחביאה את הקומפילציה לגמרי. בגרינפוט יש כפתור קומפילציה. יכולנו להחביא ולבצע את זה אוטומטית אבל החלטנו שקומפילציה היא רעיון חשוב ורצינו שתלמידים ילמדו אותו. אליס היא איפושהו באמצע. זה משקף את שלבי הלימוד השונים של קהל היעד, אבל מצביע על פילוסופיה דומה: תן להם לשחק בהתחלה, תן להם להשיג משהו, תן להם להיות יצירתיים, ואז תגניב את ההסברים על מה שקורה. לכן, טוב ללכת לפי הסדר סקרצ', אח"כ אליס ואז גרינפוט.

מלוני: מסכים.

קופר: באליס 2 אין קומפילציה בכלל.

קולינג: דוגמה נוספת היא הטיפול בשגיאות. סקרצ' מנסה לעשות תמיד דבר נכון, גם אם התלמידים לא מבינים למה זה נכון. גרינפוט מנסה לגרום לתלמידים להבין את הפרטים של איך דברים צריכים להיות כדי שיעבדו.

מלוני: מסלול טוב להתחיל עם סקרצ' ואז להתקדם

Scratch → Greenfoot → Eclipse

קופר: מסכים שהמעברים הבאים מוצלחים

Scratch → Greenfoot, Alice → Greenfoot, and

Scratch → Alice → Greenfoot

על הערך של פשטות

לא משנה מה מחליטים, תמיד יש לחץ מצד המשתמשים לקבל פונקציונליות יותר מתקדמת. איך התמודדתם עם זה?

קולינג: בכל המערכות זה קיים. מנסים למצוא איזון. בכל 3 הסביבות מנסים לשמור על פשטות.

מלוני: כל הצוות של סקרצ' רצה שהמערכת תהיה פשוטה והיו מוכנים להקריב דברים אחרים.

קופר: משתמשי אליס מבקשים כל הזמן שנוסיף מאפיינים ועד כה לא הסכימו כי שמרו על פשטות.

הבדלים בין הסביבות

עד כאן הסכמתם על הכל. על מה אתם לא מסכימים? איפה ההבדלים?

מלוני: ביחס לתכנות מונחה עצמים וג'אווה – בסקרצ' ספרייטים הם עצמים עם מצב והתנהגות אבל אפשר רק להעתיק אותם. אין מחלקות או הורשה. בגרינפוט יש את מודל העצם של ג'אווה. אליס2 איפושהו באמצע עם רמה אחת בלבד של תת מחלקות. זה משקף מטרות שונות: סקרצ' לא מנסה ללמד תכנות מונחה עצמים, בעוד אליס וגרינפוט כן.

מלוני: ניתן להשתמש בגרינפוט לבעיות מורכבות (קריפטוגרפיה). סקרצ' יותר מידי איטי לבעיות כאלה. ולכן סקרצ' לא מתאים לקורס cs1.

קופר: אליס יותר קרובה לסקרצ' בהקשר הזה.

מלוני: אליס היא סביבה תלת ממדי. זה מצוין לסוגים מסוימים של פרויקטים (סיפורים, משחקים, עולמות וירטואלים). מצד שני, יותר קל לצייר, לייבא תמונות וליצור דמויות בדו ממדי.

קולינג: בגרינפוט קשה לייצר סיפורים. יותר מתאים לתגובות (תגובה למשתמש או עצמים שמגיבים אחד לשני).

## רשימת מקורות

המאמר מתבסס בעיקר על המאמרים הבאים:

Fincher, S., Utting, I. (2010). Machines for Thinking. ACM Transactions on Computing Education, Vol. 10, No. 4, Article 13, Pub. date: November 2010.

Kölling, M. (2010). The Greenfoot Programming Environment. ACM Transactions on Computing Education, Vol. 10, No. 4, Article 14, Pub. date: November 2010

Cooper, S. (2010). The Design of Alice. ACM Transactions on Computing Education, Vol. 10, No. 4, Article 15, Pub. date: November 2010.

Maloney, J., Resnick, M., Rusk, N., Silverman, B., Eastmond, E. (2010). The Scratch Programming Language and Environment. ACM Transactions on Computing Education, Vol. 10, No. 4, Article 16, Pub. date: November 2010

Utting, I., Cooper, S., Kölling, M. , Maloney, J., Resnick, M. (2010). Alice, Greenfoot, and Scratch – A Discussion. ACM Transactions on Computing Education, Vol. 10, No. 4, Article 17, Pub. date: November 2010.

מקורות נוספים:

Guzdial, M. (2004). Programming environments for novices. In S. Fincher and M. Petre, Eds., *Computer Science Education Research*, Taylor and Francis.

Kelleher, C., Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surv. 37*, 83–137.

Rongas, T., Kaarna, A., AND Kalviainen, H. (2004). Classification of computerized learning tools for introductory programming courses: Learning approach. In *Proceedings of the IEEE International Conference on Advanced Learning Technologies (ALT'04).*

### פורסם באתר האינטרנט של המרכז

**באתר האינטרנט של המרכז הארצי פורסמו קבצים חדשים. אתם מוזמנים להיעזר בהם.**
**בין היתר, פורסמו חומרי עזר למורה לחטיבת ביניים וכן חומרי עזר למורה בתיכון.**
**הכתובת של כל החומרים שפורסמו השנה**

**http://cse.proj.ac.il/Y16/New16/**

# הבנת הנקרא במדעי המחשב

**אודי מלכה**

**מכללת אורט ביאליק**

מקצועות רבי מלל כמו תנ"ך, היסטוריה וספרות מצריכים מן התלמידים קריאה לא מועטה של טקסט והבנה שלו. למתמטיקה ומדעי המחשב תדמית של מקצועות בהם לטקסט חשיבות מעטה ושעיקר המיומנויות הנדרשות מהתלמידים מתמקדות בטכניקה ובלוגיקה. מצבים שבהם תלמידים אינם קוראים את השאלה במלואה על כל סעיפיה או מצבים שבהם הטעויות שלהם נובעות מחוסר הבנה של הטקסט – הם מצבים שכיחים בכיתות שאני מלמד בהן מקצועות במגמת מדעי המחשב.

המאמר הנוכחי בוחן את הסוגיה תוך בדיקה של דוגמאות לפתרון הבעיה מתחום המתמטיקה. במאמר מוצעת אסטרטגיה לפתרון שאלה מרובת מלל במדעי המחשב תוך מתן דוגמא ליישום האסטרטגיה. האסטרטגיה נבנתה תוך שימוש באלמנטים מתוך אסטרטגיות שפותחו במקור עבור תלמידים לקויי למידה.

## שפת המתימטיקה ושפת תכנות

מתמטיקה היא שפה של סמלים, של מושגים, של הגדרות ושל משפטים. בדומה למתמטיקה גם שפות תכנות הנלמדות כחלק ממקצועות מדעי המחשב הן בהגדרתן שפות אשר כוללות דקדוק ותחביר, מושגים והגדרות עם קשרים פנימיים ביניהם. בדומה לשפה המדוברת, גם בשפות התכנות קיימת התפתחות במהלך הזמן ומפעם לפעם יוצאות גרסאות חדשות עם שיפורים של אלמנטים קיימים ולעיתים עם אלמנטים חדשים. מתמטיקה בנויה מספרות המרכיבות מספרים, וסמלים שונים המייצגים למשל פעולות חשבון. בשפת התכנות קיימות פקודות שונות אשר שילובים שונים שלהן יוצרים תגובות שונות ומשמעויות שונות.

התחביר עוסק באופן כללי בכללי התצורה שלפיהם מורכבים משפטים ומילים. יש לגשר בין השפה הטבעית המחייבת התייחסות אוריינית לטקסט השלם לבין השפה המתמטית/תכנותית המחייבת הבחנה במרכיבי שונים בשפה. **הפערים במידע בתוך השפה המתמטית או שפת התכנות הם אלו שקובעים עד כמה השפה הטבעית תשלים את החסר**. כלומר, כאשר נתונה משוואה אזי אין מקום לשפה הטבעית אך כאשר מדובר בבעיה כמותית שממנה יש לדלות את הנתונים, היחסים ביניהם ולבנות את המשוואה אזי השפה הטבעית היא זו שעומדת לרשותנו. בדומה, כאשר קיים קוד ( source code) אזי הצורך של המשתמש או הדרישה של המערכת כבר תורגמו מהעולם האמיתי לתוך מודל שיושם בשפת תכנות אך כאשר הדרישה היא לקרוא משימה וליישם אותה בשפת תכנות אזי לשפה הטבעית יש חשיבות מכרעת והבנה או חוסר הבנה של הטקסט ישפיעו על התכנון ועל התוצאה הסופית. בפתרון בעיות מילוליות, מתמטיות או תכנותיות שיש בהן המללה, התלמיד ניתקל בשתי שפות שונות זו מזו המופיעות יחד בערבוביה - השפה הטבעית והשפה המתמטית/תכנותית. גם כאשר התלמיד פוגש מקרים חדשים או אפילו לומד נושא חדש שלא הכיר בעבר, הוא מתמודד איתם על פי סכמות קודמות שיש לו במודל המנטאלי העוסקות באותו עניין.

מודלים מנטאליים כוללים קבוצת חוקים ואילוצים הנבנים מתוך תפיסות, אמונות וידע אשר נצברו מהתנסויות אישיות. מודלים אלו משפיעים ולעיתים שולטים בתהליך החשיבה ובכך הם מספקים ידע שעולה מתוך התת-מודע ומהווים אינטואיציה ראשונית. מודל מנטאלי הוא ייצוג של מושג מופשט המתפתח אצל כל אחד מאיתנו ומשמש לספק הסבר ויכולת לחזות תוצאות המשויכות לנושא מסוים.

חלק מן הקושי בהבנה של תלמידים בנושאים מסוימים נובע מהמודלים המנטאליים שנבנו אצלם בעת שלמדו נושאים אלו או בעקבות מודלים כאלו שנבנו קודם לכן בנושא דומה (Ben-Ari, 1998). למשל, לא תמיד קל לתלמיד שרגיל להשתמש בסימן "=" לצורך מסוים ובמשמעות מסוימת, מלימודי המתמטיקה או מלימודי תוכנת הגיליון האלקטרוני אקסל, לעובדה שלאותו הסימן יש משמעות אחרת בשפות תכנות והיא פעולת הצבה. ההשלכה של שימוש בסכמות מסוימות ולא אחרות מביאה התלמיד למצב של ציפייה לתוצאה מסוימת על פי הסכמה המעורבת. הסכמה מתרחבת כאשר קיימת התאמה בין הציפיות והתוצאה או לשינוי הסכמה במידה ואין התאמה או לרכישת סכמה חדשה. במצב של רכישת סכמה הלומד פוגש מקרים חדשים ומתמודד אתם לפי הסכמות הקודמות שלו, הקשורות לאותו העניין. הלומד מצפה להתרחשות או לתוצאה מסוימת. אם ההתרחשות תואמת את ציפיותיו, חלה הרחבה של הסכמה הקיימת אצלו, ואם לאו, חלה הפרה היכולה לגרום לשינוי הסכמה ולרכישת סכמה חדשה. בשלב זה נזקק התלמיד לעיבוד האינפורמציה המילולית לצורך הפיכתה לתרגיל מתמטי/משוואה או לבעיה תכנותית.

**אסטרטגיות המשלבות מילות מפתח**

אסטרטגיות העושות שימוש במילות מפתח הן יעילות כאשר התלמיד ממפה את הבעיה למודל המתאים. יחד עם זאת שימוש במילות מפתח בלבד היא דווקא אסטרטגיה שאינה יעילה (Bernadette & Douglas, 1996). בשיטה זו המוצגת במאמר של Bernadette & Douglas, המוצגת כשיטה המתאימה להוראת ללקויי למידה, התלמידים מתורגלים לזהות מילות מפתח (כגון "more", "all" together", "gave away", "left over") ולשייך אותן לפעולות מתמטיות. לטענת כותבי המאמר לעיתים האסטרטגיה עובדת אך לעיתים היא בעייתית. לדוגמא בבעיה הבאה:

"Juan gave away 9 cookies in the morning. He gave away 4 cookies in the afternoon. How many cookies did he give away that day?"

הביטוי "give away" אינו יכול להיות מתורגם באופן אוטומאטי לפעולת חיסור. בעיה זו קיימת גם בפתרון בעיות אלגוריתמיות (כהן, 2008). בחירה באסטרטגיה העושה שימוש במילות מפתח מחייבת מיפוי נכון של הבעיה למודל מתאים.

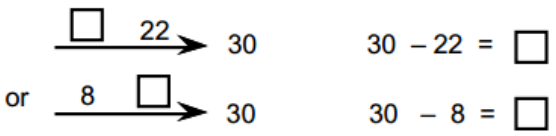
המאמר של Jintendra העוסק בייצוג גראפי של בעיות מתמטיות ומציג 4 מודלים המשתמשים במילות מפתח:

1. דיאגראמות
2. משוואות יחס
3. טבלאות
4. משוואות פעולה הפוכה

<u>1. דיאגראמות</u>

כל אחת מפעולות החשבון הבסיסיות (חיבור, חיסור, כפל וחילוק) ניתנת לייצוג גרפי.

לייצוג חיבור וחיסור יש שתי משפחות: המספרים הקטנים והמספר הגדול (המייצג סכום):




בבעיות מסוג זה קיימת חשיבות גם לזיהוי השוואות (רון מכר 30 פריטים יותר מדנה) וגם הכללות (שכלבים וחתולים למשל שייכים לקבוצה גדולה יותר - של חיות).

<u>2. טבלאות</u>

שימוש בטבלאות לארגון נתונים.

למשל, נתונה הבעיה הבאה:

**לתלמידה אן יש 69 ספרים בכריכה רכה ו-51 ספרים בכריכה קשה בשתי קופסאות. קופסא אחת כחולה והשנייה אדומה. אן ארזה 62 ספרים בקופסא הכחולה, 15 מתוכם בעלי כריכה קשה.**

1. כמה ספרים יש בקופסא האדומה ?
2. איזו קופסא מכילה יותר ספרים בעלי כריכה רכה?
3. כמה ספרים בעלי כריכה קשה יש בקופסא האדומה ?
4. מהו מספר הספרים הכולל בשתי הקופסאות ?

בשאלה זו התלמידים נדרשים לעבוד על פי אסטרטגיה שבה מוטלת עליהם לבנות טבלה עם העמודות השונות המייצגות את הנדרש בשאלות. הזנה של נתונים לתוך הטבלה תבליט להם את פעולות החשבון שהם נדרשים להם והפתרונות בסופו של דבר ימצאו בטבלה עצמה.

| | כריכה רכה | כריכה קשה | סה"כ |
|---|---|---|---|
| קופסא כחולה | | 15 | 62 |
| קופסא אדומה | | | |
| סה"כ | 69 | 51 | |

הטבלה עוזרת לתלמיד לזהות את הנתונים שהוא נדרש לגלות ואשר יסייעו במציאת התשובה. חלקם יכולים להיות נתונים שהוא נדרש למצוא בסעיפים מאוחרים יותר וחלקם לא.

3. משוואות פעולה הפוכה

קיים מגוון של בעיות שונות שבהן יש צורך לבצע פעולות מהסוף להתחלה על מנת להגיע לפתרון. דוגמא של בעיה כזו:

**קח מספר, הכפל אותו ב-3. חסר 40. חלק את התוצאה ב-2 והתוצאה היא 7.**

**מהו המספר המקורי ?.**

לפתרון הבעיה הראשונה התלמידים נדרשים לבנות את המודל הבא:

?

$\square \times 3 = \square$

$\square - 40 = \boxed{14}$

$\boxed{14} \div 2 = 7$

באמצעות חישובים מהסוף להתחלה יוכלו התלמידים להגיע לתוצאה: (בטור הבא)

?

$\boxed{18} \times 3 = \boxed{54}$

$\boxed{54} - 40 = \boxed{14}$

$\boxed{14} \div 2 = 7$

בעיה נוספת:

יש לאסוף סכום מסוים לתרומות. הסכום יוכפל על ידי התורם הראשי. אחרי תשלום חשבונות של 120$ ותשלום נוסף של 58$ נשאר סכום של 3000$ להוצאות נוספות.

מהו הסכום שיש לאסוף מתרומות ? לפתרון הבעיה יש לבנות את המודל הבא:

?

$\square \times 2 = \square$

$\square - 120 = \square$

$\square - 58 = 3000$

באמצעות חישובים מהסוף להתחלה יוכלו התלמידים להגיע לתוצאה:

?

$\boxed{1589} \times 2 = \boxed{3178}$

$\boxed{3178} - 120 = \boxed{3058}$

$\boxed{3058} - 58 = 3000$

4. משוואות יחס

שימוש בפעולות חשבון של כפל וחילוק יכול להתבצע אף הוא באמצעות מייצג גרפי:

**"בקרטון יש 8 בקבוקים. כמה בקבוקים יש ב-25 קרטונים ?"**

| | |
|---|---|
| ייצוג המידע הראשוני: | $\frac{8 \text{ bottles}}{1 \text{ cartons}}$ |
| ייצוג הבעיה: | $\frac{8 \text{ bottles}}{1 \text{ cartons}} = \frac{\square}{24}$ |
| זיהוי השוויון ביחס ביניהם: | $\frac{8 \text{ bottles}}{1 \text{ cartons}} \left(\frac{\square}{24}\right) = \frac{\square}{24}$ |
| שימוש בכפל לפתרון הבעיה: | $\frac{8 \text{ bottles}}{1 \text{ cartons}} \left(\frac{24}{24}\right) = \frac{\boxed{192}}{24}$ $\boxed{192 \text{ bottles}}$ |

**התמודדות עם טקסט במדעי המחשב**

בהשראת האסטרטגיות המתמטיות החלטתי לאמץ מספר מאפיינים מתוך האסטרטגיות לפתרון בעיה מתוך מבחן בגרות במדעי המחשב. במתמטיקה קיימים נתונים ופעולות עליהם. האסטרטגיות השונות מיועדות לעזור לתלמיד לזהות את הנתונים ולפרק את הטקסט בבעיה למשוואות מתמטיות. כך למעשה מתבצע התרגום של השפה הטבעית לשפה המתמטית ולאחריו פנוי התלמיד לעסוק בפתרון המשוואה. ניתן להתייחס לפתרון בעיות מילוליות במדעי המחשב באופן דומה: השפה הטבעית מתורגמת למשימות הנדרשות לביצוע באמצעות פקודות מחשב. התוכנית נדרשת להתחיל עם קלט אפשרי (או שלא) אך מסתיימת (כמעט) תמיד עם פלט. הפעולות המתבצעות בתווך מכוונות למימוש מטרת התוכנית שיכולה להיות מחולקת למטרות משנה.

אחד הקשיים המהותיים מתמקד בשאלות העוסקות בתכנות וביחס לכל הפעולות שתוארו עד כה הוא השילוב בין תתי המשימות הדרושות ליישום מטרת התוכנית. כמעט תמיד תתי המשימות נדרשות לעבוד בשילוב אחת עם השנייה. במקרה הפשוט הן מתבצעות אחת לאחר השנייה ואז התלות ביניהן היא בסדר הפעלתן אך לעיתים פלט של תת משימה אחת מהווה קלט או תנאי מקדים לפעולתה של אחרת. לעיתים מספר תתי משימות יכולות לעבוד במקביל עם/בלי קשר ביניהן ולעיתים קיים קושי להפריד אותן למקטעים נפרדים. במקרים מסוימים ייתכן מצב שבו תת משימה שכבר נבדקה ונמצאה תקינה תציג תוצאה לא נכונה בגלל משאב משותף שהיא חולקת עם תת משימה אחרת שלא הייתה קיימת בעת הבדיקות של תת המשימה הראשונה. החלוקה לתתי משימות אינה "המצאה חדשנית" והיא מובנית בשפה באמצעות כתיבה של **פעולות**. פתרון בעיות באמצעות פעולות מתאים כאשר לתרגיל/שאלה רמת מורכבות מסוימת והיא אינה מתאימה כאשר התרגיל פשוט יחסית ואינו מצריך מספר רב של משתנים/תנאים/לולאות ושאר מנגנונים הקיימים בשפת תכנות (אלא אם כוללים התייחסות לפעולות המיועדות להפעלה של אובייקטים) או כאשר אין צורך בשימוש חוזר של הקוד המיועד להתבצע על ידי הפעולה.

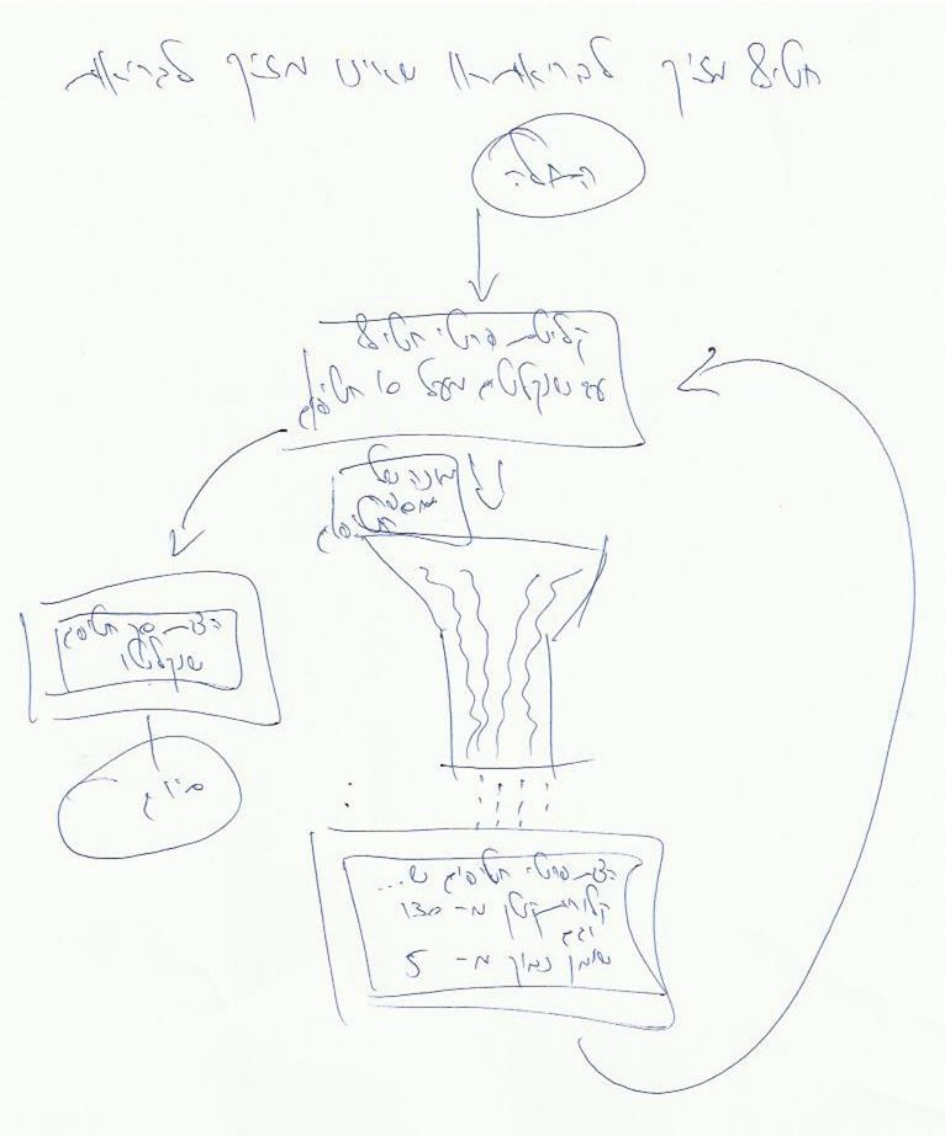
שיטה נוספת שנעשה בה שימוש בהוראת שפות תכנות היא שימוש **בתרשימי זרימה**. למעשה מדובר בייצוג גרפי של הבעיה כאשר באמצעות תרשימים אלו יכול לנתח הלומד את זרימת המידע והפעולות הנדרשות כדי להביא לפתרון התרגיל. בחלק ממוסדות הלימוד נעשה שימוש בתרשימי זרימה עוד בטרם נלמדה שפת תכנות כלשהי ובחלק מן מוסדות הלימוד לא נעשה שימוש כלל בשפה ויזואלית זו.

ייצוג ויזואלי כמו תרשים זרימה מציג את הזרימה של המידע, הפקודות וההסתעפויות שקיימות בתוכנית עם אפשרות לעקוב אחר אלו שבאמת מתבצעות בעקבות קלט מסוים. הקושי של התלמיד הוא לעיתים לראות את הממשיות של התוכנית שהוא כותב או קורא וההקשר שלה לעולם האמתי וזו אפשרית באמצעות **מטאפורות חזותיות**. זה הזמן לציין שניתן לשלב בין מספר כלים לדוגמא, תרשים זרימה המציג פעולות או תרשים זרימה המשלב מטאפורות חזותיות.

בתרשים הזרימה בעמוד הבא ניתן להבחין <u>במסנן</u> שדרכו עוברים רק פרטי חטיפים שאינם מזיקים לבריאות.

הצגה של סמלים או איורים בתוך תרשים זרימה שאינם סימנים מוסכמים מפרה את כללי תרשים הזרימה כסכימה ויזואלית תקנית והופכים אותו לסקיצה שהיא לרוב קריאה רק עבור היוצר.

שיטה אחרת, אשר תוצג בפירוט באמצעות דוגמא בהמשך, משלבת מספר אלמנטים המצויים באסטרטגיות שונות: סימון/כתיבה של מטרת השאלה, סימון מילות מפתח (וחלוקתן על פי קטגוריות) ומספור פעולות לביצוע.



לשיטה זו מומלץ להוסיף שתי שאלות אשר ישאלו בסיום: האם כתבתי את התוכנית הנכונה והאם כתבתי אותה בצורה הנכונה (שאלות אלו מתמצות לתוך מושג שנקרא בתעשייה בכלל וגם בתחום הנדסת התוכנה: verification and validation-V&V.

סעיף זה הוא בהשראת מאמרם של בן-דוד קוליקנט ומוסאי שעוסק בהבדל שבין תוכנית **נכונה** לבין אחת **שעובדת** שיכולה לעיתים גם לא לפעול כראוי אך נתפסת ע"י התלמיד כ-"טובה מספיק" (בן-דוד קוליקנט ומוסאי, 2008). במאמר אחר מצוין היתרון של התיווך הוויזואלי אך מצורפת הזהרה מפני הכבדה על תהליכים קוגניטיביים המתרחשים אצל הלומד, כיוון שהם מכילים פרטי מידע נוספים, כמו: צבע, גודל, קול (במקרה של סרטונים) ועוד (ברוזה ובן דוד קוליקנט, 2010).

לסיכום, צוינו 4 דרכים עיקריות להתמודד עם טקסט בהקשר לשאלות מתחום מדעי המחשב:

1. חלוקת השאלה למשימות משנה (פעולות).
2. תרשימי זרימה.
3. ייצוג גרפי – בשילוב מטאפורה חזותית.
4. ייצוג גרפי המתמקד בטקסט.

כעת נציג שאלה מבחינת בגרות ופתרונה על פי הדרך ייצוג גרפי המתמקד בטקסט.

**שאלה 8 ממבחן בגרות תשע"ב 2012**

**8.** הוחלט לבדוק את הסימונים התזונתיים הרשומים על אריזות של חטיפים, כדי לקבוע אם חטיף אינו מזיק לבריאות או מזיק לבריאות.

הסימונים התזונתיים שנבדקו הם מספר הקלוריות בחטיף, ומשקל השומן שבו בגרמים.

חטיף אינו מזיק לבריאות אם מספר הקלוריות שבו קטן מ- 130 ומשקל השומן שבו נמוך מ- 5 גרמים.

כתוב ב- Java או ב- C# , תכנית שתקלוט את הפרטים האלה:

שם החטיף, מספר הקלוריות שבו, משקל השומן שבו.

התכנית תדפיס את שמות החטיפים שאינם מזיקים לבריאות.

הקליטה תסתיים כאשר מספר החטיפים שאינם מזיקים לבריאות שנקלטו יהיה גדול מ- 10 .

כמו כן התכנית תמנה ותדפיס את מספר החטיפים שנקלטו בסך הכול.

הערה: אין צורך לבדוק את תקינות הקלט.

**על השלבים לפתרון תרגיל**

**א.** סימון בטקסט את המטרת התוכנית (או כתיבה שלה מעל הטקסט), סימון קלט ופלט (כתיבת הערות הקשורות בקלט פלט).

**ב.** סימון מילות מפתח המתייחסות לפעולות חשבון, אופרטורים לוגיים וערכים (כל קטגוריה בצבע אחר). הסימון עוזר לתלמיד ליצור יחידות עצמאיות עם משמעות שאותן הוא יודע לתרגם לפקודות בשפת תכנות. סעיף זה הוא סעיף קדם לזה שאחריו כיוון שהוא מאגד נתונים, פעולות חשבון, פעולות לוגיות לידי קטע קוד המהווה יחידה עצמאית ורק נוצר לבדוק מהו סדר הביצוע שבו היא נדרשת לפעול.

* כאשר מדובר בתרגול ניתן לכתוב את השאלה מחדש ולפצל את מקטעי הטקסט לשורות נפרדות.
* כדאי להדגיש כי מיקוד יתר במילות מפתח יכול לגרום לסטייה ממטרת התוכנית וניסיון אוטומטי לפעול על פי תבנית מוגדרת מראש שאינה מתאימה לדרישות השאלה.

**ג.** מספור הפעולות לביצוע על פי הסדר הנדרש בתוכנית. סעיף זה הוא קריטי ומהווה יתרון מהותי גם לתלמידים שאינם לקויים כיוון שהוא מחייב אותם להתמודד עם הלוגיקה של התרגיל בטרם יתחילו לכתוב את התוכנית.

סעיפים המסוגלים להתבצע במקביל ללא תלות בסדר הכרונולוגי של ביצוע הפעולות יסומנו באמצעות תתי סעיפים (4.א, 4.ב וכו').

**ד.** יישום (כתיבת תוכנית) על פי סדר השלבים ושימוש במילות המפתח להכוונה לכתיבת הפעולות לביצוע.

**ה.** נשאלות שתי שאלות :

1. האם פתרתי את השאלה הנכונה ?
כלומר האם התוכנית אם מיישמת את הדרישות שהוצגו ומהווה פתרון מלא לשאלה.
2. האם פתרתי בדרך הנכונה ?
כלומר האם הכלים, הפקודות והרעיון הכללי הם מתאימים (והם המתאימים ביותר) לפתרון בעיה זו ובעיות מסוג זה.

**יישום השלבים השונים בפתרון התרגיל**

**שלב א'**

סימון בטקסט את המטרת התוכנית (או כתיבה שלה מעל הטקסט), סימון קלט ופלט (כתיבת הערות הקשורות בקלט פלט).

יש לסמן את הקלט והפלט על גבי התרגיל. במידה ויש הערה לגבי קליטה יש לציין אותה.
לאחר שלב זה הטקסט בתרגיל שנבחר יראה כך :

8. הוחלט לבדוק את הסימונים התזונתיים הרשומים על אריזות של חטיפים, כדי לקבוע אם חטיף אינו מזיק לבריאות או מזיק לבריאות.
הסימונים התזונתיים שנבדקו הם מספר הקלוריות בחטיף, ומשקל השומן שבו בגרמים.
חטיף אינו מזיק לבריאות אם מספר הקלוריות שבו קטן מ-130 ומשקל השומן שבו נמוך מ-5 גרמים.
כתוב תוכנית ב- Java או ב- C#, תכנית שתקלוט את הפרטים האלה:
שם החטיף, מספר הקלוריות שבו, משקל השומן שבו. *
התוכנית תדפיס את שמות החטיפים שאינם מזיקים לבריאות. *
הקליטה תסתיים כאשר מספר החטיפים שאינם מזיקים לבריאות שנקלטו יהיה גדול מ-10.
כמו כן התכנית תמנה ותדפיס את מספר החטיפים שנקלטו בסך הכול. *
הערה: אין לבדוק את תקינות הקלט.

עד ש"חטיפים אינם מזיקים" גדול מ-10

8. הוחלט לבדוק את הסימונים התזונתיים הרשומים על אריזות של חטיפים, כדי לקבוע אם חטיף אינו מזיק לבריאות או מזיק לבריאות.
הסימונים התזונתיים שנבדקו הם מספר הקלוריות בחטיף, ומשקל השומן שבו בגרמים.
חטיף אינו מזיק לבריאות אם
מספר הקלוריות שבו קטן מ-130
ומשקל השומן שבו נמוך מ-5 גרמים.

שני התנאים צריכים להתקיים

כתוב תוכנית ב- Java או ב- C#, תכנית שתקלוט את הפרטים האלה:
שם החטיף, מספר הקלוריות שבו, משקל השומן שבו. *
התוכנית תדפיס את שמות החטיפים שאינם מזיקים לבריאות. *
הקליטה תסתיים כאשר מספר החטיפים שאינם מזיקים לבריאות שנקלטו יהיה גדול מ-10.
כמו כן התכנית תמנה ותדפיס את מספר החטיפים שנקלטו בסך הכול. *
הערה: אין לבדוק את תקינות הקלט.

עד ש"חטיפים אינם מזיקים" גדול מ-10

**שלב ב'**

סימון מילות מפתח וכתיבת הערות המתייחסות לפעולות חשבון, אופרטורים לוגיים וערכים (כל קטגוריה בצבע אחר).

* בחירת הצבע יכולה להיעשות על ידי המורה מציג את השיטה או בחירה חופשית של התלמיד המשתמש בה. חשוב מאד לקבוע בתחילה את הצבעים ואת ייצוגם ולהקפיד לעבוד עם המקרא שנבחר בכל תרגיל שהתלמיד בוחר להשתמש בשיטה זו ככלי עזר בדרך אל הפתרון.

**שלב ג'**

מספור הפעולות לביצוע על פי הסדר הנדרש בתוכנית.
סעיף זה הוא קריטי ומהווה יתרון מהותי גם לתלמידים שאינם לקויים כיוון שהוא מחייב אותם להתמודד עם הלוגיקה של התרגיל בטרם יתחילו לכתוב את התוכנית.
סעיפים המסוגלים להתבצע במקביל ללא תלות בסדר הכרונולוגי של ביצוע הפעולות יסומנו באמצעות תתי סעיפים (4.א, 4.ב וכו').

8. הוחלט לבדוק את הסימונים התזונתיים הרשומים על אריזות של חטיפים, כדי לקבוע אם חטיף אינו מזיק לבריאות או מזיק לבריאות.

הסימונים התזונתיים שנבדקו הם מספר הקלוריות בחטיף, ומשקל השומן שבו בגרמים.

חטיף אינו מזיק לבריאות אם מספר הקלוריות שבו קטן מ-130 ומשקל השומן שבו נמוך מ-5 גרמים. (2.ב) (2.א)

שני התנאים צריכים להתקיים

כתוב תוכנית ב- Java או ב- C#, תכנית שתקלוט את הפרטים האלה:

שם החטיף, מספר הקלוריות שבו, משקל השומן שבו. (1.א)

התוכנית תדפיס את שמות החטיפים שאינם מזיקים לבריאות.

הקליטה תסתיים כאשר מספר החטיפים שאינם מזיקים לבריאות שנקלטו יהיה גדול מ-10. (1.ב)

עד ש"חטיפים אינם מזיקים" גדול מ-10

כמו כן התכנית תמנה ותדפיס את מספר החטיפים שנקלטו בסך הכול. (3)

הערה: אין לבדוק את תקינות הקלט.

כך בעת כתיבת הקוד התלמיד יכול לציין לעצמו את תתי המשימות שאותן פתר ולבחון כיצד הוא יכול לשלב בין תתי משימות שהינן בעלות תלות אחת בשנייה.

**שלב ד'**

שלב זה מיועד ליישום כלומר לכתיבת תוכנית באמצעות שפת תכנות כלשהי תוך שימוש בהכוונה ובסדר שנבנה בשלבים הקודמים. התלמיד נדרש לחשוב עבור כל שלב מהם המאפיינים הנוספים שהוא דרוש להם על מנת ליישם את הכתוב באותו שלב על מנת לעבור לשלב הבא.

בכל שלב יש לשקול את הקשר והתלות שבין השלבים.

עוד בטרם יחל התלמיד בכתיבת התוכנית הוא יוכל לסרוק את השלבים השונים (בעלי סימוני וצבעים שונים). השלבים הקודמים יעזרו לתלמיד להחליט אם הדרך שבה הוא מתכוון לפתור את התרגיל היא לגיטימית ובאמת תביא אותו בסיום לפתרון נכון.

להלן התוכנית (שפת המימוש שנבחרה היא java):

```
import java.util.Scanner;
public class Bagrut {

public static void main(String[] args) {

    int countAll=0, countNoDamage=0;
    int fat, calories;
    String snack;
    Scanner input=new Scanner(System.in);
    while(countNoDamage>10){
        System.out.println ("Enter snack
                    name, calories and fat:");
        snack=input.next();
        calories=input.nextInt();
        fat=input.nextInt();
        countAll++;
        if(calories<130 && fat<5){
            System.out.println(snack+
                            " is fine.");
            countNoDamage++;
        }
    }
    System.out.println("number of snacks is:
                    "+countAll);
  }
}
```

להלן התוכנית עם סימוני הצבעים של מילות המפתח ושאר הסימנים אשר משולבים לאחר הכתיבה של התוכנית:

```
import java.util.Scanner;

public class Bagrut {

public static void main(String[] args) {

    int countAll=0, countNoDamage=0;
    int fat, calories;
    String snack;
    Scanner input= new Scanner (System.in);

ב.1 while(countNoDamage>10){
        System.out.println ("Enter snack name,
                                calories and fat:");
        snack=input.next();
א.1     calories=input.nextInt();   *
        fat=input.nextInt();
        countAll+;

                א.2        ב.2
        if (calories<130 && fat<5){
            System.out.println (snack+
                                " is fine.");   *
            countNoDamage++;
                }
            }
        System.out.println("number of snacks is:
                                "+countAll);   *
}
}
```

**שלב ה'**

בשלב זה יש לשאול שתי שאלות:

1. האם פתרתי את השאלה הנכונה?

לרוב מענה על שאלה זו יתמקד בפלט של התוכנית ובדיקה האם הוא מהווה את הפלט המפורט בגוף השאלה.

2. האם פתרתי בדרך הנכונה?

ראוי לציין כי לרוב קיימות מספר דרכים לפתרון. יחד עם זאת, חזרה על שאלות אלו בשלב זה לאחר הניסיון (מוצלח או לא) לפתור את התרגיל יכול להעלות תובנות חדשות.

תשובות לשאלה הנדונה מבחינת הבגרות:

1. פתרתי את השאלה הנכונה.

הדרישה הייתה להציג את שמות החטיפים שאינם מזיקים ואת סך החטיפים שנקלטו ושני אלו מתקיימים בתוכנית.

2. פתרתי בדרך הנכונה:

מונים רלוונטיים אופסו בתחילת התוכנית, הלולאה מקיימת את הדרישה לקליטת פרטי חטיפים מעל 10 חטיפים, עבור כל חטיף מתבצעת בדיקה האם מזיק לבריאות או לא (ונספר כאשר אינו מזיק) והפלט הינו של שמות החטיפים מזיקים ואת סך החטיפים שנקלטו כפי שכבר צוין בסעיף הקודם.

## סיכום

במאמר זה הוצגו מספר אסטרטגיות על מנת לעזור לתלמידים לקויי למידה, תלמידים מתקשים וגם לתלמידים ללא לקות - להתגבר על נקודות קריטיות בהבנה וארגון טקסט. הדוגמאות שהוצגו ממחישות את הבעייתיות של התמודדות עם טקסט והמרה שלו מהשפה המדוברת, הטבעית, לשפה אחרת (מתמטית או שפת תכנות). בתהליך פיתוח תוכנה המעבר משפה לשפה מתרחש תכופות. הדרישות המגדירות את המפרט של המוצר מגיעות מהלקוח ועל מנת לפתח את התוכנה המתאימה בדרך המתאימה יש לוודא כי מבינים את צרכי הלקוח ודרישותיו וכי אלו כתובים ומנוסחים באופן ברור (במסמך אפיון) על מנת לאפשר יישום מדויק באמצעות מימוש בשפת תכנות מתאימה.

אסטרטגיות שונות המשמשות לקויי למידה יכולות להועיל גם לתלמידים ללא לקות ובכל מקרה כדאי לזכור כי חלק מהאסטרטגיות מיועד לפתח הרגלים נכונים וכאשר אלו מוטמעים אזי כדאי באופן מבוקר להשתחרר מן הכלים הטכניים שתרמו להטמעת ההרגלים.

רוב המורים במערכת החינוך אינם בעלי הכשרה או כלים לאבחון לקויות או קשיים של תלמידים. כיצד אם כן נוכל כמורים ללא כלים אבחוניים בידינו, להתאים את הכלים והאסטרטגיות לתלמידים מסוימים, מתקשים או לקויים, אם אין אנו יודעים בדיוק כיצד לזהות אצלם בעיה, להגדירה ולמקד אותה?. התשובה שלי היא גיוון בדרכי הוראה, בדיקה עצמית (רפלקציה), קבלת משוב יזום מן התלמידים ובעיקר מודעות לשונות התלמידים – כל אלו הם המפתח לשינוי .

## מקורות


בן-דוד קוליקנט י'., מוסאי מ'.(2008). האם רעדו להם הידיים? תפיסות תלמידים ביחס לנכונות תכניות. *היבטים בהוראת מדעי המחשב* , ינואר 2008, עמ' 38-43.

ברוזה, א'., בן-דוד קוליקנט, י' (2010). שימוש בסיפור הקשר ככלי הממנף למידה משמעותית בקרב תלמידים תת-משיגים במתמטיקה: השוואה בין מתווך ויזואלי דינמי לבין מתווך טקסטואלי, *דפים, 50*, עמ' 249-220. מכון מופת.

כהן, ד'. (2008). פתרון נמהר של בעיות אלגוריתמיות בעקבות מילות מפתח. *היבטים בהוראת מדעי המחשב*, ינואר 2008, עמ' 29-25.

מרגולין ב'., אילני ב"ש (2008). בין לשון למתמטיקה – חינוך לחשיבה אוריינית בפתרון בעיות מילוליות במתמטיקה, *דפים, 45*, עמ' 114 – 139. מכון מופת

Ben-Ari, M. (1998). Constructivism in Computer Science Education. *SIGCSE Bulletin* 30, 1, 257-261.

Bernadette, K & Douglas, C. (1996). *Teaching Problem-Solving Strategies for Word Problems to Students with Learning Disabilities*.

Jintendra, A. (2002). *Teaching Students Math Problem-Solving Through Graphic Representations*. TEACHING Exceptional Children, Vol. 34, No. 4, pp. 34-38.



## קורסים מתוכננים בתשע"ז

**אתם מוזמנים לעקוב אחר אתר האינטרנט של המרכז הארצי. בין היתר מתוכננים:**

- **ימי עיון לחט"ב**
- **קורס מורים מובילים ליסודי**
- **קורס מורים מובילים לחטיבת ביניים**
- **קורס מורים מובילים לתיכון**
- **קורס מבוא לסייבר באמצעות שפת פייתון בבאר שבע**
- **קורס הובלת תהליכים בחיפה**
- **סמינר קיץ למורים מובילים**

**הודעות נוספות יפורסמו באתר**

# הבטים בהוראת מדעי המחשב: משוב לגיליון יוני 2016

**קוראים יקרים**

לאחר שסיימתם לקרוא את העיתון, אנא מלאו ושלחו משוב זה בהקדם למינהלת מל״מ.

תודה על שיתוף הפעולה,

צוות המרכז הארצי

**משוב לגיליון יוני 2016 של "הבטים בהוראת מדעי המחשב"**

**נא למלא ולשלוח אל:**

**מינהלת מל"מ**

**בניין קנדה, קומה 1**

**קריית הטכניון, חיפה 3200003**

**פקס: 04-8295010**

1. שם בית הספר _____

2. שם המשיב_____ מספר המורים שעיינו בגיליון זה בבי״ס _____

3. סמנו במשבצת המתאימה את חוות דעתכם:

| | | | | הערות |
|---|---|---|---|---|
| חוות דעת כללית על הגליון | טובה מאד | טובה | לא טובה | |
| החשיבות של כתב העת | רבה מאד | רבה | לא חשוב | |
| מידת העניין | מעניין מאד | מעניין | לא מעניין | |
| תרומה לעבודתי | תורם מאד | תורם | לא תורם | |

4. הערות נוספות:

_____

_____

_____

_____

_____