הבנת הארכיטקטורה של אפליקציה

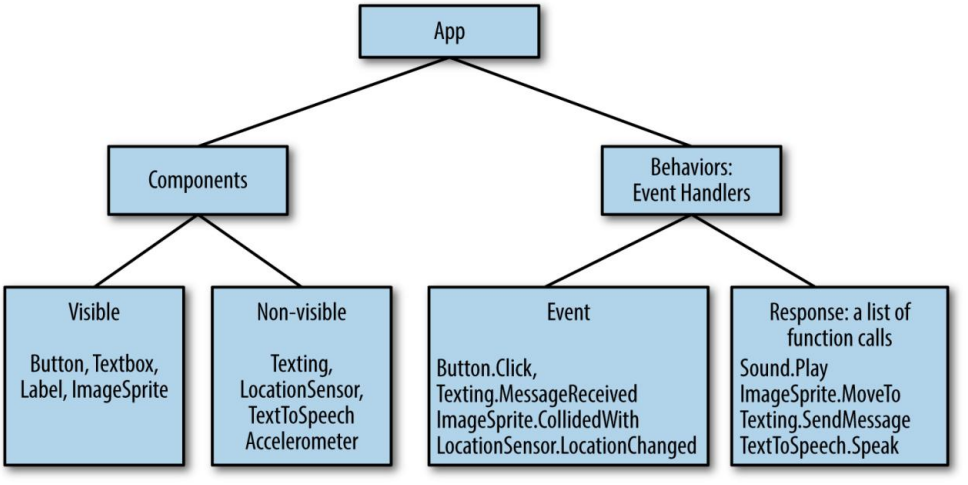
איור .14-1

פרק זה בוחן את המבנה של אפליקציה מנקודת מבטו של מתכנת. זה מתחיל באנלוגיה המסורתית שאפליקציה היא כמו מתכון ואז ממשיך להמשיג מחדש אפליקציה כמערכת של רכיבים המגיבים לאירועים. הפרק גם בוחן כיצד אפליקציות יכולות לשאול שאלות, לחזור, לזכור ולדבר עם האינטרנט, כל אלו יתוארו ביתר פירוט בפרקים מאוחרים יותר.

אנשים רבים יכולים לומר לך ממה היא אפליקציה נקודת מבט של משתמש, אבל להבין מה זה מנקודת מבט של מתכנת היא מסובכת יותר. לאפליקציות יש מבנה פנימי שעליך להבין כדי ליצור אותם בצורה יעילה.

אחת הדרכים לתאר את החלקים הפנימיים של האפליקציה היא לחלק אותה לשני חלקים, מרכיביה והתנהגויותיה. בקירוב, אלה תואמים לשני החלונות העיקריים שבהם אתה משתמש ב-Inventor: ppAאתה משתמש Component Designer-בכדי לציין את האובייקטים (הרכיבים) של האפליקציה, ואתה משתמש Blocks Editor-בכדי לתכנת כיצד האפליקציה מגיבה למשתמש ולאירועים חיצוניים (התנהגות האפליקציה).

איור 14-1מספק סקירה כללית של ארכיטקטורת אפליקציה זו. בפרק זה, נעשה זאת לחקור את הארכיטקטורה הזו בפירוט.



איור .14-2הארכיטקטורה הפנימית של אפליקציית App Inventor

רכיבים

ישנם שני סוגים עיקריים של רכיבים באפליקציה: גלוי ולא גלוי. הרכיבים הגלויים של האפליקציה הם אלה שתוכל לראות כאשר האפליקציה מופעלת -לחצנים, תיבות טקסט ותוויות. אלה מכונה לעתים קרובות ממשק המשתמש של האפליקציה.

רכיבים שאינם גלויים הם אלו שאינך יכול לראות, כך שהם אינם חלק מהמשתמש מִמְשָׁק. במקום זאת, הם מספקים גישה לפונקציונליות המובנית של המכשיר; לדוגמה, רכיב ה-SMS שולח ומעבד טקסטים של ,SMSרכיב ה- LocationSensorקובע את מיקום המכשיר, ורכיב TextToSpeechמדבר.

הרכיבים שאינם נראים לעין הם הטכנולוגיה בתוך המכשיר -דבורים קטנות שעושות עבודות עבור האפליקציה שלך.

גם רכיבים גלויים וגם לא גלויים מוגדרים על ידי קבוצה של מאפיינים.

מאפיינים הם חריצי זיכרון לאחסון מידע על הרכיב. לרכיבים גלויים כמו לחצנים ותוויות יש מאפיינים כגון Width, Heightו- ,Alignmentאשר יחד מגדירים כיצד הרכיב נראה.

מאפייני הרכיב הם כמו תאי גיליון אלקטרוני: אתה משנה אותם Component Designer-ב כדי להגדיר את המראה הראשוני של רכיב. אתה יכול גם לשנות את הערכים עם בלוקים.

התנהגות

רכיבי האפליקציה הם בדרך כלל פשוטים וקלים להבנה: תיבת טקסט מיועדת להזנת מידע, כפתור מיועד ללחיצה וכן הלאה. ההתנהגות של אפליקציה, לעומת זאת, היא קשה מבחינה רעיונית ולעתים קרובות מורכבת. ההתנהגות מגדירה כיצד

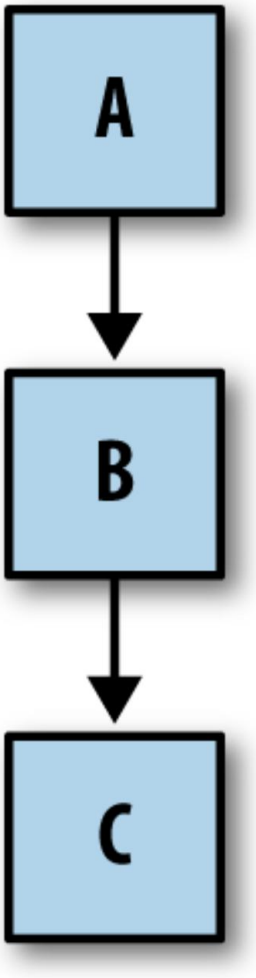
האפליקציה צריכה להגיב לאירועים, הן ביוזמת המשתמש (למשל, לחיצת כפתור) והן חיצונית (למשל, הודעת SMS שמגיעה לטלפון). הקושי לציין התנהגות אינטראקטיבית כזו הוא הסיבה לכך שתכנות הוא כל כך מאתגר.

למרבה המזל, App Inventor מספק שפה מבוססת בלוקים ברמה גבוהה לציון התנהגויות. הבלוקים הופכים התנהגויות תכנות ליותר כמו חיבור חלקי פאזל יחד, בניגוד לשפות תכנות מסורתיות מבוססות טקסט, הכוללות למידה והקלדה של כמויות קוד גדולות. ו-App Inventor נועד להקל במיוחד על ציון התנהגויות של תגובה לאירועים. הסעיפים הבאים מספקים מודל להבנת התנהגות האפליקציה וכיצד לציין אותה ב-App Inventor.

אפליקציה כמתכון

באופן מסורתי, תוכנה הושוותה לעתים קרובות למתכון. כמו מתכון, אפליקציה מסורתית עוקבת אחר רצף ליניארי של הוראות שעל המחשב לבצע, כמו למשל באיור 14-2.

אפליקציה טיפוסית עשויה להתחיל עסקה בנקאית (A), לבצע כמה חישובים ו שנה חשבון של לקוח (B), ולאחר מכן הדפס את היתרה החדשה על המסך (C).

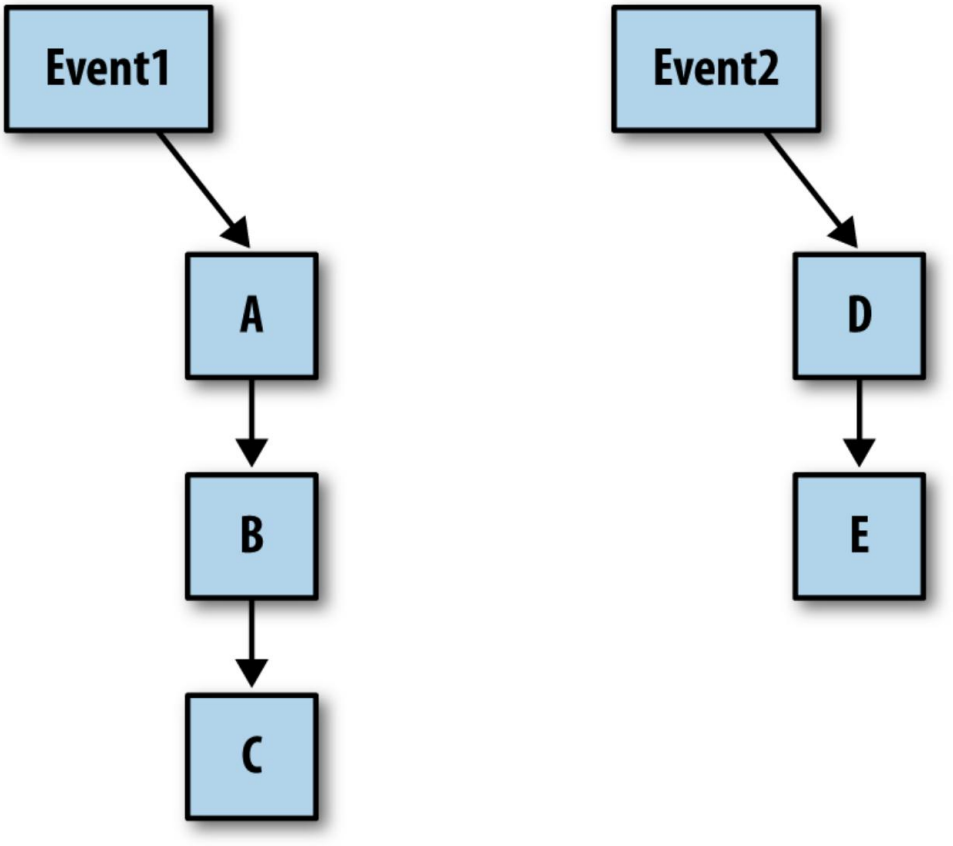


איור .14-3תוכנה מסורתית עוקבת אחר רצף ליניארי של הוראות

אפליקציה כקבוצה של מטפלי אירועים

האפליקציה כפרדיגמת מתכונים מתאימה היטב למחשב מוחץ המספרים המוקדם, אבל היא לא מתאימה לטלפונים ניידים, לאינטרנט ובכלל לרוב המחשוב שנעשה היום. רוב התוכנות המודרניות לא מבצעות שלל הוראות בסדר קבוע מראש; במקום זאת, הוא מגיב לאירועים - לרוב, אירועים שיזמו משתמש הקצה של האפליקציה. לדוגמה, אם המשתמש מקיש על כפתור, האפליקציה מגיבה בביצוע פעולה כלשהי (למשל, שליחת הודעת טקסט). עבור טלפונים ומכשירים עם מסך מגע, פעולת גרירת ה-regnF שלך על פני המסך היא אירוע נוסף. ה

האפליקציה עשויה להגיב לאותו אירוע על ידי ציור קו מהנקודה שבה הפנר שלך יוצר קשר ראשון עם המסך ועד לנקודה שבה הרמת אותו.
אפליקציות מודרניות מומשגות טוב יותר כמכונות תגובה לאירועים. האפליקציות כן כולל מתכונים -רצפים של הוראות -אך כל מתכון מבוצע רק בתגובה לאירוע כלשהו, כפי שמוצג באיור .14-3

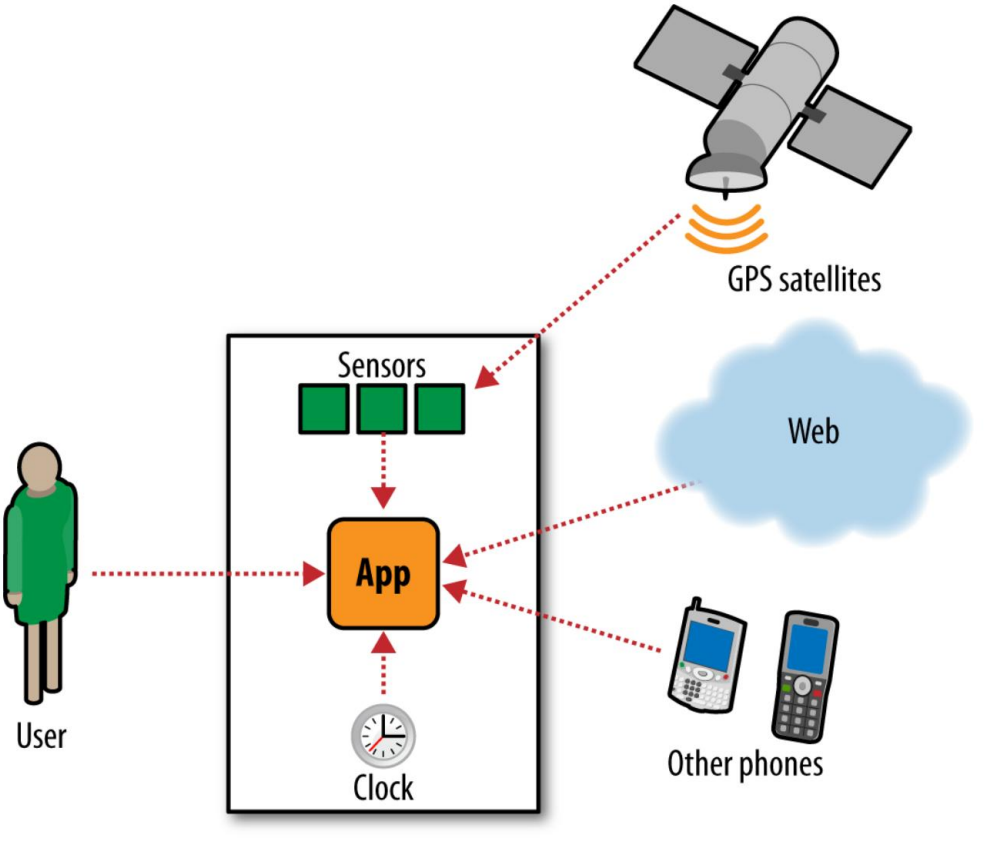


איור .14-4אפליקציה כמספר מתכונים המחוברים לאירועים

כאשר מתרחשים אירועים, האפליקציה מגיבה בקריאה לרצף של פונקציות. פונקציות הן דברים שאתה יכול לעשות עם, או עם, רכיב; אלה יכולות להיות פעולות כגון שליחת טקסט ,SMS או פעולות שינוי מאפיינים כגון שינוי הטקסט בתווית של ממשק המשתמש. להתקשר או להפעיל פונקציה פירושו לבצע את הפונקציה -לגרום לזה לקרות. אנו קוראים לאירוע ולקבוצת הפונקציות שבוצעו בתגובה לו מטפל באירועים.

אירועים רבים יוזמים על ידי משתמש הקצה, אך חלקם לא. אפליקציה יכולה להגיב לאירועים שקורים בטלפון, כגון שינויים בחיישן הכיוון שלה ובשעון (כלומר, הזמן החולף), או שהיא יכולה להגיב לאירועים שמקורם בחוץ

הטלפון, כגון הודעת טקסט או שיחה נכנסת מטלפון אחר, או נתונים המגיעים מהאינטרנט (ראה איור .(14-4



איור .14-5אפליקציה יכולה להגיב הן לאירועים פנימיים והן לאירועים חיצוניים

אחת הסיבות לכך שתכנות App Inventorהוא כל כך אינטואיטיבי היא שהיא מבוססת ישירות על פרדיגמת אירוע-תגובה זו; מטפלי אירועים הם פרימיטיביים בשפה (בהרבה שפות, זה לא המקרה). אתה מתחיל להגדיר התנהגות על ידי גרירת בלוק אירוע, שיש לו את הצורה, "do. "When <event>לדוגמה, שקול אפליקציה, ,SpeakItהמגיבה ללחיצות על כפתורים על ידי דיבור בקול של הטקסט שהמשתמש הקליד בתיבת טקסט. ניתן לתכנת יישום זה עם מטפל באירועים בודדים, כפי שמוצג באיור .14-5

איור .14-6מטפל באירועים עבור אפליקציית SpeakIt

בלוקים אלו מציינים שכאשר המשתמש לוחץ על הכפתור בשם ,SpeakItButton רכיב TextToSpeech צריך לומר את המילים שהמשתמש הקליד בתיבת הטקסט בשם .TextBox1 התגובה היא הקריאה לפונקציה .TextToSpeech1.Speak האירוע הוא .SpeakItButton.Click המטפל באירועים כולל את כל הבלוקים באיור .14-5

עם ,App Inventor כל הפעילות מתרחשת בתגובה לאירוע. האפליקציה שלך לא צריכה מכילים בלוקים מחוץ לאירוע של אירוע מתי כן לחסום. לדוגמה, הבלוקים באיור 14-6 לא משיגים דבר כשהם מוצפים לבד.

איור .14-7בלוקים צפים לא יעשו שום דבר מחוץ למטפל באירועים

סוגי אירועים

האירועים שיכולים לעורר פעילות נכנסים לקטגוריות המפורטות בטבלה .14-1

טבלה .14-1אירועים שיכולים לעורר פעילות

סוג האירוע	
אירוע ביוזמת משתמש כאשר המשתמש לוחץ על כפתור1, בצע...	
אירוע אתחול כאשר האפליקציה מופעלת, בצע...	
אירוע טיימר כשחלפות 20מילישניות, עשה...	
אירוע אנימציה כאשר שני עצמים מתנגשים, עשה...	
כאשר הטלפון מקבל הודעת טקסט, בצע...	אירוע חיצוני

אירועים ביוזמת המשתמש

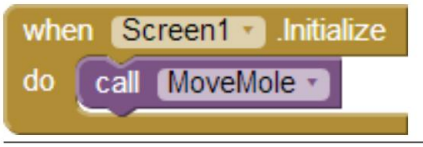
אירועים ביוזמת המשתמש הם סוג האירוע הנפוץ ביותר. עם טפסי קלט, זה בדרך כלל המשתמש שמקיש על כפתור שמפעיל תגובה מהאפליקציה. אפליקציות גרפיות נוספות מגיבות לנגיעות ולגרירות.

אירועי אתחול

לפעמים, האפליקציה שלך צריכה לבצע פונקציות מסוימות מיד עם ההפעלה, לא בתגובה לכל פעילות משתמש קצה או אירוע אחר. איך זה נכנס לפרדיגמת הטיפול באירועים?

שפות לטיפול באירועים כמו App Inventor מחשיבות את השקת האפליקציה כאירוע. אם אתה רוצה שפונקציות ספציפיות יבוצעו עם פתיחת האפליקציה, אתה גורר החוצה בלוק מסך אתחול אירוע ומציב בתוכו את בלוקי קריאת הפונקציות הרלוונטיים.1.

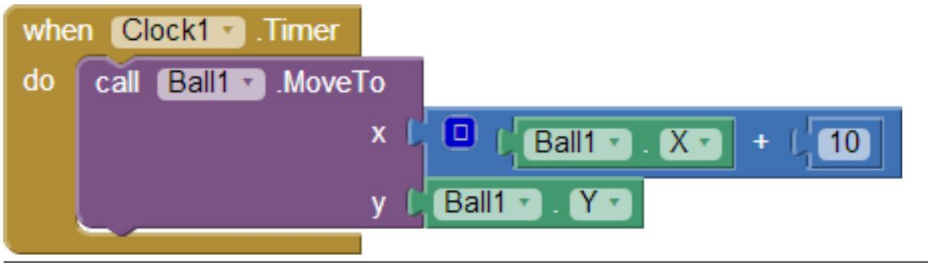
לדוגמה, במשחק MoleMash(פרק 3), הליך MoveMole נקרא עם הפעלת האפליקציה (ראה איור 14-7) כדי למקם את השומה באופן אקראי.



איור 14-8. שימוש בבלוק אירוע Screen1.Initialize כדי להזיז את השומה כאשר האפליקציה מופעלת

אירועי טיימר

פעילות מסוימת באפליקציה מופעלת על ידי הזמן החולף. אתה יכול לחשוב על אנימציה כאובייקט שזז כשהוא מופעל על ידי אירוע טיימר. ל-App Inventor יש רכיב שעון שבו אתה יכול להשתמש כדי להפעיל אירועי טיימר. לדוגמה, אם אתה רוצה שכדור על המסך יזוז 10 פיקסלים אופקית במרווח זמן מוגדר, הבלוקים שלך ייראו כמו איור 14-8.



איור 14-9. שימוש בבלוק אירוע טיימר כדי להזיז כדור בכל פעם ששעון 1. טיימר חדש

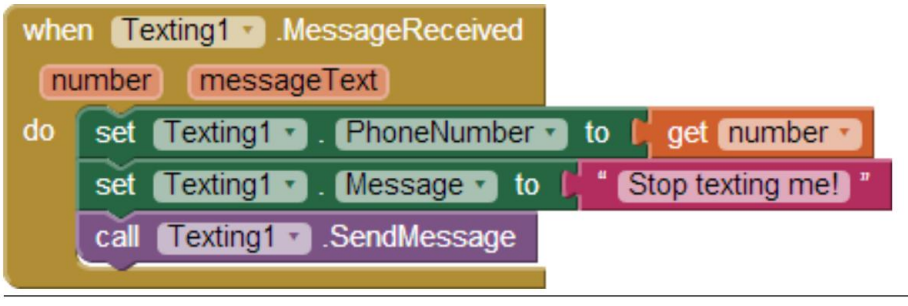
אירועי אנימציה

פעילות הכוללת אובייקטים גרפיים (ספרייטים) בתוך קנבסים תפעיל אירועים. אז אתה יכול לתכנת משחקים ואנימציות אינטראקטיביות אחרות על ידי ציון מה אמור להתרחש באירועים כגון אובייקט שמגיע לקצה הקנבס או שני אובייקטים מתנגשים, כפי שמתואר באיור 14-9. למידע נוסף, ראה פרק 17.

איור 14-10. כאשר הספרייט Flying Saucer פוגע באובייקט אחר, השמעת צליל

אירועים חיצוניים

כאשר הטלפון שלך מקבל מידע מיקום מלווייני GPS, מופעל אירוע. באופן דומה, כאשר הטלפון שלך מקבל הודעת טקסט, מופעל אירוע (איור 14-10).



איור 14-11. האירוע Texting1.MessageReceived מופעל בכל פעם שמתקבל טקסט

כניסות חיצוניות כאלה למכשיר נחשבות לאירועים, לא שונים מהלחץ של המשתמש על כפתור.
לפיכך, כל אפליקציה שתיצור תהיה קבוצה של מטפלים באירועים: אחד לאתחול דברים, חלק שיגיב לקלט של משתמש הקצה, חלק יופעל על ידי זמן, וחלק מופעל על ידי אירועים חיצוניים. התפקיד שלך הוא להמשיג את האפליקציה שלך בצורה זו ולאחר מכן לעצב את התגובה לכל מטפל באירועים.

מטפלי אירועים יכולים לשאול שאלות

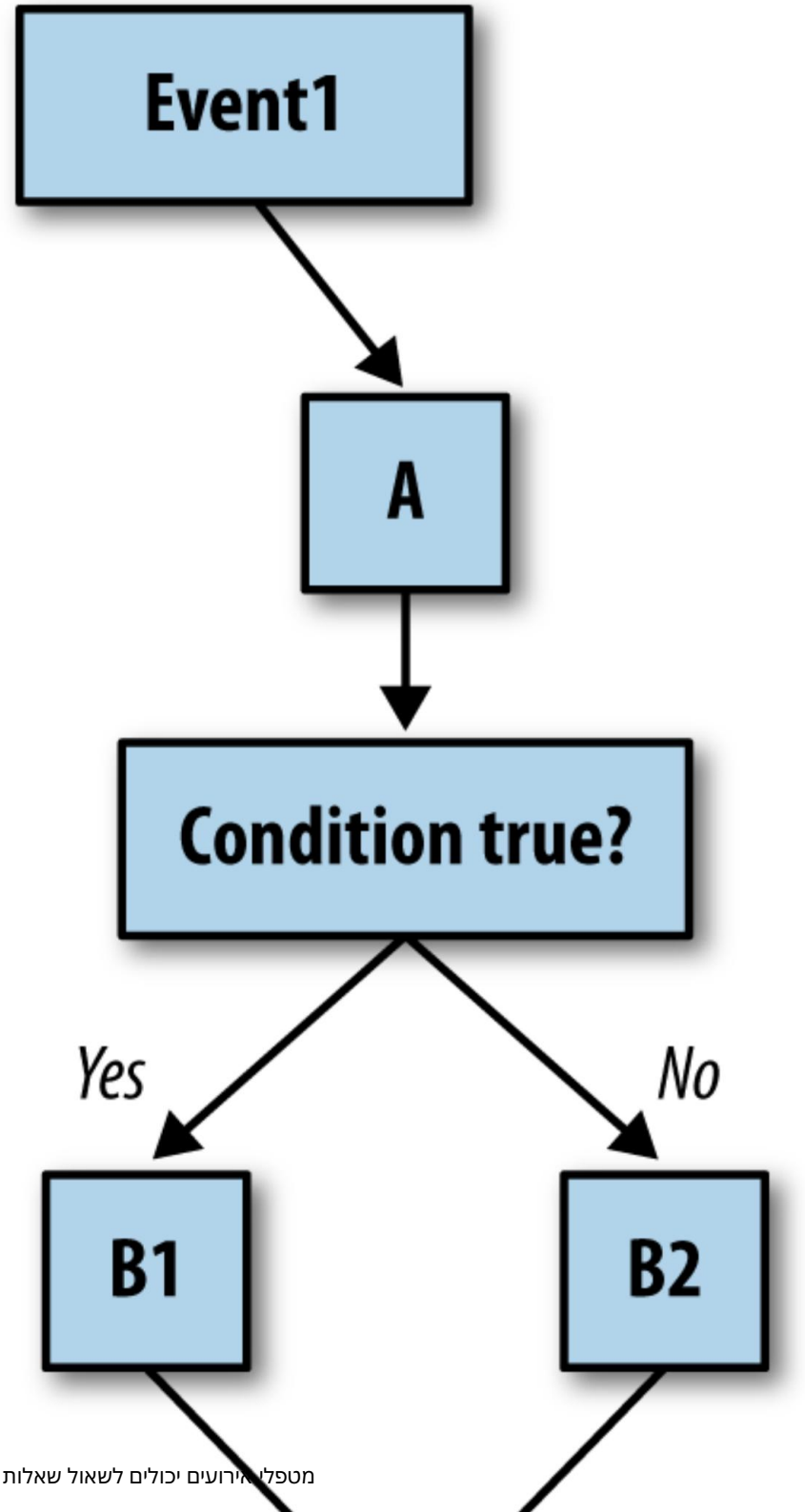
התגובות לאירועים אינן תמיד מתכונים ליניאריים; הם יכולים לשאול שאלות ולחזור על פעולות. "שאילת שאלות" פירושה שאילתה לנתונים שהאפליקציה מאחסנת ולקבוע את מהלך (הענף) שלה על סמך התשובות. אנחנו אומרים שלאפליקציות כאלה יש סניפים מותנים. איור 14-11 ממחיש בדיוק ענף כזה.
בתרשים, כאשר האירוע מתרחש, האפליקציה מבצעת פעולה A ולאחר מכן בודקת מצב. פונקציה B1 מבוצעת אם התנאי נכון. אם התנאי הוא

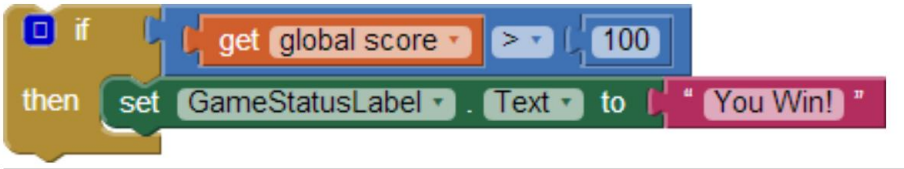
,falseהאפליקציה במקום זאת מבצעת .B2בכל מקרה, האפליקציה ממשיכה לבצע פונקציה C.

מבחנים מותנים הם שאלות כגון "האם הציון הגיע ל-001?" או "עשה את הודעת טקסט שקיבלתי זה עתה הגיעה מג'ו?" מבחנים יכולים להיות גם נוסחאות מורכבות יותר, כולל אופרטורים יחסיים מרובים (קטן מ, גדול מ, שווה ל) ואופרטורים לוגיים (ו, או, לא).

אתה מציין התנהגויות מותנות ב-ppA Inventorעל ידי שימוש בחסימות אם ואם אחרת . לדוגמה, הבלוק באיור 12-14ידווח "אתה מנצח!" אם השחקן קלע 100נקודות.

בלוקים מותנים נדונים בפירוט בפרק .18

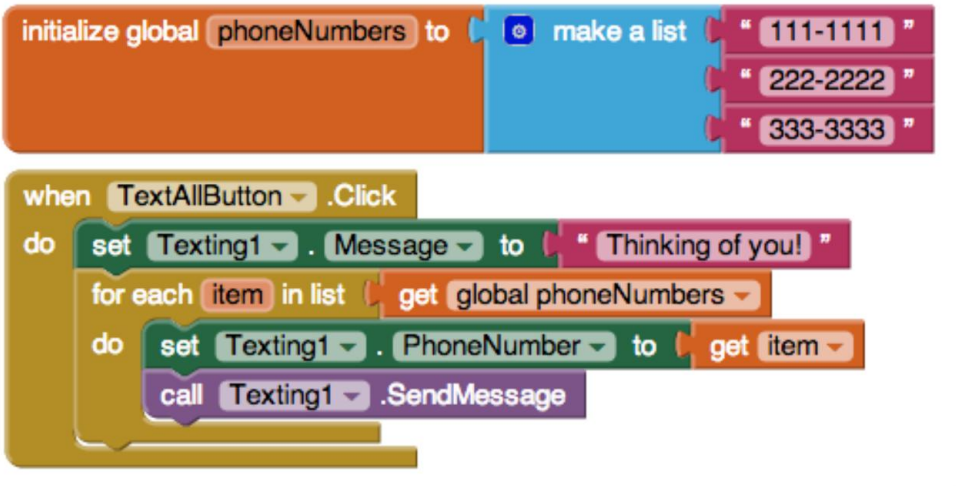




איור 14-13. שימוש בחסימה אם כדי לדווח על ניצחון כאשר השחקן מגיע ל-001 נקודות

מטפלי אירועים יכולים לחזור על חסימות

בנוסף לשאלת שאלות והסתעפות על סמך התשובה, תגובה לאירוע יכולה גם לחזור על פעולות מספר פעמים. App Inventorמספק מספר בלוקים לחזרה, כולל ה- while do. For each andשניהם מקיפים בלוקים אחרים. כל הבלוקים בתוך עבור כל אחד מבוצעים פעם אחת עבור כל פריט ברשימה. לדוגמה, אם תרצה לשלוח את אותה הודעה לרשימה של מספרי טלפון, תוכל להשתמש בבלוקים באיור 14-13.



איור 14-14. הבלוקים בתוך עבור כל בלוק חוזרים על עצמם עבור כל פריט ברשימה PhoneNumbers

החסימות בתוך ה- עבור כל בלוק חוזרות על עצמן -במקרה זה, שלוש פעמים, מכיוון שברשימה PhoneNumbersיש שלושה פריטים. בדוגמה זו, ההודעה "חושב עליך..." נשלחת לכל שלושת המספרים. בלוקים חוזרים נדונים בפירוט בפרק 20.

מטפלי אירועים יכולים לזכור דברים

מכיוון שמטפל באירועים מבצע חסימות, הוא צריך לעתים קרובות לעקוב אחר מידע. ניתן לאחסן מידע בחריצי זיכרון הנקראים משתנים, אותם אתה מגדיר בעורך הבלוקים. משתנים הם כמו מאפיינים של רכיבים, אבל הם לא משויכים לשום רכיב מסוים. באפליקציית משחק, למשל, אתה יכול להגדיר משתנה שנקרא ניקוד, ומטפלי האירועים שלך ישנו את הערך שלו כשהמשתמש עושה משהו בהתאם. משתנים מאחסנים נתונים באופן זמני בזמן שאפליקציה פועלת; כאשר אתה סוגר את האפליקציה, הנתונים אובדים ואינם זמינים יותר.

לפעמים, האפליקציה שלך צריכה לזכור דברים לא רק בזמן שהיא פועלת, אלא גם כשהיא סגורה ואז נפתחת מחדש. אם עקבת אחר ציון גבוה בהיסטוריה של משחק, למשל, תצטרך לאחסן את הנתונים האלה כך שהם יהיו זמינים בפעם הבאה שמישהו ישחק במשחק. נתונים שנשמרים גם לאחר סגירת אפליקציה נקראים נתונים מתמידים, והם מאוחסנים בסוג כלשהו של מסד נתונים.

נחקור את השימוש הן בזיכרון לטווח קצר (משתנים) והן בזיכרון לטווח ארוך זיכרון (נתוני מסד נתונים) בפרק 16ובפרק ,22בהתאמה.

מטפלי אירועים יכולים ליצור אינטראקציה עם האינטרנט

אפליקציות מסוימות משתמשות רק במידע שבטלפון או במכשיר. אבל אפליקציות רבות מתקשרות עם האינטרנט, או על ידי הצגת דף אינטרנט בתוך האפליקציה, או על ידי שליחת בקשות לממשקי APIשל שירותי אינטרנט (ממשקי תכנות יישומים). אומרים שאפליקציות כאלה הן "מותאמות לאינטרנט".

טוויטר הוא דוגמה לשירות אינטרנט שאיתו אפליקציית App Inventorיכולה לדבר. אתה יכול לכתוב אפליקציות שמבקשות ומציגות ציוצים קודמים של החברים שלך וגם לעדכן את סטטוס הטוויטר שלך. אפליקציות שמדברות עם יותר משירות אינטרנט אחד נקראות .mashups נסקור אפליקציות התומכות באינטרנט בפרק .24

סיכום

יוצר אפליקציה חייב לראות את האפליקציה שלו הן מנקודת מבט של משתמש הקצה והן מנקודת מבטו של מתכנת. עם ,App Inventorאתה מעצב איך אפליקציה נראית ואתה מעצב את ההתנהגות שלה -קבוצת המטפלים באירועים שגורמים לאפליקציה להתנהג כפי שאתה רוצה. אתה בונה את מטפלי האירועים האלה על ידי הרכבה והגדרת בלוקים המייצגים אירועים, פונקציות, ענפים מותנים, לולאות חוזרות, שיחות אינטרנט, פעולות מסד נתונים ועוד, ולאחר מכן בודקים את העבודה שלך על ידי הפעלת האפליקציה בטלפון שלך. לאחר שכותבים כמה תוכניות, מתברר המיפוי בין המבנה הפנימי של אפליקציה לביטוי הפיזי שלה. כשזה קורה, אתה מתכנת!

הנדסה וניפוי באגים באפליקציה

איור 15.1: HelloPurr, MoleMash והאפליקציות האחרות המכוסות בפרקים המוקדמים של ספר זה הם פרויקטי תוכנה קטנים יחסית ואינם דורשים כמות משמעותית של הנדסת תוכנה. ברגע שאתה לוקח על עצמך פרויקט מסובך יותר, אתה תבין שהקושי בבניית תוכנה גדל במהירות עבור כל פיסת מורכבות שאתה מוסיף -זה לא קרוב לקשר ליניארי.

מהר מאוד תלמד שכדי לבנות תוכנה מורכבת אפילו במידה בינונית, אתה צריך מחשבה מוקדמת, תכנון, שרטוטים, בדיקות משתמשים ומערכות, ובכלל, טכניקות ומיומנויות שהן יותר הנדסיות מתכנות. עבור רובנו, נדרשות כמה דפיקות קשות לפני שאנו מבינים את העובדה הזו. בשלב זה, אתה תהיה מוכן ללמוד כמה עקרונות הנדסת תוכנה וטכניקות ניפוי באגים. אם אתה כבר בשלב הזה, או אם אתה אחד מאותם אנשים מעטים שרוצים ללמוד כמה טכניקות בתקווה להימנע מכמה מאותם כאבי גדילה, הפרק הזה הוא בשבילך.

עקרונות הנדסת תוכנה

הנה כמה עקרונות בסיסיים שנעסוק בפרק זה:

- שתף את המשתמשים הפוטנציאליים שלך בתהליך מוקדם ובתדירות גבוהה ככל האפשר.
- בנו אב טיפוס ראשוני ופשוט ולאחר מכן הוסיפו לו בהדרגה.
- קוד ובדוק במרווחים קטנים, לעולם לא יותר מכמה בלוקים בכל פעם.
- עצב את ההיגיון עבור האפליקציה שלך לפני תחילת הקוד.
- הפרד, השכבה, וכבוש.
- הגיבו לחסימות שלכם כדי שאחרים (וגם אתם) תוכלו להבין אותם.
- למד להתחקות אחר בלוקים בעזרת עיפרון ונייר כדי שתבין אותם מֵכָנִיקָה.

אם תמלא אחר העצה הזו, תחסוך לעצמך זמן ותסכול ותבנה טוב יותר תוֹכנָה. אבל, כנראה שלא תעקבו אחריו בכל פעם! חלק מהעצות הללו עשויות

נראה מנוגד לאינטואיציה. הנטייה הטבעית שלך היא לחשוב על רעיון, להניח שאתה יודע מה המשתמשים שלך רוצים, ואז להתחיל לחבר בלוקים עד שאתה חושב שסיימת את האפליקציה. בוא נחזור לעקרון הראשון ונראה איך להבין מה המשתמשים שלך רוצים לפני שתתחיל לבנות משהו.

לפתור בעיות אמיתיות

בסרט "שדה החלומות", הדמות ריי שומעת קול לוחש, "אם תבנה אותו, [הם] יבואו". ריי מקשיב ללחש, בונה שדה בייסבול באמצע חלקת התירס שלו באיווה, ואכן, ה-White Sox מ-9191 ואלפי מעריצים מופיעים.

כדאי לדעת כבר עכשיו שעצת הלוחש לא חלה על תוכנה.

למעשה, זה הפוך ממה שאתה צריך לעשות. ההיסטוריה של התוכנה עמוסה בפתרונות נהדרים שאין להם שום בעיה. פתרון בעיה אמיתית הוא מה שעושה אפליקציה מדהימה ופרויקט מוצלח ואולי משתלם. וכדי לדעת מה הבעיה, אתה צריך לדבר עם האנשים שיש להם את זה. זה מכונה לעתים קרובות עיצוב ממוקד משתמש, וזה יעזור לך לבנות אפליקציות טובות יותר.

אם אתה פוגש כמה מתכנתים, שאל אותם כמה אחוז מהתוכניות שהם כתבו נפרסו בפועל עם משתמשים אמיתיים. תתפלאו עד כמה האחוז נמוך, אפילו עבור מתכנתים גדולים. רוב פרויקטי התוכנה נתקלים בבעיות רבות כל כך שהם אף פעם לא רואים אור.

עיצוב ממוקד משתמש פירושו לחשוב ולדבר עם משתמשים פוטנציאליים מוקדם ולעתים קרובות. באמת, זה צריך להתחיל עוד לפני שאתה מחליט מה לבנות. התוכנה המצליחה ביותר נבנתה כדי לפתור את נקודת הכאב של אדם מסוים, ואז -ורק אז -הוכללה לדבר הגדול הבא.

בנו אב טיפוס והצג משתמשים

רוב המשתמשים הפוטנציאליים לא יספקו משוב שימושי אם תבקשו מהם לקרוא מסמך המפרט מה האפליקציה תעשה ולתת את המשוב שלהם על סמך זה.

מה שכן עובד זה להראות להם מודל אינטראקטיבי לאפליקציה שאתה הולך ליצור -אב טיפוס. אב טיפוס הוא גרסה לא שלמה, לא מעודכנת של האפליקציה.

כאשר אתה בונה אותו, אל תדאג לפרטים או לשלמות או לממשק גרפי יפה; בנה אותו כך שיעשה מספיק כדי להמחיש את הערך המרכזי של האפליקציה. לאחר מכן, הצג את זה למשתמשים הפוטנציאליים שלך, היה בשקט והקשב.

פיתוח מצטבר

כשאתה מתחיל את האפליקציה הראשונה שלך בגודל משמעותי, הנטייה הטבעית שלך עשויה להיות להוסיף את כל הרכיבים והבלוקים שתצטרך במאמץ גדול אחד ולאחר מכן להוריד את האפליקציה לטלפון שלך כדי לראות אם היא עובדת. קח, למשל, אפליקציית חידון.

ללא הדרכה, רוב המתכנתים המתחילים יוסיפו בלוקים עם רשימה ארוכה של השאלות והתשובות, בלוקים לטיפול בניווט החידון, בלוקים לטיפול בבדיקת תשובת המשתמש, וחסימות לכל פרט בלוגיקה של האפליקציה, הכל לפני הבדיקה כדי לראות אם כל זה עובד. בהנדסת תוכנה קוראים לזה גישת המפץ הגדול.

כמעט כל מתכנת חדש משתמש בגישה זו. בשיעורים שלי (המחבר וולבר) באוניברסיטת סן פרנסיסקו, אשאל לעתים קרובות סטודנט, "איך הולך?" כשהתלמיד עובד על אפליקציה.

"אני חושב שסיימתי," יענה התלמיד.

"נֶהְדָר. אני יכול לראות את זה?"

"אמם, עדיין לא; אין לי את הטלפון שלי איתי."

"אז לא הפעלת את האפליקציה בכלל?" אני שואל.

"לא."

אני אסתכל מעבר לכתפו של התלמיד בקונפיגורציה מדהימה וצבעונית של 30או בלוקים, אף אחד מהם לא נבדק. הבעיה היא שכאשר אתה בודק בבת אחת, הרבה יותר קשה לאבחן את הבאגים, ויהיו באגים -גדולים שעירים!

כנראה העצה הטובה ביותר שאני יכול לתת לתלמידים שלי —ומתכנתים שואפים בכל מקום -האם זה:

קוד קצת, בדוק קצת, חזור.

בנה את האפליקציה שלך חתיכה אחת בכל פעם, תוך כדי בדיקה. אתה תמצא באגים, בסדר, אבל זעירים שאתה יכול בקלות להחליק. והתהליך יהפוך לספק באופן מפתיע, כי תראה תוצאות מוקדם יותר כשתעקוב אחריו.

מאות ספרים ותזה נכתבו על תוכנות מצטברות התפתחות. אם אתה מעוניין בתהליך בניית תוכנה (ודברים אחרים), בדוק את מתודולוגיית הפיתוח הזריז.[1]

עיצוב לפני קידוד

יש שני חלקים לתכנות: הבנת ההיגיון של האפליקציה, ולאחר מכן תרגום ההיגיון הזה לקוד בשפת תכנות כלשהי. לפני שתתמודד עם התרגום, הקדיש זמן מה להגיון. ציין מה צריך לקרות גם עבור המשתמש וגם פנימי באפליקציה. בדוק את ההיגיון של כל מטפל באירועים לפני שתמשיך לתרגם את ההיגיון הזה לבלוקים.

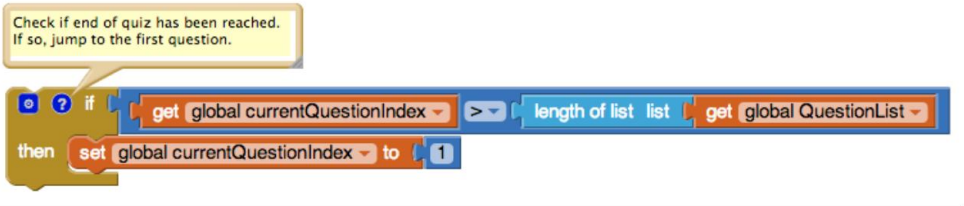
ספרים שלמים נכתבו על מתודולוגיות שונות של עיצוב תוכניות. יש אנשים שמשתמשים בדיאגרמות כגון תרשימי מבנה או תרשימי מבנה לעיצוב, בעוד שאחרים מעדיפים טקסט וסקיצות בכתב יד. יש אנשים המאמינים שכל "עיצוב" צריך

1בק, קנט; et al. (2001). "מניפסט לפיתוח תוכנה זריז". Agile Allianceאוחזר ב-5 ביוני 2014

בסופו של דבר ישירות לצד הקוד שלך כהערה (הערות), לא במסמך נפרד. המפתח למתחילים הוא להבין שיש היגיון בכל התוכנות שאין לו שום קשר לשפת תכנות מסוימת. התמודדות בו-זמנית עם ההיגיון הזה וגם התרגום שלו לשפה, לא משנה כמה השפה אינטואיטיבית, יכולה להיות מהממת. לכן, לאורך כל התהליך, התרחק מהמחשב וחשוב על האפליקציה שלך, ודא שאתה ברור מה אתה רוצה שהיא תעשה, ותעד את מה שאתה ממציא בדרך כלשהי. לאחר מכן, הקפד לחבר את תיעוד העיצוב הזה לאפליקציה שלך כדי שאחרים יוכלו ליהנות ממנו.

הערה את הקוד שלך

אם השלמת כמה מהמדריכים בספר זה, כנראה שראית את הקופסאות הצהובות הקטנות בתוך הבלוקים (ראה איור .15-1). אלה נקראים הערות. ב-Inventor, ppA אתה יכול להוסיף הערות לכל בלוק על ידי לחיצה ימנית עליו ובחירה הוסף תגובה. הערות הן רק הערות; הם אינם משפיעים כלל על הפעלת האפליקציה.



איור 15-2. שימוש בהערה על בלוק if כדי לתאר מה הוא עושה באנגלית פשוטה

אז למה להגיב? ובכן, אם האפליקציה שלך מצליחה, היא תחיה חיים ארוכים. גם לאחר שהייתם במרחק של שבוע בלבד מהאפליקציה שלכם, אתם תשכחו מה חשבתם באותו זמן ולא יהיה לכם מושג למה נועדו חלק מהבלוקים. מסיבה זו, גם אם אף אחד אחר לא יראה את החסימות שלך, עליך לספק הערות עבורם.
ואם האפליקציה שלכם תצליח, היא ללא ספק תעבור בידיים רבות. אֲנָשִׁים ירצה להבין אותו, להתאים אותו ולהרחיב אותו. ברגע שתתקלו בחוויה הנפלאה של פתיחת פרויקט עם קוד של מישהו ללא הערות, תבינו לגמרי למה הערות חיוניות.

הערת תוכנית אינה אינטואיטיבית, ומעולם לא פגשתי מתכנת מתחיל שחשב שזה חשוב. לעומת זאת, גם מעולם לא פגשתי מתכנת מנוסה שלא עשה זאת.

חלק, שכבה וכבוש

הבעיות הופכות למכריעות כשהן גדולות מדי. המפתח הוא לפרק בעיה. ישנן שתי דרכים עיקריות לעשות זאת. הדבר שאנחנו הכי מכירים הוא לשבור

בעיה למטה לחלקים (A, B, C)והתמודד עם כל אחד בנפרד. דרך שנייה, פחות נפוצה, היא לפרק בעיה לשכבות מפשוטה למורכבת. הוסף כמה בלוקים להתנהגות פשוטה כלשהי, בדוק את התוכנה כדי לוודא שהיא מתנהגת כפי שאתה רוצה, ולאחר מכן הוסף עוד שכבה של מורכבות, וכן הלאה.
באמצעות אפליקציית חידון הנשיא בפרק 10כדוגמה, בואו נעריך את שני אלה שיטות. נזכיר כי אפליקציית חידון הנשיא מאפשרת למשתמש לנווט בין השאלות על ידי לחיצה על כפתור הבא. זה גם בודק את התשובות של המשתמש כדי לקבוע אם היא נכונה. לכן, בעיצוב האפליקציה הזו, אתה עשוי לחלק אותה לשני חלקים -ניווט בשאלות ובדיקת תשובות, ותכנת כל אחד בנפרד.
עם זאת, בתוך כל אחד משני החלקים הללו, אתה יכול גם לפרק את התהליך מפשוט למורכב. לכן, עבור ניווט שאלות, התחל ביצירת הקוד כדי להציג רק את השאלה הראשונה ברשימת השאלות, ובדוק אותה כדי לוודא שהיא עובדת. לאחר מכן, בנה את הקוד כדי להגיע לשאלה הבאה, אך התעלם מהנושא של מה קורה כאשר אתה מגיע לשאלה האחרונה. לאחר שאישרת שהחידון ייקח אותך לסוף, הוסף את הבלוקים כדי לטפל ב"מקרה המיוחד" של המשתמש שהגיע לשאלה האחרונה.

זה לא מקרה של או/או אם אתה צריך לפרק בעיה לחלקים
או לשכבות של מורכבות: כדאי לעשות את שניהם. לאלה שיכולים לעשות זאת היטב -אדריכלי תוכנה -יש ביקוש גבוה ביותר.

להבין את השפה שלך: מעקב באמצעות עט ו עיתון

כאשר אפליקציה בפעולה, היא גלויה רק חלקית. משתמש הקצה של אפליקציה רואה רק את הפנים החיצוניות שלה, את התמונות והנתונים המוצגים בממשק המשתמש. פעולתה הפנימית של תוכנה מוסתרת לעולם החיצון, בדיוק כמו המנגנונים הפנימיים של המוח האנושי (למרבה המזל!). כאשר אפליקציה מופעלת, איננו רואים את ההוראות (בלוקים), איננו רואים את מונה התוכנה שעוקב אחר איזו הוראה מבוצעת כעת, ואיננו רואים את תאי הזיכרון הפנימיים של התוכנה (המשתנים והמאפיינים שלה) . בסופו של דבר, זה איך שאנחנו רוצים את זה: משתמש הקצה צריך לראות רק את מה שהתוכנה מציגה במפורש. עם זאת, בזמן שאתה מפתח ובודק תוכנה, אתה רוצה לראות את כל מה שקורה.

אתה, המתכנת, רואה את הקוד במהלך הפיתוח, אבל רק תצוגה סטטית של
זה. לפיכך, עליך לדמיין את התוכנה בפעולה: מתרחשים אירועים, מונה התוכניות עובר לבלוק הבא ומבצע אותו, הערכים בתאי הזיכרון משתנים, וכן הלאה.

תכנות דורש מעבר בין שתי תצוגות שונות. אתה מתחיל עם המודל הסטטי -בלוקי הקוד -ומנסה לדמיין כיצד התוכנית תתנהג בפועל. כאשר אתה מוכן, אתה עובר למצב בדיקה: משחק בתפקיד משתמש הקצה ובדיקת התוכנה כדי לראות אם היא מתנהגת כפי שאתה מצפה. אם לא, אתה

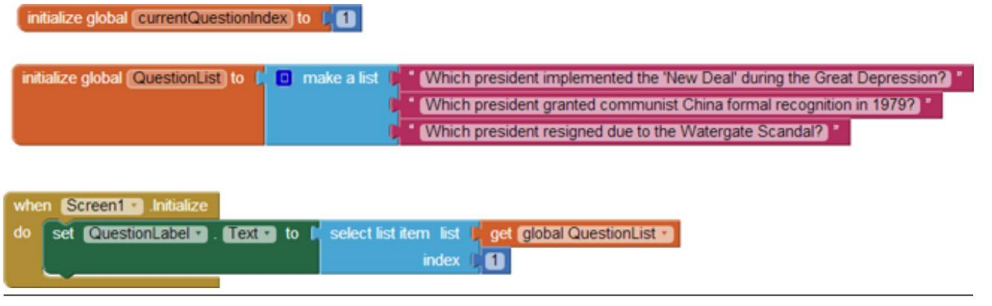
חייב לחזור לתצוגה הסטטית, לשנות את המודל שלך ולבדוק שוב. בתהליך זה הלוך ושוב, אתה מתקדם לעבר פתרון מקובל.

כאשר אתה מתחיל לתכנת, יש לך רק מודל חלקי של איך מחשב התוכנית עובדת - כל התהליך נראה כמעט קסום. אתה מתחיל עם כמה אפליקציות פשוטות: לחיצה על כפתור גורמת לחתול מיאו! לאחר מכן אתה עובר ליישומים מורכבים יותר, עוברים על כמה מדריכים, ואולי מבצעים כמה שינויים כדי להתאים אותם. המתחיל מבין חלקית את פעולתן הפנימית של האפליקציות, אבל בהחלט לא מרגיש שליטה בתהליך. המתחיל יגיד לעתים קרובות, "זה לא עובד," או, "זה לא עושה את מה שהוא אמור לעשות." המפתח הוא ללמוד כיצד הדברים עובדים עד כדי כך שאתה חושב בצורה סובייקטיבית יותר על התוכנית ואומר דברים כגון, "התוכנית שלי עושה את זה," ו"ההיגיון שלי גורם לתוכנית ל...."

אחת הדרכים ללמוד איך תוכניות עובדות היא לעקוב אחר הביצוע של אפליקציה פשוטה כלשהי, מייצג על הנייר בדיוק מה קורה בתוך המכשיר כאשר כל בלוק מתבצע. דמיין את המשתמש מפעיל איזה מטפל באירועים ואז עובר ומראה את ההשפעה של כל בלוק: כיצד משתנים המשתנים והמאפיינים באפליקציה? כיצד משתנים הרכיבים בממשק המשתמש? כמו קריאה צמודה בשיעור ספרות, המעקב הזה שלב אחר שלב מאלץ אותך לבחון את מרכיבי השפה - במקרה זה, App Inventor חוסם.

המורכבות של המדגם שאתה מתחקה כמעט ואינו מהותי; המפתח הוא שאתה מאט את תהליך החשיבה שלך ותבדוק את הסיבה והתוצאה של כל בלוק.
בהדרגה תתחילו להבין שהכללים השולטים בתהליך כולו אינם מכריעים כפי שחשבתם במקור.

לדוגמה, שקול את הבלוקים המתוארים באיור 15-2, שהם קלים שינויים באפליקציית החידון של הנשיא (פרק 8).



איור 15-3. הגדרת הטקסט של QuestionLabel בלפריט הראשון של QuestionList בכאשר האפליקציה מתחילה

אתה מבין את הקוד הזה? האם תוכל להתחקות אחר זה ולהראות בדיוק מה קורה בכל שלב?

אתה מתחיל להתחקות על ידי ציור תיבות תאי זיכרון עבור כל המשתנים הרלוונטיים נכסים. במקרה זה, אתה צריך תיבות עבור ה- currentQuestionIndex- ו QuestionLabel.Text, כפי שמוצג בטבלה .15-1

טבלה .15-1 קופסאות תא זיכרון למעקב

currentQuestionIndex	QuestionLabel.Text

לאחר מכן, חשבו מה קורה כאשר אפליקציה מתחילה -לא מנקודת המבט של המשתמש, אלא פנימית, בתוך האפליקציה כשהיא מאתחלת. אם סיימתם חלק מהמדריכים, אתם בטח יודעים זאת, אבל אולי לא חשבתם על זה במונחים מכניים. כאשר אפליקציה מתחילה:

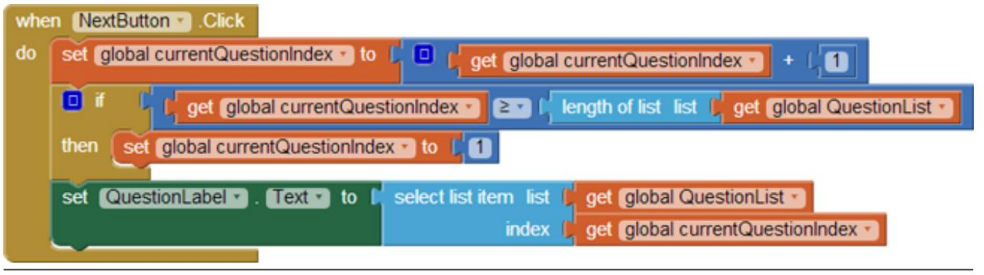
1. כל מאפייני הרכיב מוגדרים על סמך הערכים ההתחלתיים שלהם .Designer ב-Component
2. כל ההגדרות והאתחולים המשתנים מבוצעים.
3. הבלוקים במטפל האירוע Screen.Initialize מבוצעים.

מעקב אחר תוכנית עוזר לך להבין את המכניקה הזו. אז מה צריך להיכנס את הקופסאות לאחר שלב האתחול?
כפי שמוצג בטבלה ,15-2 ה-1 נמצא ב- currentQuestionIndex מכיוון שההגדרה המשתנה מבוצעת כשהאפליקציה מתחילה, והיא מאתחלת אותה ל-1. השאלה הראשונה נמצאת ב- QuestionLabel.Text מכיוון Screen.Initialize- שבוחר את הפריט הראשון מתוך QuestionList ו שם את זה שם.

טבלה .15-2 הערכים לאחר אתחול אפליקציית חידון הנשיא

currentQuestionIndex	QuestionLabel.Text
1	איזה נשיא יישם את "הדיל החדש" במהלך השפל הגדול?

לאחר מכן, עקוב אחר מה קורה כאשר המשתמש לוחץ על כפתור הבא, באמצעות הבלוקים מוצג באיור .15-3



איור 15-4. חסימה זו מבוצעת כאשר המשתמש לוחץ על NextButton

בחנו כל בלוק, אחד אחד. ראשית, ה- currentQuestionIndex מוגדל.
ברמה מפורטת עוד יותר, הערך הנוכחי של המשתנה (1) מתווסף ל-1, והתוצאה (2) ממוקמת
ב- currentQuestionIndex. המשפט if הוא שקר מכיוון שהערך של (2)
currentQuestionIndex קטן מהאורך של QuestionList (3).
לכן, הפריט השני נבחר ומוכנס לתוך QuestionLabel.Text, כפי שמוצג בטבלה 15-3.

טבלה 15-3. הערכים לאחר לחיצה על NextButton

currentQuestionIndex	QuestionLabel.Text
2	איזה נשיא העניק לסין הקומוניסטית הכרה רשמית ב-1979?

עקוב אחר מה שקורה בלחיצה השנייה. כעת, currentQuestionIndex מוגדל והופך
ל-3. מה קורה עם ה- if? לפני הקריאה הקדימה, בחן אותו מקרוב ובדוק אם אתה יכול לאתר
אותו כראוי.
במבחן if, הערך של (3) currentQuestionIndex גדול או שווה לאורך של
QuestionList. כתוצאה מכך, ה- currentQuestionIndex מוגדר ל-1 והשאלה הראשונה
ממוקמת בתווית, כפי שמוצג בטבלה 15-4.

טבלה 15-4. הערכים לאחר לחיצה שנייה על NextButton

currentQuestionIndex	QuestionLabel.Text
1	איזה נשיא יישם את "הדיל החדש" במהלך השפל הגדול?

המעקב חשף באג: השאלה האחרונה ברשימה לעולם לא מופיעה! האם אתה
יודע איך לתקן את זה?

כאשר אתה יכול לעקוב אחר אפליקציה לרמת פירוט זו, אתה הופך למתכנת, א
מהנדס. אתה מתחיל להבין את המכניקה של שפת התכנות, קליטת משפטים ומילים בקוד
במקום לתפוס פסקאות במעורפל.

כן, שפת התכנות מורכבת, אבל לכל "מילה" יש פרשנות ברורה וישירה על ידי המכונה. אם אתה מבין איך כל בלוק ממפה למשתנה או תכונה כלשהי שמשתנה, אתה יכול להבין איך לכתוב או לתקן את האפליקציה שלך. אתה מבין שאתה בשליטה מלאה.

עכשיו, אם הייתם אומרים לחברים שלכם, "אני לומד כיצד לתת למשתמש ללחוץ על כפתור הבא כדי להגיע לשאלה הבאה; זה ממש קשה," הם יחשבו שאתה משוגע. למעשה, תכנות כזה הוא מאוד קשה, לא בגלל שהמושגים כל כך מורכבים, אלא בגלל שאתה צריך להאט את המוח שלך כדי להבין איך הוא, או מחשב, מעבד כל שלב ושלב, כולל הדברים האלה שהמוח שלך עושה בתת מודע.

איתור באגים באפליקציה

מעקב אחר אפליקציה צעד אחר צעד, על הנייר, היא אחת הדרכים להבין תכנות; זוהי גם שיטה בדוקה לאיתור באגים באפליקציה כאשר יש לה בעיות.
סביבות תכנות, כולל ,App Inventorמספקות גם את גרסת ההייטק של איתור עט-נייר באמצעות כלי איתור באגים שמאוטמים חלק מהתהליך. כלים כאלה משפרים את תהליך פיתוח האפליקציה על ידי מתן תצוגה מוארת של אפליקציה בפעולה. כלים אלו מאפשרים למתכנת לבצע את הפעולות הבאות:

- השהה אפליקציה בכל נקודה ובחן את המשתנים והמאפיינים שלה
- בצע הוראות פרטניות (בלוקים) כדי לבחון את השפעותיהן

צופה במשתנים

הערכים של מאפיינים ומשתנים של רכיבים אינם גלויים כאשר אתה בודק אפליקציה ב-ppA Inventor.טכניקת איתור באגים נפוצה היא הוספת בלוקים כדי להציג ערכים אלה בתוויות של ממשק המשתמש במהלך הבדיקה, ולאחר מכן להסיר את התוויות ולהציג את הקוד לאחר ניפוי באגים של האפליקציה.
לגרסה הקודמת של (App Inventor Classic) App Inventorהיה מנגנון לצפייה בערכי משתנים ומאפיינים בעורך הבלוקים בזמן בדיקה, מבלי להשתמש בתוויות בממשק המשתמש. התוכנית היא שמנגנון כזה יתווסף גם ל-ppA ,2 Inventorאז שימו לב אליו כי הוא עוזר מאוד באיתור באגים ובהבנת קוד.

בדיקת בלוקים בודדים

אמנם אתה יכול להשתמש במנגנון ה-hctaW כדי לבחון משתנים במהלך הפעלת אפליקציה, אבל כלי אחר בשם Do It מאפשר לך להתנסות בנפרד

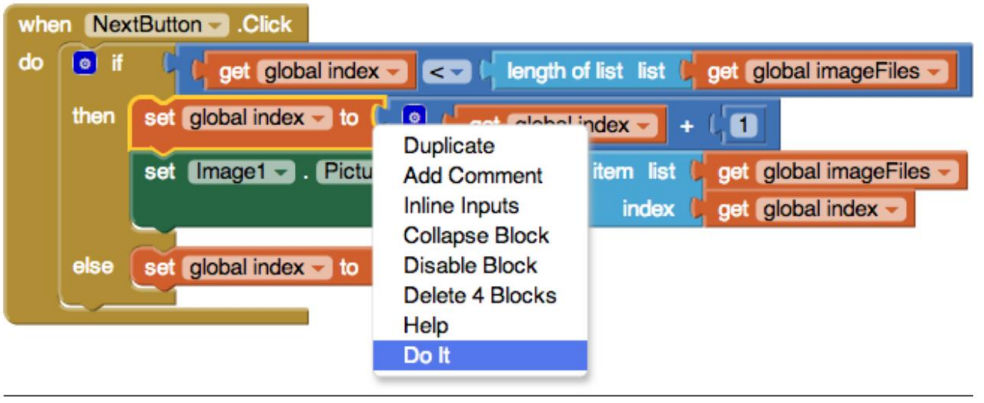
בלוקים מחוץ לרצף הביצוע הרגיל. לחץ לחיצה ימנית על כל בלוק ובחר עשה זאת; החסימה תתבצע. אם הבלוק הוא ביטוי שמחזיר ערך, App Inventor יציג את הערך הזה בתיבה מעל הבלוק.

האם זה שימושי מאוד לאיתור בעיות לוגיקה בלוקים שלך. שקול שוב את מטפל האירוע NextButton.Click של החידון , ונניח שיש לו בעיה לוגית שבה אתה לא מנווט בין כל השאלות. אתה יכול לבדוק את התוכנית על ידי לחיצה על הבא בממשק המשתמש ולבדוק אם השאלה המתאימה מופיעה בכל פעם. אולי אפילו תצפה ב- currentQuestionIndex כדי לראות כיצד כל קליק משנה אותו.

למרבה הצער, סוג זה של בדיקות מאפשר לך רק לבחון את ההשפעה של מטפלי אירועים שלמים. האפליקציה תבצע את כל החסימות במטפל באירועים עבור לחיצת הכפתור לפני שתאפשר לך לבחון את משתני ה-hctaW שלך או את ממשק המשתמש.

בעזרת הכלי Do It, תוכלו להאט את תהליך הבדיקה ולבחון את מצב האפליקציה לאחר כל חסימה. הסכימה הכללית היא ליזום אירועי ממשק משתמש עד שתגיע לנקודת הבעיה באפליקציה. לאחר שגילית שהשאלה השלישית לא מופיעה באפליקציית החידון, תוכל ללחוץ פעם אחת על הלחצן Next כדי להגיע לשאלה השנייה. לאחר מכן, במקום ללחוץ שוב על NextButton ולבצע את כל המטפל באירועים במכה אחת, תוכל להשתמש ב-oD It כדי לבצע את החסימות בתוך ה- NextButton.Click מטפל באירועים, אחד בכל פעם. תתחיל בלחיצה ימנית על שורת הבלוקים העליונה (התוספת של currentQuestionIndex) ובחירה ב-oD It, כפי שמוצג באיור 15-4.

זה ישנה את האינדקס ל-3. לאחר מכן הפעלת האפליקציה תיפסק Do It - גורם לביצוע רק הבלוק הנבחר וכל חסימות כפופות. זה מקנה לך, הבוחן, את היכולת לבחון את המשתנים הנצפים ואת ממשק המשתמש. כשתהיה מוכן, תוכל לבחור את שורת הבלוקים הבאה (בדיקת אם) ולבחור ב-oD It כך שהיא תתבצע. בכל שלב של הדרך, אתה יכול לראות את ההשפעה של כל בלוק.



איור 15-5. שימוש בכלי Do It כדי לבצע את הבלוקים אחד בכל פעם

פיתוח מצטבר עם Do It

חשוב לציין שביצוע בלוקים בודדים אינו מיועד רק לניפוי באגים. אתה יכול גם להשתמש בו במהלך הפיתוח כדי לבדוק בלוקים תוך כדי תנועה. לדוגמה, אם יצרת נוסחה ארוכה לחישוב המרחק במיילים בין שתי קואורדינטות ,GPSתוכל לבדוק את הנוסחה בכל שלב כדי לוודא שהבלוקים יוצרים

לְחוּש.

השבתת בלוקים

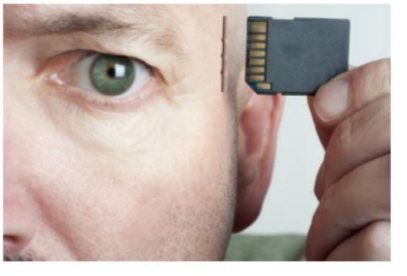
דרך נוספת לעזור לך לנפות באגים ולבדוק את האפליקציה שלך בהדרגה היא להשבית חסימות. על ידי כך, אתה יכול להשאיר חסימות בעייתיות או שלא נבדקו באפליקציה, אך לכוון את המערכת להתעלם מהם באופן זמני בזמן שהאפליקציה פועלת. לאחר מכן תוכל לבדוק את הבלוקים הפעילים ולגרום להם לעבוד במלואו מבלי לדאוג לאלו הבעייתיים. אתה יכול להשבית כל בלוק על ידי לחיצה ימנית עליו ובחירה באפשרות השבת חסימה. החסימה תופיע באפור, וכשתפעיל את האפליקציה תתעלם ממנה. כשתהיה מוכן, תוכל להפעיל את החסימה על ידי לחיצה ימנית עליו שוב ובחירה באפשרות הפעל חסימה.

סיכום

הדבר הגדול ב-AppInventorהוא כמה זה קל. האופי הוויזואלי שלה גורם לך להתחיל לבנות אפליקציה מיד, ואתה לא צריך לדאוג לגבי הרבה פרטים ברמה נמוכה. אבל, המציאות היא שממציא האפליקציות לא יכול להבין מה האפליקציה שלך צריכה לעשות בשבילך, הרבה פחות איך בדיוק לעשות את זה. למרות שזה מפתה פשוט לקפוץ ישר ל-Blocks Editor rengiseDולהתחיל לבנות אפליקציה, חשוב להקדיש זמן למחשבה ותכנון מפורט בדיוק מה האפליקציה שלך תעשה. זה נשמע קצת כואב, אבל אם תקשיב למשתמשים שלך, אבטיפוס, בודק ותעקוב אחר ההיגיון של האפליקציה שלך, אתה תבנה אפליקציות טובות יותר תוך זמן קצר.

תכנות זיכרון האפליקציה שלך

איור .1-16



בדיוק כמו שאנשים צריכים לזכור דברים, כך גם אפליקציות. פרק זה בוחן כיצד ניתן לתכנת אפליקציה לזכור מידע.
כשמישהו אומר לך את מספר הטלפון של פיצה, המוח שלך מאחסן אותו בחריץ זיכרון. אם מישהו קורא כמה מספרים כדי שתוכל להוסיף, אתה מאחסן את המספרים ותוצאות הביניים בחריצי זיכרון. במקרים כאלה, אינך מודע לחלוטין לאופן שבו המוח שלך אוגר מידע או זוכר אותו.

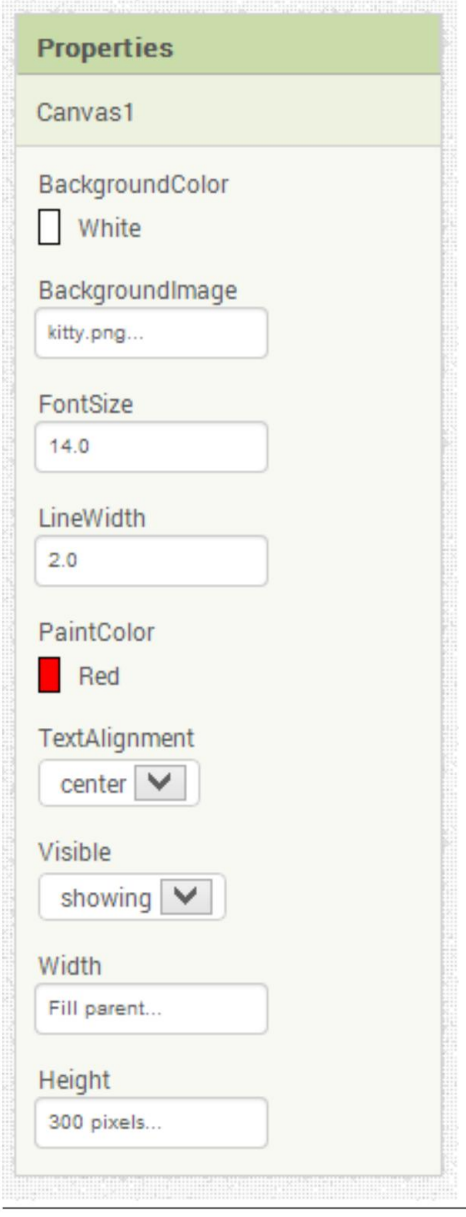
לאפליקציה יש גם זיכרון, אבל הפעולות הפנימיות שלה הרבה פחות מסתוריות מאלה של המוח שלך. בפרק זה תלמד כיצד להגדיר זיכרון של אפליקציה, כיצד לאחסן בו מידע וכיצד לאחזר מידע זה במועד מאוחר יותר.

בשם חריצי זיכרון

זיכרון של אפליקציה מורכב מקבוצה של חריצי זיכרון בעלי שם. חלק מחריצי הזיכרון הללו נוצרים כאשר אתה גורר רכיב לתוך האפליקציה שלך; חריצים אלו נקראים מאפיינים. אתה יכול גם להגדיר חריצי זיכרון בעלי שם שאינם משויכים לרכיב מסוים; אלה נקראים משתנים. בעוד שמאפיינים משויכים בדרך כלל למה שנראה באפליקציה, ניתן לחשוב על משתנים כזיכרון ה"שריטה" הנסתר של האפליקציה.

נכסים

המשתמש באפליקציה יכול לראות רכיבים גלויים כגון לחצנים, תיבות טקסט ותוויות.
עם זאת, באופן פנימי, כל רכיב מוגדר לחלוטין על ידי קבוצה של מאפיינים. הערכים המאוחסנים בחריצי הזיכרון של כל מאפיין קובעים כיצד הרכיב
מופיע.
אתה מגדיר את ערכי המאפיינים ישירות .Component Designer-בלדוגמה,
איור 16-1מציג את הפאנל לשינוי המאפיינים של רכיב . Canvas



איור .2-16אתה יכול להגדיר מאפייני רכיב ב-rengiseD; אתה מגדיר את הערכים ההתחלתיים של המאפיינים (הם לא מציגים את הערכים הנוכחיים בזמן שאפליקציה פועלת)

לרכיב Canvasיש מאפיינים רבים מסוגים שונים. למשל, ה
BackgroundColor- וPaintColorהם חריצי זיכרון שמכילים צבע. תמונת הרקע מכילה שם פלפל .(kitty.png)המאפיין Visibleמחזיק בוליאני

ערך true)או ,falseתלוי אם התיבה מסומנת). חריצי הרוחב והגובה מכילים מספר או ייעוד מיוחד (למשל, "מילוי הורה").

כאשר אתה מגדיר מאפיין ,Component Designer-באתה מציין את האותיות הראשוניות ערך הנכס -ערכו כאשר האפליקציה מתחילה. ניתן גם לשנות ערכי נכס תוך כדי הפעלת האפליקציה, עם בלוקים. עם זאת, הערכים המוצגים ,Component Designer-בכגון אלו באיור ,16-1אינם משתנים; אלה תמיד מציגים רק את הערכים ההתחלתיים. זה יכול לבלבל כשאתה בודק אפליקציה -הערך הנוכחי של מאפייני האפליקציה אינו גלוי.

הגדרת משתנים

כמו מאפיינים, משתנים נקראים חריצי זיכרון, אך הם אינם משויכים לרכיב מסוים. אתה מגדיר משתנה כאשר האפליקציה שלך צריכה לזכור משהו שאינו מאוחסן בתוך מאפיין רכיב. לדוגמה, ייתכן שאפליקציית משחק תצטרך לזכור לאיזו רמה המשתמש הגיע. אם מספר הרמה עומד להופיע ברכיב , Labelייתכן שלא תזדקק למשתנה, כי אתה יכול פשוט לאחסן את הרמה במאפיין Textשל רכיב . Label

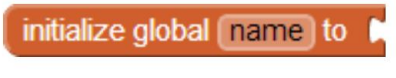
אבל אם מספר הרמה אינו משהו שהמשתמש יראה, תגדיר משתנה שבו לאחסן אותו.

חידון הנשיאים (פרק 8)הוא דוגמה נוספת לאפליקציה שצריכה משתנה.

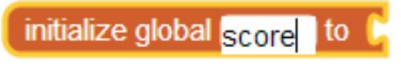
באותה אפליקציה, רק שאלה אחת של החידון צריכה להופיע בכל פעם בממשק המשתמש, ובכל זאת לחידון יש שאלות רבות (שרובן נשמרות מוסתרות מהמשתמש בכל עת). לפיכך, עליך להגדיר משתנה כדי לאחסן את רשימת השאלות.

בעוד שמאפיינים נוצרים באופן אוטומטי בעת גרירת רכיב לתוך המעצב, משתנים מוגדרים במפורש בעורך הבלוקים על ידי גרירת בלוק גלובלי לאתחל . אתה יכול לתת שם למשתנה על ידי לחיצה על הטקסט "שם" בתוך הבלוק, ותוכל לציין ערך ראשוני עבורו על ידי גרירת מספר, טקסט, צבע, או יצירת בלוק רשימה וחיבורו. להלן השלבים שאתה עושה. בצע כדי ליצור משתנה בשם ציון עם ערך התחלתי של :0

1. בבלוקים מובנים, פתח את מגירת המשתנים וגרור החוצה את הבלוק הגלובלי של האתחול .



2. שנה את שם המשתנה על ידי לחיצה על הטקסט "שם" והקלדה "ציון".



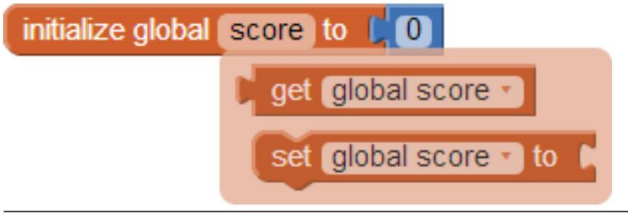
3. ממגירת המתמטיקה, גרור החוצה בלוק מספרים וחבר אותו לחלק הפתוח שקע של ההגדרה המשתנה כדי להגדיר את הערך ההתחלתי.

כאשר אתה מגדיר משתנה, אתה מורה לאפליקציה להגדיר חריץ זיכרון בעל שם לאחסון ערך. חריצי זיכרון אלו, כמו במאפיינים, אינם גלויים למשתמש כאשר האפליקציה פועלת.

בלוק המספרים שאתה מחבר מציין את הערך שיש למקם בחריץ כאשר האפליקציה מתחילה. מלבד אתחול עם מספרים או טקסט, אתה יכול גם לאתחל את המשתנה באמצעות יצירת רשימה או יצירת בלוק רשימה ריק . זה מודיע לאפליקציה שהמשתנה יאחסן רשימה של חריצי זיכרון במקום ערך בודד. למידע נוסף על רשימות, ראה פרק 19.

הגדרה וקבלת משתנה

כאשר אתה מגדיר משתנה, App Inventorיוצר עבורו שני בלוקים: הגדר וקבל . אתה יכול לגשת לבלוקים אלה על ידי ריחוף מעל שם המשתנה בבלוק האתחול, כפי שמוצג באיור 16-2.



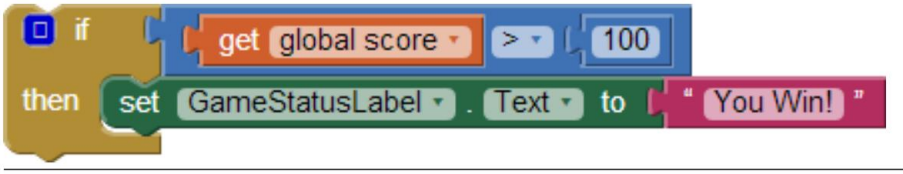
איור 16-3. בלוק האתחול מכיל בלוקים של set and get עבור אותו משתנה

ההגדרה גלובלית לחסימה מאפשרת לך לשנות את הערך המאוחסן במשתנה. לדוגמה, בלוק המספרים באיור 16-3 מציב את הערך 5 בציון המשתנה. המונח "גלובלי" בציון הגלובלי שנקבע לחסימה מתייחס לעובדה שניתן להשתמש במשתנה בכל מטפלי האירועים והנהלים של התוכנית. עם הגרסה החדשה ביותר של App Inventor, אתה יכול גם להגדיר משתנים שהם "מקומיים" להליך מסוים או למטפל באירועים - כלומר, ניתן להשתמש במשתנים מקומיים רק על ידי הפרוצדורה או האירוע שאליהם הם משויכים (עוד על זה קצת מאוחר יותר בפרק).

איור 16-4. הצבת מספר 5 בציון המשתנה

אתה משתמש בבלוק שכותרתו קבל ציון גלובלי כדי לאחזר את הערך של משתנה. לדוגמה, אם אתה רוצה לבדוק אם הערך בתוך חריץ הזיכרון היה גדול מ

,100תחבר את הבלוק לקבל ציון גלובלי למבחן אם , כפי שמוצג באיור .16-4



איור .16-5שימוש בגוש הניקוד הגלובלי כדי לקבל את הערך המאוחסן במשתנה

הגדרת משתנה לביטוי

כפי שראית, אתה יכול להכניס ערכים פשוטים כמו 5למשתנה, אבל לעתים קרובות תגדיר את המשתנה לביטוי מורכב יותר (ביטוי הוא המונח של מדעי המחשב לנוסחה). לדוגמה, כאשר המשתמש לוחץ על Nextכדי להגיע לשאלה הבאה באפליקציית חידון, תצטרך להגדיר את המשתנה currentQuestionלאחד יותר מהערך הנוכחי שלו. כאשר מישהו מאבד עשר נקודות באפליקציית משחק, אתה צריך לשנות את משתנה הניקוד ל-01 פחות מהערך הנוכחי שלו. במשחק כמו MoleMash(פרק ,(3אתה משנה את המיקום האופקי (x)של השומה למיקום אקראי בתוך קנבס.
אתה תבנה ביטויים כאלה עם קבוצה של בלוקים שמתחברים לסט גלובלי לחסום.

הגדלת משתנה

אולי הביטוי הנפוץ ביותר הוא להגדלת משתנה, או קביעת משתנה על סמך הערך הנוכחי שלו. לדוגמה, במשחק, כאשר שחקן קולע נקודה, ניתן להגדיל את הציון המשתנה ב-5. איור 16-5 מציג את הבלוקים ליישום התנהגות זו.

איור .16-6הגדלת הציון המשתנה ב-5

אם אתה יכול להבין סוגים אלה של בלוקים, אתה בדרך להיות א
מְתַכנֵת. אתה קורא את הבלוקים האלה כ"הגדר את הציון ל-evf יותר ממה שהוא כבר", וזו דרך נוספת לומר להגדיל את המשתנה שלך. הדרך שבה זה עובד היא שהבלוקים מתפרשים מבפנים החוצה, לא משמאל לימין. לפיכך, הבלוקים הפנימיים ביותר -הציון הגלובלי של getוהגוש מספר - 5מוערכים תחילה. לאחר מכן, הבלוק +מתבצע והתוצאה "מוגדרת" לציון המשתנה.

נניח שהיה 10בחריץ הזיכרון לציון לפני בלוקים אלה; ה
האפליקציה תבצע את השלבים הבאים:

.1אחזר את ה-01 מחריץ הזיכרון של הציון (הערך את בלוק .(get

.2הוסף לזה 5כדי לקבל .15

.3הכנס את התוצאה, ,15לתוך חריץ הזיכרון של הציון (בביצוע הסט).

בניית ביטויים מורכבים

במגירת Math, App Inventorמספק מגוון רחב של פונקציות מתמטיות הדומות לאלו שתמצא בגיליון אלקטרוני או במחשבון. ישנם אופרטורים אריתמטיים (למשל, ,(/,*,–,+ בלוקים להפקת ערכים אקראיים ואופרטורים כגון sqrt, cosinusוסינוס.

אתה יכול להשתמש בלוקים אלה כדי לבנות ביטוי מורכב ואז לחבר אותם בתור הביטוי בצד ימין של קבוצה גלובלית לחסימה. לדוגמה, כדי להעביר ספרייט תמונה לעמודה אקראית בתוך גבולות בד, תגדיר ביטוי המורכב מבלוק כפל ,(*)בלוק חיסור ,(–)מאפיין Canvas1.Width , מאפיין ImageSprite1.תכונת רוחב , ובלוק שבר אקראי , כפי שמוצג באיור 16-6.

איור .16-7אתה יכול להשתמש בלוקים במתמטיקה כדי לבנות ביטויים מורכבים כמו זה

כמו בדוגמה המצטברת בסעיף הקודם, הבלוקים מתפרשים על ידי האפליקציה בצורה מבפנים החוצה. נניח של- Canvasיש רוחב של 300ול- ImageSpriteיש רוחב של ,50 האפליקציה תבצע את השלבים הבאים:

.1אחזר את ה-003 וה-05 מחריצי הזיכרון עבור Canvas1.Widthו- ,ImageSprite.Width בהתאמה.

.2חיסור: .300 – 50 = 250

.3קרא לפונקציית השבר האקראי כדי לקבל מספר בין 0ל-1 (נניח .(.5

.4הכפל: .250 * .5 = 125

.5הכנס את ה-521 לתוך חריץ הזיכרון עבור המאפיין . ImageSprite1.X

הצגת משתנים

כאשר אתה משנה מאפיין רכיב, כמו בדוגמה הקודמת, ממשק המשתמש מושפע ישירות. זה לא נכון לגבי משתנים; לשינוי משתנה אין השפעה ישירה על מראה האפליקציה. אם רק תגדיל ציון משתנה אבל לא משנה את ממשק המשתמש בדרך אחרת, המשתמש לעולם לא ידע שיש שינוי. זה כמו העץ הפתגם שנפל ביער: אם אף אחד לא היה שם לשמוע את זה, האם זה באמת קרה?

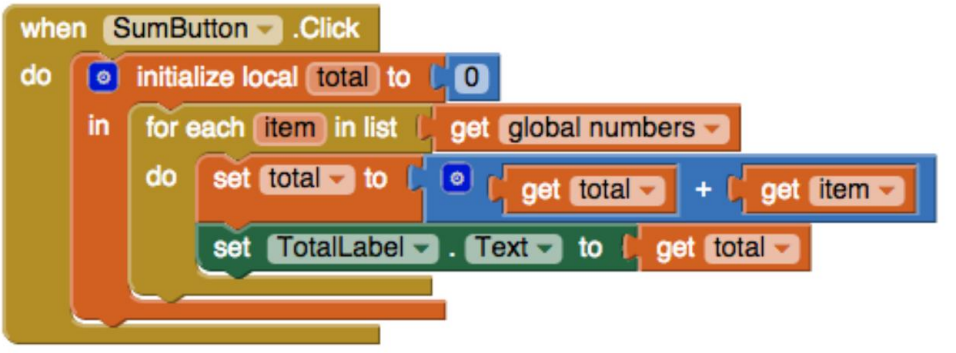
לפעמים, אינך רוצה להפגין מיד שינוי בממשק המשתמש כאשר משתנה משתנה. לדוגמה, במשחק אתה עשוי לעקוב אחר סטטיסטיקות (למשל, זריקות שהוחמצו) שיופיעו רק כשהמשחק יסתיים.
זהו אחד היתרונות של אחסון נתונים במשתנה בניגוד למאפיין רכיב: אתה יכול להציג רק את הנתונים שאתה רוצה כאשר אתה רוצה להציג אותם.
אתה יכול גם להפריד את החלק החישובי של האפליקציה שלך ממשק המשתמש, מה שיקל על שינוי ממשק משתמש זה מאוחר יותר.
לדוגמה, עם משחק, אתה יכול לאחסן את הניקוד ישירות בתווית או במשתנה. אם אתה מאחסן אותו בתווית, אתה תגדיל את מאפיין הטקסט של התווית כאשר נצברו נקודות, והמשתמש יראה את השינוי ישירות. אם אחסנת את הניקוד במשתנה והגדלת את המשתנה כאשר נצברו נקודות, תצטרך לכלול בלוקים כדי להעביר גם את הערך של המשתנה לתווית.

עם זאת, אם החלטתם לשנות את האפליקציה כדי להציג את הציון בצורה שונה, אולי עם סליידר, יהיה קל יותר לשנות את הפתרון המשתנה. לא תצטרך לשנות את כל המקומות שמשנים את הציון; תצטרך רק לשנות את הבלוקים שמציגים את הניקוד.

משתנים מקומיים

המשתנים שתוארו בפרק זה עד כה הם משתנים גלובליים ואתה מגדיר אותם עם אתחול גלובלי לחסימה. ה"גלובלי" מתייחס לעובדה שניתן להשתמש במשתנה בכל מטפלי האירועים והנהלים. אומרים למשתנים כאלה יש היקף גלובלי.

עם הגרסה העדכנית ביותר של App Inventor, כעת תוכל גם להגדיר משתנים מקומיים , כלומר משתנים שהשימוש בהם (ההיקף) מוגבל למטפל או פרוצדורה בודדים של אירועים (ראה איור 16-7).



איור 16-8. המשתנה "סך הכל" הוא מקומי; ניתן להשתמש בו רק באירוע SumButton.Click

אם המשתנה נחוץ רק במקום אחד, כדאי להגדיר אותו כמקומי, כמו
המשתנה "סה"כ" נמצא באיור 16-7. על ידי כך, אתה מגביל את התלות באפליקציה שלך
ומבטיח שלא תשנה משתנה בטעות. תחשוב על משתנה מקומי כמו הזיכרון הפרטי במוח
שלך - אתה בהחלט לא רוצה שלמוחים אחרים תהיה גישה אליו!

סיכום

כאשר אפליקציה מופעלת, היא מתחילה לבצע את פעולותיה ולהגיב לאירועים המתרחשים.
כאשר מגיבים לאירועים, האפליקציה צריכה לפעמים לזכור דברים. עבור משחק, זה עשוי להיות
הניקוד של כל שחקן או הכיוון שאליו חפץ נע.

האפליקציה שלך זוכרת דברים בתוך מאפייני רכיב, אבל כאשר אתה צריך חריצי זיכרון
נוספים שאינם משויכים לרכיב, אתה יכול להגדיר משתנים.
אתה יכול לאחסן ערכים במשתנה ולאחזר את הערך הנוכחי, בדיוק כמו שאתה עושה עם
מאפיינים.
בדומה לערכי מאפיינים, ערכי משתנים אינם גלויים למשתמש הקצה. אם אתה רוצה
שמשתמש הקצה יראה את המידע המאוחסן במשתנה, אתה מוסיף בלוקים שמציגים את
המידע הזה בתווית או ברכיב ממשק משתמש אחר.

יצירת אפליקציות מונפשות

פרק זה דן בשיטות ליצירת אפליקציות עם אנימציות פשוטות (אובייקטים זזים).


איור 17-1

תלמד את היסודות של יצירת משחקים דו מימדיים עם App Inventorותהיה נוח עם ספרייטים של תמונה וטיפול באירועים כגון שני אובייקטים מתנגשים.

כאשר אתה רואה אובייקט נע בצורה חלקה לאורך מסך המחשב, מה שאתה באמת רואה הוא רצף מהיר של תמונות עם האובייקט במקום מעט שונה בכל פעם. זוהי אשליה שאינה שונה ברמה מ-skoobpif, שבהם אתה רואה תמונה נעה על ידי דפדוף מהיר בין הדפים. זה הקונספט מאחורי האופן שבו מיוצרים פלמס מונפש!

עם ,App Inventorתתכנת אנימציה על ידי הצבת רכיבי Ball ו- ImageSpriteבתוך רכיב Canvasולאחר מכן הזזה ושינוי של אובייקטים אלה בכל שבריר שניה עוקב. בפרק זה תלמדו כיצד פועלת מערכת הקואורדינטות של , Canvasכיצד ניתן להשתמש באירוע Clock.Timerכדי להפעיל תנועה, כיצד לשלוט במהירות של אובייקטים וכיצד להגיב לאירועים כגון שני אובייקטים המתרסקים זה בזה. .

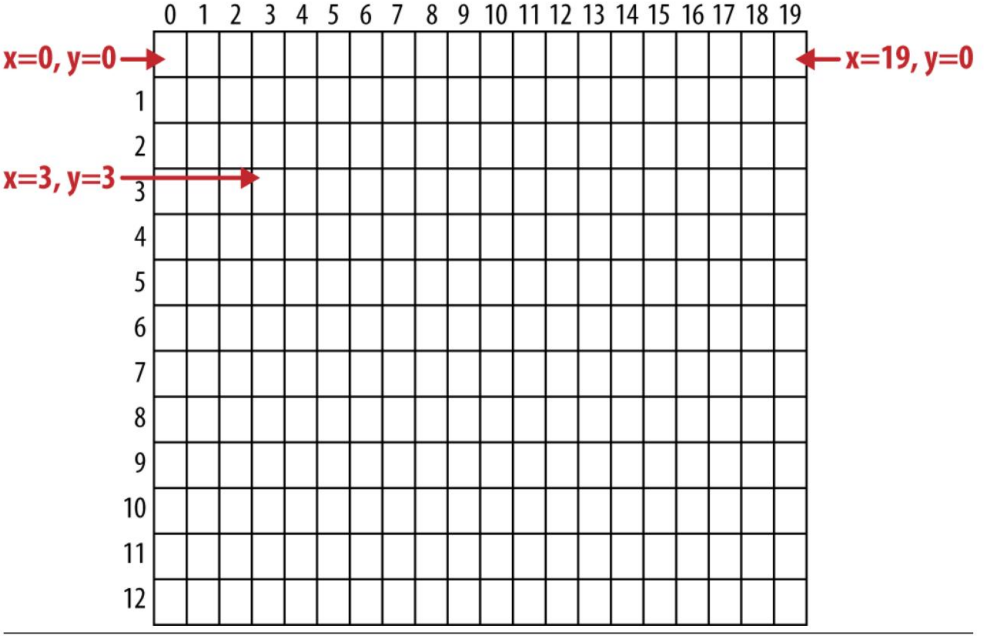
הוספת רכיב קנבס לאפליקציה שלך

מלוח הציור וההנפשה, גרור רכיב Canvasלתוך האפליקציה שלך.
לאחר שתמקם אותו, ציין את הרוחב והגובה שלו . לעתים קרובות, תרצה שה- Canvas יתפרש על פני רוחב מסך המכשיר. כדי לעשות זאת, בחר "מלא הורה" בעת ציון הרוחב.

אתה יכול לעשות את אותו הדבר עבור ה- ,Heightאבל בדרך כלל תגדיר את זה למספר מסוים (למשל, 300פיקסלים) כדי להשאיר מקום לרכיבים אחרים מעל ומתחת לקנבס.

מערכת קואורדינטות הקנבס

ציור על קנבס הוא באמת רשת של פיקסלים, כאשר פיקסל הוא נקודת הצבע הקטנה ביותר שיכולה להופיע על המסך. המיקום של כל פיקסל מוגדר על ידי קואורדינטות xy במערכת רשת, כפי שמוצג באיור 17-1. במערכת הקואורדינטות הזו, x מגדיר מיקום במישור האופקי (מתחיל ב-0 בקצה השמאלי והולך וגדל ככל שעוברים ימינה על פני המסך), ו-y מגדיר מיקום במישור האנכי (החל מ-0 בחלק העליון וגדל ככל שאתה מטה את המסך).

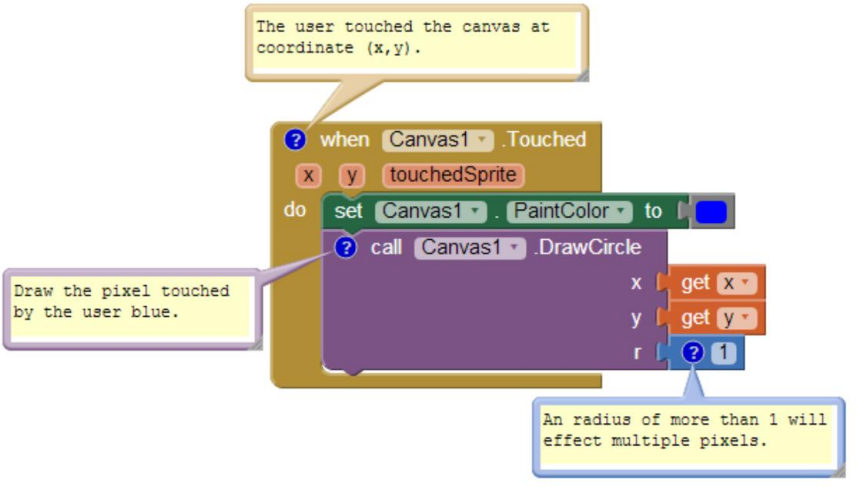


איור 17-2. מערכת הקואורדינטות של קנבס

התא השמאלי העליון ב- Canvas מתחיל ב-0 עבור שתי הקואורדינטות, כך שהמיקום הזה מיוצג (x=0,y=0). ככל שאתה זז ימינה, קואורדינטת ה-x גדלה; ככל שאתה זז למטה, קואורדינטת ה-y גדלה. התא מימין מיד לפינה השמאלית העליונה הוא (x=1,y=0). הפינה הימנית העליונה יש קואורדינטת x השווה לרוחב ה- Canvas מינוס 1. לרוב מסכי הטלפון יש רוחב בסביבות 300 פיקסלים, אבל עבור המדגם המוצג כאן, הרוחב הוא 20, כך שהפינה הימנית העליונה היא הקואורדינטה (x=19,y=0).

ניתן לשנות את מראה הקנבס בשתי דרכים: על ידי ציור עליו, או על ידי הצבה והזזה של אובייקטים בתוכו. פרק זה מתמקד בעיקר באחרון, אך בוא נדון תחילה כיצד אתה "מצייר" וכיצד ליצור אנימציה על ידי ציור (זהו גם הנושא של אפליקציית PaintPot בפרק 2).

כל תא של ה- Canvasמכיל פיקסל המגדיר את הצבע שאמור להופיע שם.
רכיב ה- Canvasמספק את בלוקים Canvas.DrawLine -וCanvas.DrawCircle לצביעת פיקסלים. תחילה עליך להגדיר את המאפיין Canvas.PaintColor לצבע הרצוי ולאחר מכן לקרוא לאחד מבלוקים Drawכדי לצייר בצבע זה. עם DrawCircle, אתה יכול לצבוע עיגולים בכל רדיוס, אבל אם תגדיר את הרדיוס ל-1, כפי שמוצג באיור 17-2, תצבע פיקסל בודד.



איור 17-3. DrawCircle ברדיוס של 1צובע פיקסל בודד בכל נגיעה

App Inventorמספק פלטת צבעים בסיסיים שבהם אתה יכול להשתמש כדי לצבוע פיקסלים (או רכיבי ממשק משתמש אחרים). אתה יכול לגשת למגוון רחב יותר של צבעים באמצעות ערכת מספור הצבעים המוסברת בתיעוד App Inventorבכתובת explore/ appinventor.mit.edu/ support/blocks/colors.html./2iahttp://
מלבד צביעת פיקסלים בודדים, אתה יכול גם למקם רכיבי Ball -וImageSprite על קנבס. ספרייט הוא אובייקט גרפי הממוקם בתוך סצנה גדולה יותר (ב-App Inventor, אתה"סצנה" היא רכיב .(Canvasגם רכיבי הכדור וגם ImageSprite -ההם ספרייטים; הם שונים רק במראה החיצוני. כדור הוא עיגול בעל מראה שניתן לשנות רק על ידי שינוי הצבע או הרדיוס שלו, בעוד ש- ImageSpriteיכול לקבל כל מראה, כפי שמוגדר על ידי דמות תמונה. ניתן להוסיף כדורים ו- ImageSpritesרק בתוך קנבס; אתה לא יכול לגרור אותם לממשק המשתמש מחוץ לממשק המשתמש.

הנפשת אובייקטים עם אירועי טיימר

אחת הדרכים לציין אנימציה ב-ppAInventorהיא לשנות אובייקט בתגובה לאירוע טיימר. לרוב, תעביר ספרייטים למיקומים שונים על הבד במרווחי זמן מוגדרים. שימוש באירועי טיימר היא השיטה הנפוצה ביותר להגדרת מרווחי הזמן המוגדרים. מאוחר יותר, נדון גם בשיטה חלופית לתכנות אנימציה תוך שימוש במאפייני המהירות והכותרת של רכיבי ImageSpriteו- . Ball

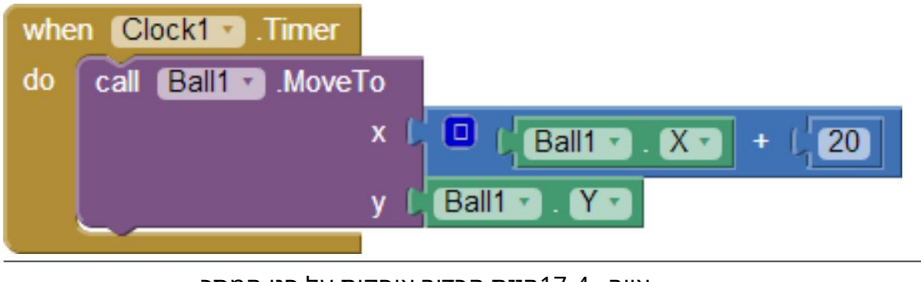
לחיצות על כפתורים ואירועים אחרים ביוזמת המשתמש הם פשוטים להבנה: המשתמש עושה משהו, והאפליקציה מגיבה בביצוע פעולות מסוימות. עם זאת, אירועי טיימר שונים מכיוון שהם אינם מופעלים על ידי משתמש הקצה אלא על ידי הזמן החולף. אתה צריך להמשיג את השעון של הטלפון המפעיל אירועים באפליקציה במקום שמשתמש יעשה משהו.

כדי להגדיר אירוע טיימר, תחילה תגרור רכיב שעון לתוך האפליקציה שלך בתוך מעצב הרכיבים. לרכיב השעון יש מאפיין TimerIntervalהמשויך אליו. המרווח מוגדר באלפיות שניות (1/1,000של שניה). אם תגדיר את TimerIntervalל-005, זה אומר שאירוע טיימר יופעל כל חצי שנייה.

ככל TimerInterval-שקטן יותר, כך קצב הפריימים של האנימציה מהיר יותר.
לאחר הוספת שעון והגדרת TimerIntervalב-rengiseD, תוכל לגרור אירוע Clock.Timerבעורך הבלוקים. אתה יכול לשים כל בלוקים שאתה אוהב באירוע הזה, והם יבוצעו בכל מרווח.

יצירת תנועה

כדי להציג ספרייט שזז לאורך זמן, תשתמש בפונקציה MoveToשנמצאת ברכיבים של ImageSprite ו- . Ballלדוגמה, כדי להזיז כדור אופקית על פני המסך, תשתמש בלוקים כמו אלה באיור .17-3

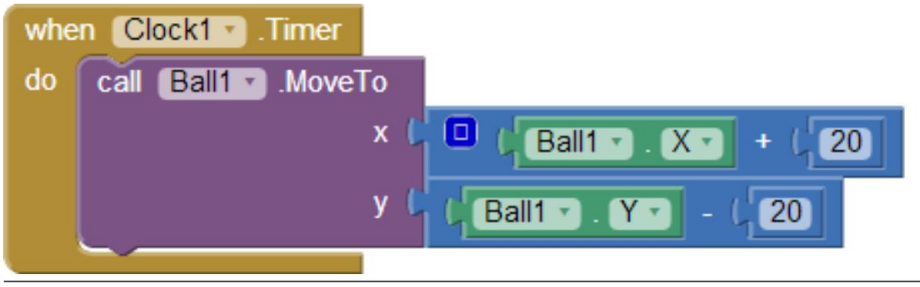


איור .17-4הזזת הכדור אופקית על פני המסך

MoveToמעביר אובייקט למיקום מוחלט על הבד, לא לכמות יחסית. אז כדי להזיז אובייקט מסוים, אתה מגדיר את הארגומנטים MoveToל-

המיקום הנוכחי של האובייקט בתוספת היסט. מכיוון שאנו נעים אופקית, הארגומנט x מוגדר למיקום xהנוכחי (Ball1.X)בתוספת הסט ,20בעוד שהארגומנט yמוגדר להישאר בהגדרה הנוכחית שלו .(Ball1.Y)

כדי להזיז אובייקט למטה, תשנה רק את קואורדינטת Ball1.Yולהשאיר את Ball1.X זהה. אם תרצה להזיז את הכדור באלכסון, תוסיף היסט לקואורדינטות ה-x וה-y, כפי שמוצג באיור .17-4



איור .17-5הקיזת קואורדינטות xו-y כדי להזיז את הכדור באלכסון

מְהִירוּת

כמה מהר הכדור זז בדוגמה הקודמת? המהירות תלויה הן בהגדרות שאתה מספק עבור המאפיין TimerIntervalשל Clock1והן בפרמטרים שאתה מציין בפונקציה MoveTo. אם ה- Clock.TimerIntervalמוגדר ל-000,1 אלפיות השנייה, המשמעות היא שאירוע Clock1.Timerיופעל בכל שנייה. עבור הדוגמה האופקית המוצגת באיור ,17-3הכדור יזוז 20 פיקסלים לשנייה.

אבל TimerIntervalשל 1,000אלפיות השנייה לא מספק אנימציה חלקה במיוחד; הכדור יזוז רק פעם בשנייה, מה שייראה קופצני. כדי להשיג תנועה חלקה יותר, אתה צריך מרווח קצר יותר. אם ה- TimerIntervalהיה מוגדר במקום זאת ל-001 מילישניות, הכדור היה זז 20 פיקסלים כל עשירית שנייה, או 200פיקסלים לשנייה -קצב שייראה חלק הרבה יותר.

זיהוי התנגשות

כדי ליצור משחקים ואפליקציות מונפשות אחרות, אתה צריך פונקציונליות מורכבת יותר מאשר רק תנועה. למרבה המזל, App Inventorמספק כמה בלוקים ברמה גבוהה להתמודדות עם אירועי אנימציה כמו אובייקט שמגיע לקצה המסך או שני אובייקטים מתנגשים.

בהקשר זה, חסימה ברמה גבוהה פירושה ש-ppA Inventorדואג לפרטים ברמה נמוכה יותר של קביעה מתי מתרחש אירוע כגון התנגשות. אתה יכול לבדוק את ההתרחשויות האלה באופן ידני על ידי בדיקת מאפייני ספרייט וקנבס בתוך אירועי Clock.Timer. רמה זו של תכנות דורשת היגיון מורכב למדי,

למרות זאת. למרבה המזל, App Inventorמספק אותם עבורכם, וטוב שכך כי האירועים הללו משותפים להרבה משחקים ואפליקציות אחרות.

EdgeReached

שקול שוב את האנימציה באיור 17-4, שבה האובייקט נע באלכסון מהצד השמאלי העליון לימין התחתון של הבד. כפי שתוכנת, הכדור יזוז באלכסון ואז נעצר כשהגיע לקצה הימני או התחתון של הבד (המערכת לא תזיז אובייקט מעבר לגבולות הבד).

אם במקום זאת רצית שהאובייקט יופיע שוב בפינה השמאלית העליונה בכל פעם שהוא הגיע לקצה התחתון או הימני, תוכל להגדיר תגובה לאירוע Ball.EdgeReachedבדומה לזה שמוצג באיור 17-5.

איור 17-6. גורם לכדור להופיע שוב בפינה השמאלית העליונה כאשר הוא מגיע לקצה

EdgeReachedמופעל כאשר כדור או ספרייט תמונה פוגעים בכל קצה של קנבס. מטפל באירועים זה, בשילוב עם התנועה האלכסונית שצוינה עם אירוע הטיימר שתואר קודם לכן (איור 17-4), גורם לכדור לנוע באלכסון משמאל למעלה לימין תחתון, לקפוץ בחזרה לשמאל העליון כשהוא מגיע לקצה, ו ואז לעשות הכל שוב, לנצח (או עד שהאפליקציה תגיד לה אחרת).

בדוגמה זו, לא הבחנו בין איזה קצה נפגע. לאירוע EdgeReachedיש פרמטר, edge, המציין את הקצה המסוים באמצעות הקוד הבא:

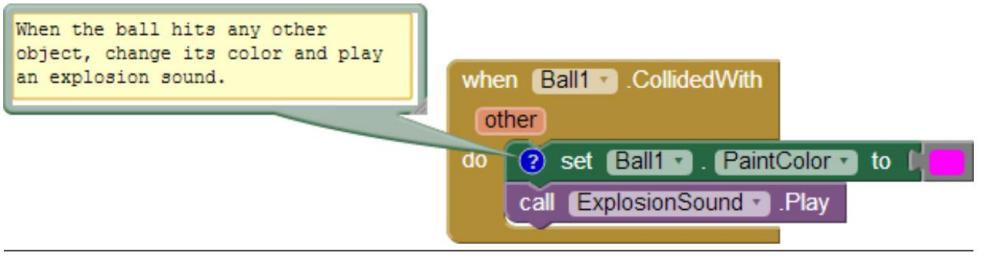
- צפון 1 =
- צפון מזרח 2 =
- מזרח 3 =
- דרום מזרח 4 =
- דרום 1– =
- דרום מערב 2– =

•מערב 3– =

•צפון מערב 4– =

מתנגש עם ו-oN LongerCollidingWith

משחקים ואפליקציות מונפשות אחרות מסתמכות לרוב על פעילות המתרחשת כאשר שני עצמים או יותר מתנגשים (למשל, מחבט שפוגע בכדור).
חשבו על משחק, למשל, שבו אובייקט משנה צבעים ומשמיע צליל פיצוץ כשהוא פוגע באובייקט אחר. איור 17-6מציג את הבלוקים עבור מטפל באירוע כזה.

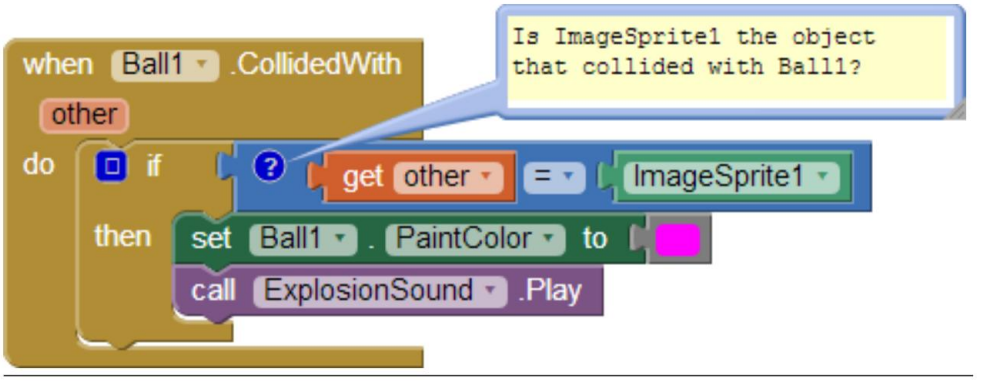


איור .17-7לגרום לכדור לשנות צבע ולהשמיע צליל פיצוץ כשהוא פוגע בחפץ אחר

NoLongerCollidingWithמספק את האירוע ההפוך מ- .CollidedWithזה מופעל רק כאשר שני עצמים התאחדו ואז נפרדו. אז, עבור המשחק שלך, אתה עשוי לכלול את הבלוקים המתוארים באיור .17-7

איור .17-8שינוי הצבע בחזרה ועצירת רעש הפיצוץ כאשר החפצים נפרדים

שים לב שגם CollidedWith-לוגם ל- NoLongerCollidingWithיש ארגומנט, אחר, המציין את האובייקט המסוים איתו התנגשת (או ממנו נפרדת). זה מאפשר לך לבצע פעולות רק כאשר האובייקט (למשל, Ball1)מקיים אינטראקציה עם אובייקט אחר מסוים, כפי שמוצג באיור .17-8

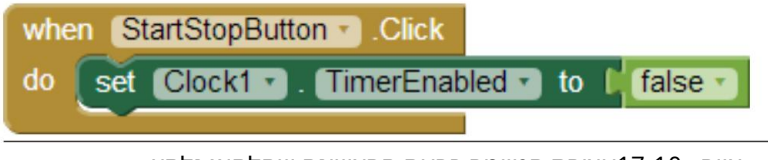


איור 17-9. בצע את התגובה רק אם Ball1פגע ב-ImageSprite1

בלוק ImageSprite1הוא אחד שעדיין לא דנו בו. בלוק זה מתייחס לרכיב בכללותו, לא למאפיין מסוים של הרכיב. כאשר אתה צריך להשוות רכיבים (למשל, כדי לדעת אילו מהם התנגשו), אתה משתמש בבלוק זה.
לכל רכיב יש בלוק כזה במגירה שלו, ולגוש יש שם זהה לזה של הרכיב.

אנימציה אינטראקטיבית

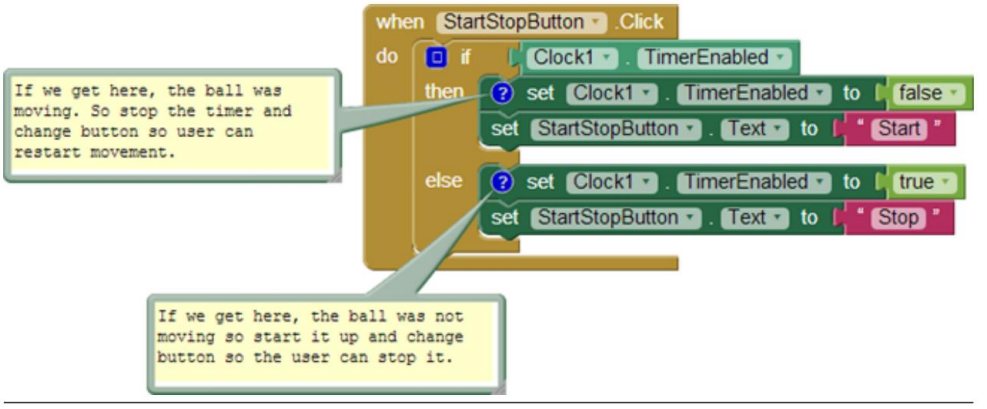
בהתנהגויות המונפשות שדנו בהן עד כה, משתמש הקצה אינו מעורב. משחקים הם אינטראקטיביים, כמובן, כאשר משתמש הקצה משחק תפקיד מרכזי. לעתים קרובות, משתמש הקצה שולט במהירות או בכיוון של אובייקט באמצעות לחצנים או אובייקטים אחרים של ממשק משתמש.
כדוגמה, בואו נעדכן את האנימציה האלכסונית על ידי מתן אפשרות למשתמש לעצור ולהתחיל את התנועה האלכסונית. אתה יכול לעשות זאת על ידי תכנות מטפל באירועים Button.Clickכדי להשבית ולהפעיל מחדש את אירוע הטיימר של השעון רְכִיב.
כברירת מחדל, המאפיין timerEnabledשל רכיב השעון מסומן. אתה יכול להשבית אותו באופן דינמי על ידי הגדרתו ל-eslaf במטפל באירועים. המטפל באירועים באיור 17-9, למשל, יפסיק את הפעילות של טיימר שעון בלחיצה הראשונה.



איור 17-10. עצירת הטיימר בפעם הראשונה שהלחצן נלחץ

לאחר שהמאפיין Clock1.TimerEnabledמוגדר כ-eslaf, אירוע ה- Clock1.Timerלא יופעל יותר, והכדור יפסיק לנוע.

כמובן, עצירת התנועה בלחיצה הראשונה אינה מעניינת מדי. במקום זאת,
אתה יכול "להחליף" את תנועת הכדור על ידי הוספת 'אם אחר 'במטפל באירוע שמאפשר או משבית את הטיימר, כפי שמוצג באיור .17-10
מטפל באירועים זה עוצר את הטיימר בלחיצה הראשונה ומאפס את הכפתור כך שיציג "התחל" במקום "עצור". בפעם השנייה שהמשתמש לוחץ על הכפתור, ה- TimerEnabled הוא שקר, כך שהחלק "אחר" מבוצע. במקרה זה, הטיימר מופעל, מה שמניע את האובייקט שוב, וטקסט הלחצן מועבר בחזרה ל"עצור".
למידע נוסף על בלוקים של ifelse , ראה פרק 18,ולדוגמאות של אנימציות אינטראקטיביות המשתמשות בחיישן הכיוון, ראה פרק 5ופרק 23.

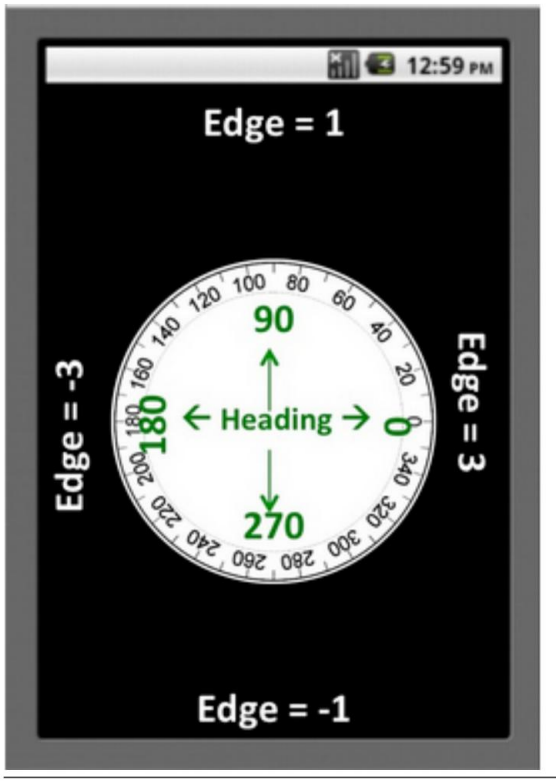


איור .17-11הוספת אם אחרת כדי שהלחיצה על הכפתור תתחיל ותעצור את תנועת הכדור

ציון אנימציית ספרייט ללא טיימר שעון

דגימות האנימציה שתוארו עד כה משתמשות ברכיב Clockומציינות שאובייקט צריך לזוז בכל פעם שאירוע ה- Clock.Timerמופעל. סכימת האירועים Clock.Timerהיא השיטה הכללית ביותר לציון הנפשה. מעבר להזזת אובייקט פשוט, תוכל גם לשנות את צבעו של אובייקט לאורך זמן, לשנות טקסט כלשהו (שיראה כאילו האפליקציה מקלידה), או שהאפליקציה תדבר מילים בקצב מסוים.

אם אתה רוצה רק להזיז אובייקטים, App Inventorמספק חלופה שלא דורשים שימוש ברכיב שעון . כפי שאולי שמתם לב, לרכיבי ImageSpriteו- Ballיש מאפיינים עבור Heading, Speedו- .Intervalבמקום להגדיר מטפל באירועים , Clock.Timerאתה יכול להגדיר את המאפיינים האלה Component Designer-באו Blocks Editorכדי לשלוט כיצד ספרייט זז.
לשם המחשה, הבה נשקול מחדש את הדוגמה שהניעה כדור באלכסון. הכותרת _ לתכונה של ספרייט או כדור יש טווח של 360מעלות, כפי שמוצג באיור .17-11

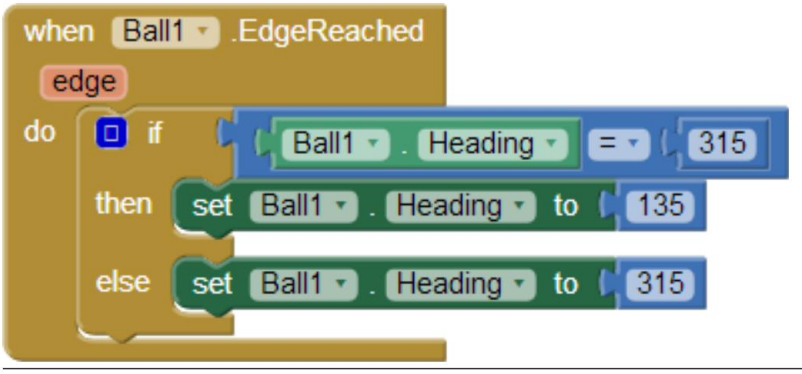


איור 17-12. למאפיין Heading יש טווח של 360 מעלות

אם תגדיר את הכותרת ל ,0- הכדור יזוז משמאל לימין. אם תגדיר אותו ל-09, הוא יזוז מלמטה למעלה. אם תגדיר אותו ל-081, הוא יזוז מימין לשמאל. אם תגדיר אותו ל-072, הוא יעבור מלמעלה למטה. ואם תגדיר את זה ל-513, הכדור ינוע משמאל למעלה לימין למטה.

כדי לגרום לאובייקט לזוז, עליך גם להגדיר את המאפיין Speed לערך שאינו 0. המהירות שהאובייקט מזיז נקבעת למעשה על ידי המאפיינים Speed ו- Interval ביחד. המאפיין Speed הוא המרחק, בפיקסלים, שהאובייקט יזוז בכל מרווח.

כדי לנסות את המאפיינים האלה, צור אפליקציית בדיקה עם קנבס וכדור והתחבר המכשיר או האמולטור שלך לבדיקה חיה. לאחר מכן, שנה את מאפייני הכיוון , המהירות והמרווח של הכדור כדי לראות כיצד הם פועלים.
לדוגמה, נניח שרצית להזיז כדור קדימה ואחורה מהצד השמאלי העליון לימין התחתון של הבד. במעצב, תוכל לאתחל את המהירות של הכדור ל-5 ו- Interval ל-001, ולאחר מכן להגדיר את המאפיין Heading ל-513. לאחר מכן תוסיף את המטפל באירועים Ball1.EdgeReached , אותו תוכל לראות באיור 17-12, כדי לשנות כיוון הכדור כשהוא מגיע לשני הקצוות.



איור 17-13. שינוי כיוון הכדור כאשר הוא מגיע לשני הקצוות

סיכום

באמצעות רכיב ה- Canvas , ניתן להגדיר תת-אזור במסך המכשיר בו אובייקטים יכולים לנוע ולקיים אינטראקציה. אתה יכול לשים רק שני סוגים של רכיבים בתוך קנבס: ImageSprites- ו Balls.

אנימציה היא אובייקט שזז או משתנה לאורך זמן. אתה יכול לתכנת אנימציה, כולל תנועה ותמורות גרפיות אחרות, עם אירוע הטיימר של רכיב השעון . אם אתה רק רוצה להזיז אובייקטים, אתה יכול להשתמש בשיטה חלופית המבוססת על מאפייני Heading, Speed ו- Interval הפנימיים של רכיבי ImageSprite- ו Ball .

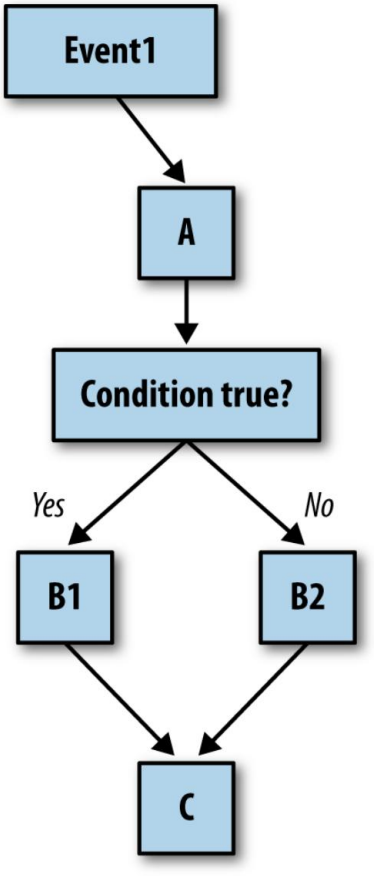
עם כל אחת מהשיטות, אתה יכול גם לנצל את הפונקציונליות ברמה גבוהה עבור טיפול באירועים המטפלים בהתנגשויות.

תכנות האפליקציה שלך לקבלת החלטות: בלוקים מותנים

איור 18-1

מחשבים, אפילו קטנים כמו הטלפון בכיס, טובים בביצוע מיליוני פעולות בשנייה אחת. אפילו יותר מרשים, הם יכולים גם לקבל החלטות על סמך הנתונים במאגרי הזיכרון שלהם והלוגיקה שצוינו על ידי המתכנת. יכולת קבלת ההחלטות הזו היא כנראה המרכיב העיקרי של מה שאנשים חושבים עליו כעל בינה מלאכותית, וזה ללא ספק חלק חשוב מאוד ביצירת אפליקציות חכמות ומעניינות! בפרק זה, נחקור כיצד לבנות את היגיון קבלת ההחלטות הזה לתוך

האפליקציות שלך.



פרק 14דן בהתנהגות של אפליקציה מוגן על ידי קבוצה של מטפלי אירועים. כל מטפל באירועים מבצע פונקציות ספציפיות בתגובה לאירוע מסוים. התגובה לא חייבת להיות רצף ליניארי של פונקציות, עם זאת; אתה יכול לציין שחלק מהפונקציות יבוצעו רק בתנאים מסוימים. לדוגמה, אפליקציית משחק עשויה לבדוק אם הניקוד של שחקן הגיע ל-001, או אפליקציה שמודעת למיקום עשויה לשאול אם הטלפון נמצא בגבולות בניין כלשהו. האפליקציה שלך יכולה לשאול שאלות כאלה, ובהתאם לתשובה, להמשיך בהתאם.

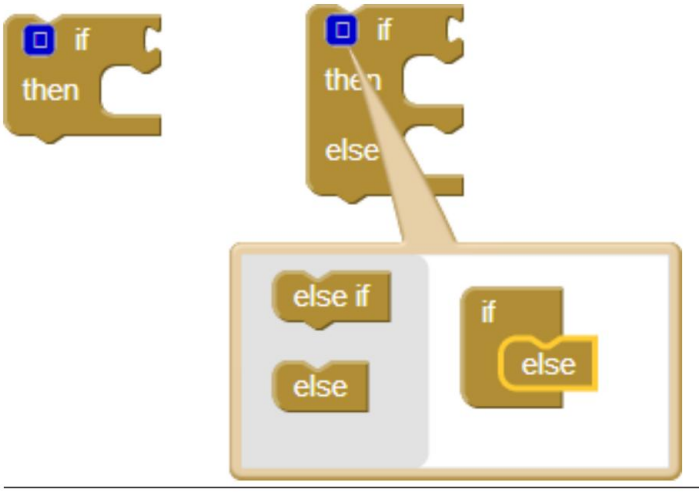
שקול את הדיאגרמה באיור .18-1כאשר מתרחש אירוע, פונקציה (בלוק) Aמבוצעת. לאחר מכן, מתבצעת מבחן החלטה. אם הבדיקה נכונה, מתבצעת .B1אם הוא שקר, B2מבוצע. בכל מקרה, שאר המטפל באירועים (C)הושלם.

מכיוון שדיאגרמות החלטה של אפליקציה כמו זו נראות כמו עצים, אנו אומרים שהאפליקציה "מסעפת" כך או אחרת בהתאם לתוצאת הבדיקה. אז, במקרה זה, היית אומר, "אם הבדיקה נכונה, הענף המכיל B1 מבוצע."

איור .18-2מטפל באירועים שבודק מצב ומסתעף בהתאם

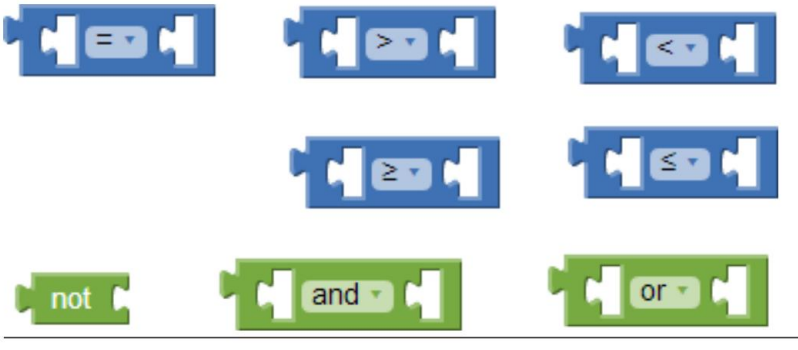
תנאי בדיקה עם esle-iif If Blocks

כדי לאפשר הסתעפות מותנית, App Inventorמספק בלוק מותנה אם-אז במגירת הבקרה. אתה יכול להרחיב את הבלוק עם כמה אחרים ואחרים אם תסתעף כפי שתרצה על ידי לחיצה על הסמל הכחול, כפי שמוצג באיור .18-2



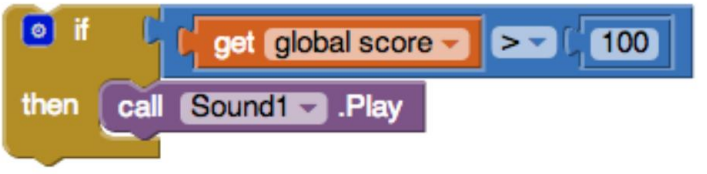
איור .18-3חסימות אם ואחרות אם

אתה יכול לחבר כל ביטוי בוליאני לשקעי הבדיקה של בלוקי if- . else ifביטוי בוליאני הוא משוואה מתמטית שמחזירה תוצאה של נכון או לא נכון. הביטוי בודק את הערך של מאפיינים ומשתנים באמצעות אופרטורים רלציוניים ולוגיים כמו אלו המוצגים באיור .18-3



איור .18-4בלוקים של אופרטורים יחסיים ולוגיים המשמשים במבחנים מותנים

הבלוקים שתשים בתוך שקע ה"אז" של בלוק ifיבוצעו רק אם
המבחן נכון. אם הבדיקה שגויה, האפליקציה עוברת לבלוקים הבאים.
עבור משחק, ייתכן שתחבר ביטוי בוליאני לבדיקת הניקוד של שחקן, כפי שמוצג באיור
.18-4

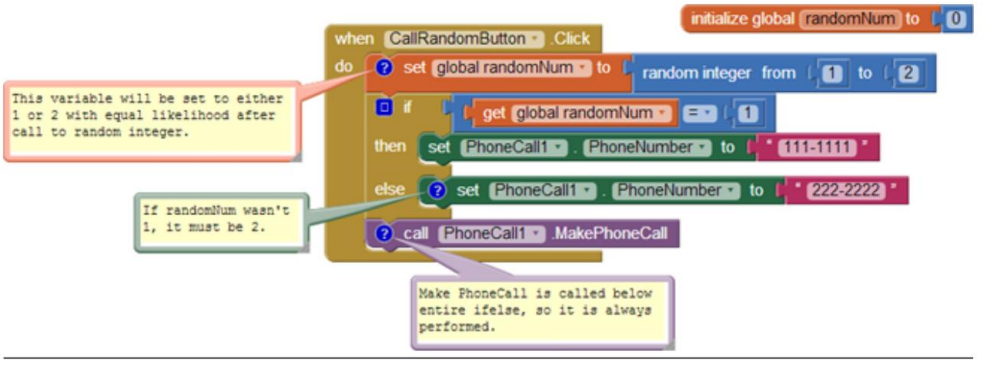


איור 18-5. ביטוי בוליאני המשמש לבדיקת הערך של הציון המשתנה

בדוגמה זו, מושמע פלי של קול אם הציון גדול מ-001. בדוגמה זו, אם המבחן הוא שקר, הצליל לא מושמע והאפליקציה קופצת מתחת לכל הבלוק אם-אז ועוברת לשלב הבא. לחסום באפליקציה שלך. אם אתה רוצה שבדיקת שקר תפעיל פעולה, אתה יכול להשתמש ב- else or else if block.

תכנות החלטת או/או

שקול אפליקציה שתוכל להשתמש בה כאשר אתה משועמם: אתה לוחץ על כפתור בטלפון שלך, והיא מתקשרת לחבר אקראי. באיור 18-5, בלוק אקראי של מספר שלם משמש ליצירת מספר אקראי ולאחר מכן בלוק if else מתקשר למספר טלפון מסוים על סמך אותו מספר אקראי.



איור 18-6. זה אחר אם בלוק קורא לאחד משני מספרים על סמך המספר השלם שנוצר באקראי

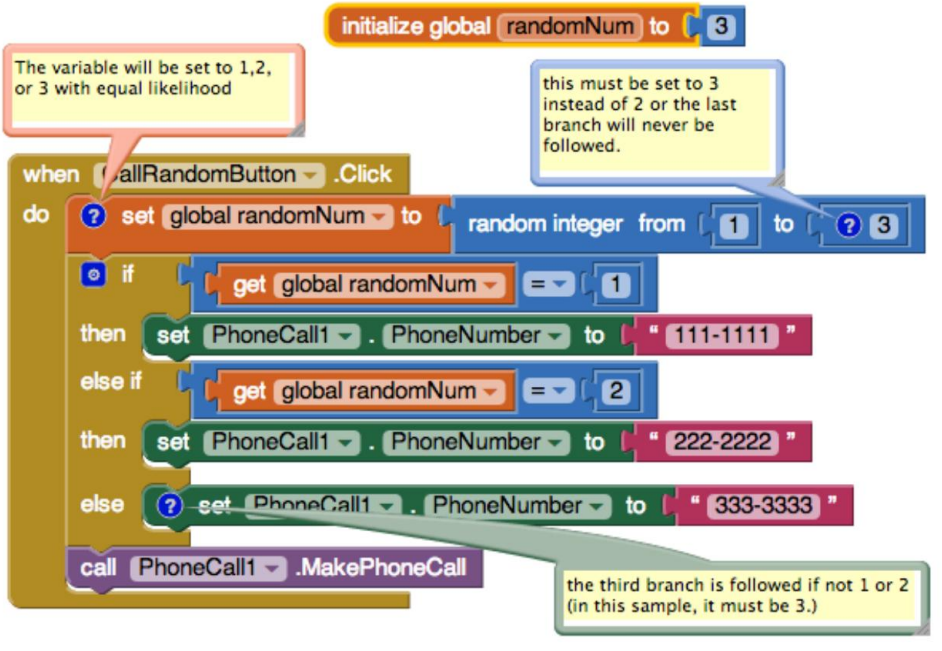
בדוגמה זו, מספר שלם אקראי נקרא עם ארגומנטים 1 ו-2, כלומר המספר האקראי המוחזר יהיה 1 או 2 בסבירות שווה. המשתנה RandomNum מאחסן את המספר האקראי המוחזר.

לאחר הגדרת RandomNum, הבלוקים משווים אותו למספר 1 במבחן . if אם הערך של RandomNum הוא 1, האפליקציה לוקחת את הסניף הראשון (ואז), ומספר הטלפון מוגדר ל-111-1111. אם הערך אינו 1, הבדיקה היא שקר, ובמקרה זה האפליקציה לוקחת את הסניף השני (אחר), ומספר הטלפון מוגדר ל-222– .2222 האפליקציה עושה

שיחת הטלפון בכל מקרה מכיוון שהשיחה ל- MakePhoneCallנמצאת מתחת לכל החסימה אם אחרת .

תנאי תכנות בתוך תנאים

למצבי החלטה רבים יש יותר מסתם שתי תוצאות לבחירה.
לדוגמה, ייתכן שתרצה לבחור בין יותר משני חברים בתוכנית השיחות האקראיות שלך. כדי לעשות זאת, אתה יכול למקם elseאם לפני הענף המקורי של , elseכפי שמוצג באיור 18-6.



איור 18-7. if, else if ועוד מספקים שלושה ענפים אפשריים

עם בלוקים אלה, אם הבדיקה הראשונה נכונה, האפליקציה מבצעת את הסניף הראשון לאחר מכן ומתקשרת למספר 111-1111. אם הבדיקה הראשונה היא שקר, ה- else ifמבוצע, אשר מפעיל מיד בדיקה נוספת. לכן, אם המבחן הראשון (RandomNum=1)הוא שקר והשני (RandomNum=2)נכון, הענף השני מבוצע ונקרא 222–2222. אם שתי הבדיקות שגויות, אחרת מתבצעת הסתעפות בתחתית והמספר השלישי (333–3333)נקרא.

שימו לב שהשינוי הזה עובד רק בגלל שהפרמטר toשל הקריאה האקראית של מספר שלם שונה ל-3 כך ש-1, 2או 3נוצרים בסבירות שווה.

אתה יכול להוסיף כמה סניפים אחרים אם תרצה. אתה יכול גם לקנן תנאים בתוך תנאים. כאשר מבחנים מותנים ממוקמים בתוך ענפים של מבחן מותנה אחר, אנו אומרים שהם מקוננים. אתה יכול לקנן תנאי תנאי ומבני בקרה אחרים כגון עבור כל לולאה לרמות שרירותיות על מנת להוסיף מורכבות

אפליקציה.

תכנות תנאים מורכבים

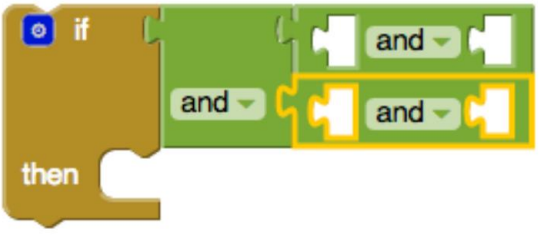
מלבד תנאי קינון, ניתן גם לציין בדיקות מותניות בודדות שהן מורכבות יותר מבדיקת שוויון פשוטה. לדוגמה, שקול אפליקציה שרוטטת כאשר הטלפון שלך (וכנראה אתה!) עוזב בניין או גבול כלשהו. אפליקציה כזו עשויה לשמש אדם על תנאי כדי להזהיר אותו כאשר הוא מתרחק מדי מגבולותיו החוקיים. הורים עשויים להשתמש בו כדי לעקוב אחר מקום הימצאו של ילדיהם. מורה עשויה להשתמש בו כדי לקחת את הגלגול באופן אוטומטי (אם לכל התלמידים שלה יש טלפון אנדרואיד!).

עבור הדוגמה הזו, בואו נשאל את השאלה הזו: האם הטלפון נמצא בגבולות מרכז הרני למדע באוניברסיטת סן פרנסיסקו? אפליקציה כזו תדרוש מבחן מורכב המורכב מארבע שאלות שונות:

- האם קו הרוחב של הטלפון קטן מקו הרוחב המרבי (37.78034)של גְבוּל?
- האם קו האורך של הטלפון קטן מקו האורך המרבי (122.45027–)של גְבוּל?
- האם קו הרוחב של הטלפון גדול מקו הרוחב המינימלי (37.78016)של גְבוּל?
- האם קו האורך של הטלפון גדול מקו האורך המינימלי (122.45059–)של גְבוּל?

אתה צריך את הרכיב LocationSensorעבור דוגמה זו. אתה אמור להיות מסוגל לעקוב כאן גם אם לא נחשפת ל- ,LocationSensorאבל תוכל ללמוד עוד על כך בפרק 23.

אתה יכול לבנות בדיקות מורכבות על ידי שימוש באופרטורים הלוגיים , או, ולא , אותם תוכל למצוא במגירת הלוגיקה. במקרה זה, אתה גורר החוצה בלוק ifוחלק ו- בלוקים, מקם את אחד מהבלוקים ו בתוך שקע ה"בדיקה" של ה- ,ifואת האחרים בתוך ה-tsrf ובלוק , כפי שמוצג באיור .7-18



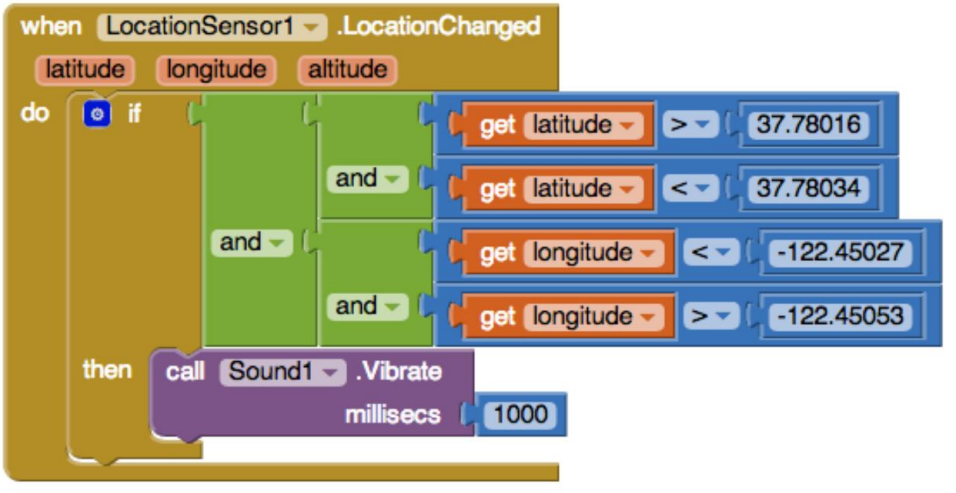
איור 18-8. מבחן אם יכול לבדוק תנאים רבים באמצעות ו, או, ובלוקים יחסיים אחרים

לאחר מכן, תגרור את הבלוקים עבור השאלה הראשונה ותכניס אותם לשקע ה"בדיקה" של הבלוק הראשון, כפי שמוצג באיור 18-8.

איור 18-9. בלוקים עבור הבדיקה הראשונה ממוקמים ב- and block

לאחר מכן תוכל למלא את השקעים האחרים עם הבדיקות האחרות ולמקם את כולו אם בתוך אירוע . LocationSensor.LocationChangedכעת יש לך מטפל באירועים שבודק את הגבול, כפי שמוצג באיור 18-9.

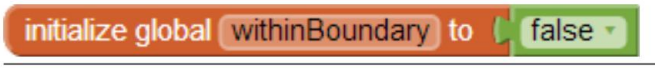
תכנות תנאים מורכבים



איור .18-10מטפל באירועים זה בודק את הגבול בכל פעם שהמיקום משתנה

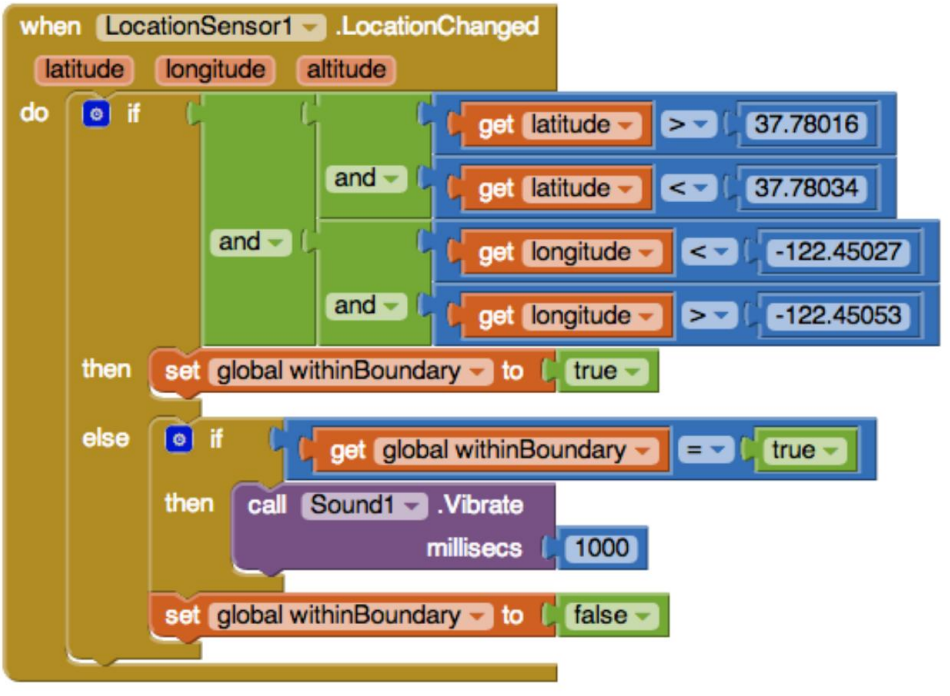
עם בלוקים אלה, בכל פעם שה- LocationSensorמקבל קריאה חדשה ושלה המיקום נמצא בתוך הגבול, הטלפון רוטט.
בסדר, עד כה זה די מגניב, אבל עכשיו בואו ננסה משהו אפילו יותר מסובך לתת לך מושג לגבי היקף סמכויות קבלת ההחלטות של האפליקציה. מה אם היית רוצה שהטלפון ירטוט רק כשהגבול נחצה מבפנים אל חוץ? לפני שתתקדם, חשב כיצד תוכל לתכנת מצב כזה.

הפתרון שלנו הוא להגדיר משתנה בתוך גבול שזוכר אם קריאת החיישן הקודמת הייתה בתוך הגבול או מחוצה לו, ולאחר מכן משווה את זה לכל קריאת חיישן עוקבת. insideBoundaryהוא דוגמה למשתנה בוליאני -במקום לאחסן מספר או טקסט, הוא מאחסן אמת או שקר. עבור דוגמה זו, היית אתחול אותו ל-eslaf, כפי שמוצג באיור ,18-10כלומר, המכשיר אינו נמצא ב-yenraH Science Centerשל USF.



איור insideBoundary .18-11מאותחל ל-eslaf

כעת ניתן לשנות את הבלוקים כך שהמשתנה insideBoundaryמוגדר על כל אחד מהם שינוי מיקום, וכדי שהטלפון רוטט רק כשהוא זז מבפנים אל מחוץ לגבול. אם לנסח זאת במונחים שאנו יכולים להשתמש עבור בלוקים, הטלפון צריך לרטוט כאשר (1המשתנה בתוך הגבול נכון, כלומר הקריאה הקודמת הייתה בתוך הגבול, ו-2) קריאת חיישן המיקום החדש נמצאת מחוץ לגבול. איור 18-11מציג את הבלוקים המעודכנים.



איור 18-12. בלוקים אלו גורמים לטלפון לרטוט רק כאשר הוא נע מתוך הגבול אל מחוץ לגבול

הבה נבחן את הבלוקים הללו יותר מקרוב. כאשר חיישן המיקום מקבל קריאה, LocationSensor מגדיר הוא בודק תחילה אם הקריאה החדשה נמצאת בתוך הגבול. אם כן, את המשתנה insideBoundary ל-true. מכיוון שאנו רוצים שהטלפון ירטוט רק כשאנחנו מחוץ לגבול, לא מתרחשת רטט בסניף הראשון הזה.

אם נגיע אל האחר, אנו יודעים שהקריאה החדשה נמצאת מחוץ לגבול. בשלב זה, אנחנו צריכים לבדוק את הקריאה הקודמת: אם אנחנו מחוץ לגבול, אנחנו רוצים שהטלפון ירטוט רק אם הקריאה הקודמת הייתה בתוך הגבול. insideBoundary נותן לנו את הקריאה הקודמת, כדי שנוכל לבדוק זאת. אם זה נכון, אנחנו רוטטים את הטלפון.

יש עוד דבר אחד שאנחנו צריכים לעשות אחרי שנאשר שהטלפון כן עבר מבפנים אל מחוץ לגבול -האם אתה יכול לחשוב מה זה? אנחנו גם צריכים לאפס בתוך insideBoundary ל-false כדי שהטלפון לא ירטוט שוב בקריאה הבאה של החיישן.

הערה אחרונה לגבי משתנים בוליאניים: בדוק את שני הבדיקות אם באיור 18-12. הם שוות ערך?

תכנות תנאים מורכבים

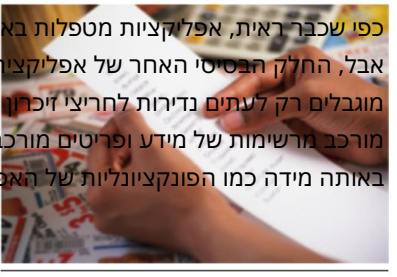
איור 18-13. האם אתה יכול לדעת אם שני אלה אם המבחנים שווים?

התשובה היא כן!" ההבדל היחיד הוא שהמבחן מימין הוא למעשה דרך מתוחכמת יותר לשאול את השאלה. המבחן משמאל משווה את הערך של משתנה בוליאני עם true. אם בתוך הגבול מכיל true, אתה משווה נכון ל- true, וזה נכון. אם המשתנה מכיל false, אתה משווה את false ל- true, שהוא false. עם זאת, רק בדיקת הערך של insideBoundary, כמו בבדיקה מימין, נותנת את אותה תוצאה וקל יותר לקוד.

סיכום

הראש שלך מסתובב? ההתנהגות האחרונה הייתה די מורכבת! אבל, זה סוג קבלת ההחלטות שאפליקציות מתוחכמות צריכות לבצע. אם תבנה התנהגויות כאלה חלק אחר חלק (או ענף אחר ענף) ותבדוק תוך כדי, תגלה שציינת היגיון מורכב - אפילו, נעז לומר, בינה מלאכותית - היא ניתנת לביצוע. זה יגרום לראש שלך לכאב ולהפעיל לא מעט את הצד ההגיוני של המוח שלך, אבל זה גם יכול להיות כיף גדול.

תכנות רשימות נתונים



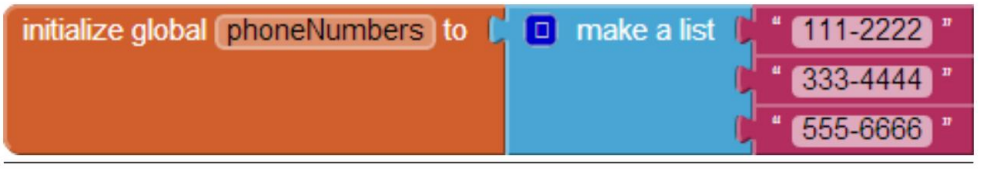
כפי שכבר ראית, אפליקציות מטפלות באירועים ומקבלות החלטות; עיבוד כזה הוא בסיסי למחשוב. אבל, החלק הבסיסי האחר של אפליקציה הוא הנתונים שלה -המידע שהיא מעבדת. נתוני אפליקציה מוגבלים רק לעתים נדירות לחריצי זיכרון בודדים, כגון הניקוד של משחק. לעתים קרובות יותר, הוא מורכב מרשימות של מידע ופריטים מורכבים הקשורים זה בזה, שחייבים להיות מאורגנים בקפידה באותה מידה כמו הפונקציונליות של האפליקציה.

איור 19-1.

בפרק זה, נבחן את הדרך שבה App Inventor מטפל בנתונים. תלמד את היסודות של תכנות הן מידע סטטי, שבו הנתונים אינם משתנים, והן מידע דינמי, שבו הנתונים מוזנים על ידי משתמש הקצה. תלמד כיצד לעבוד עם רשימות, ולאחר מכן תחקור מבנה נתונים מורכב יותר הכולל רשימות של רשימות ואפליקציית חידון רב-ברירה.

אפליקציות רבות מעבדות רשימות של נתונים. לדוגמה, פייסבוק מעבדת את הרשימה שלך של חברים ורשימות של דוחות סטטוס. אפליקציית חידון פועלת עם רשימה של שאלות ותשובות. למשחק עשוי להיות רשימה של דמויות או ציונים שיא בכל הזמנים.

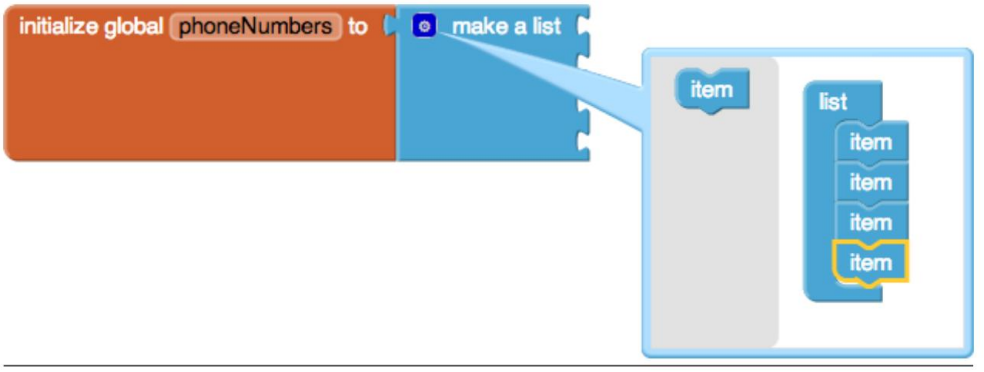
אתה מציין נתוני רשימה ב-App Inventor עם משתנה, אבל במקום לתת שם ליחיד תא זיכרון עם המשתנה, אתה שם לקבוצה של תאי זיכרון קשורים. אתה מציין שמשתנה הוא מרובה פריטים על ידי שימוש ב- make a list או יצירת בלוקי רשימה ריקים . לדוגמה, ה-phoneNumbers המשתנים באיור 19-1 מגדיר רשימה של שלושה פריטים.



איור 19-2. phoneNumbers נותנת שמות לשלושה תאי זיכרון המאוחלים עם המספרים המוצגים

יצירת משתנה רשימה

אתה יוצר משתנה רשימה בעורך הבלוקים על ידי שימוש בבלוק משתנה גלובלי של אתחול ולאחר מכן חיבור של בלוק עשה רשימה . אתה יכול למצוא את הבלוק 'יצירת רשימה ' במגירת הרשימות, ויש לו רק שני שקעים. אבל אתה יכול לציין את מספר השקעים שאתה רוצה ברשימה על ידי לחיצה על הסמל הכחול והוספת פריטים, כפי שמתואר באיור .19-2



איור .19-3לחץ על הסמל הכחול בצור רשימה כדי לשנות את מספר הפריטים

אתה יכול לחבר כל סוג של נתונים לשקעי "פריט" של יצירת רשימה. בדוגמה של phoneNumbers, פריטים צריכים להיות אובייקטי טקסט, לא מספרים, מכיוון שלמספרי טלפון יש מקפים וסמלי עיצוב אחרים שלא ניתן להכניס לאובייקט מספר, ולא תבצע חישובים כלשהם על המספרים (שבהם במקרה, תרצה אובייקטים מספר, במקום זאת).

בחירת פריט ברשימה

בזמן שהאפליקציה שלך פועלת, תצטרך לבחור פריטים מהרשימה; לדוגמה, שאלה מסוימת כאשר המשתמש עובר חידון או מספר טלפון מסוים שנבחר מתוך רשימה.
אתה ניגש לפריטים ברשימה באמצעות אינדקס; כלומר, על ידי ציון מיקום ברשימה. אם רשימה כוללת שלושה פריטים, אתה יכול לגשת לפריטים על ידי שימוש במדדים 2 ,1ו-3.
אתה יכול להשתמש בבלוק פריט הבחירה כדי לתפוס פריט מסוים, כפי שמוצג באיור .19-3

איור .19-4בחירת הפריט השני ברשימה

עם פריט בחירה, אתה מחבר את הרשימה הרצויה לשקע הראשון, ואת האינדקס הרצוי לשקע השני. עבור מדגם מספר הטלפון הזה, התוצאה של בחירת הפריט השני היא "333-4444".

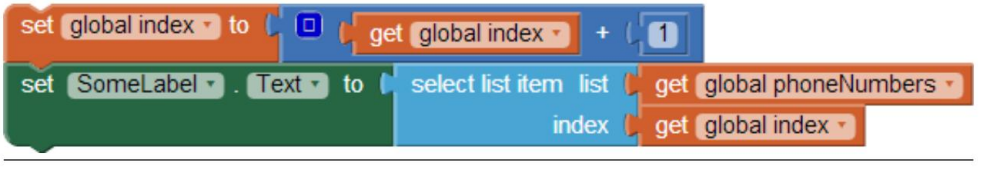
שימוש באינדקס כדי לעבור רשימה

באפליקציות רבות, תגדיר רשימה של נתונים ולאחר מכן תאפשר למשתמש לעבור (או לחצות) אותם. חידון הנשיאים בפרק 8מספק דוגמה טובה לכך: באותה אפליקציה, כאשר המשתמש לוחץ על כפתור הבא, הפריט הבא נבחר מרשימת שאלות ומוצג.

הסעיף הקודם הראה כיצד לבחור את הפריט השני ברשימה, אבל איך בוחרים את הפריט הבא? כאשר אתה חוצה רשימה, מספר הפריט שאתה בוחר משתנה בכל פעם; זה המיקום הנוכחי שלך ברשימה. לכן, עליך להגדיר משתנה כדי לייצג את המיקום הנוכחי. "אינדקס" הוא השם הנפוץ למשתנה כזה, והוא בדרך כלל מאותחל ל-1 (המיקום הראשון ברשימה), כפי שמוצג באיור 19-4.

איור 19-5. אתחול המדד המשתנה ל-1

כאשר המשתמש עושה משהו כדי לעבור לפריט הבא, אתה מגדיל את האינדקס משתנה על ידי הוספת ערך של 1אליו, ולאחר מכן בחר מתוך הרשימה באמצעות הערך המוגדל הזה. איור 19-5מציג את הבלוקים לעשות זאת.



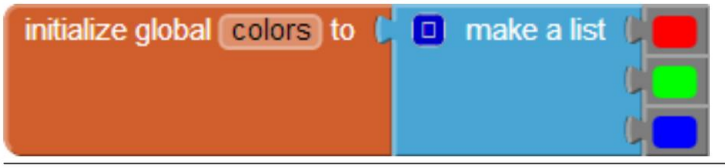
איור 19-6. הגדלת ערך האינדקס ושימוש בערך המוגדל כדי לבחור את פריט הרשימה הבא

דוגמה: מעבר ברשימת צבעי צבע

הבה נשקול אפליקציה לדוגמה שבה המשתמש יכול לעיין בכל צבע צבע פוטנציאלי עבור הבית שלו על ידי הקשה על "כפתור צבע". בכל פעם שהמשתמש מקיש, צבע הכפתור משתנה. כאשר המשתמש עובר את כל הצבעים האפשריים, האפליקציה חוזרת לראשון.

עבור דוגמה זו, נשתמש במספר צבעים בסיסיים. עם זאת, אתה יכול להתאים אישית את בלוקי הקוד כך שיעברו דרך כל קבוצה של צבעים.

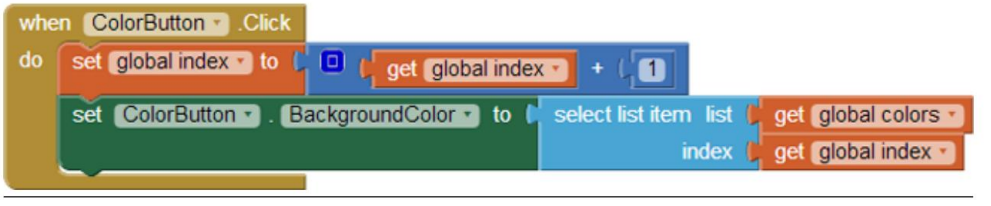
הצעד הראשון שלך הוא להגדיר משתנה רשימה עבור רשימת הצבעים ולאתחל אותו עם כמה צבעי צבע כפריטים, כפי שמתואר באיור 19-6.



איור 19-7. אתחול צבעי הרשימה עם רשימה של צבעי צבע

לאחר מכן, הגדר משתנה אינדקס שעוקב אחר המיקום הנוכחי ברשימה. זה אמור להתחיל ב-1. אתה יכול לתת למשתנה שם תיאורי כגון currentColorIndex, אבל אם אתה לא עוסק במספר אינדקסים באפליקציה שלך, אתה יכול פשוט לקרוא לו "אינדקס", כמו באיור 19-4.

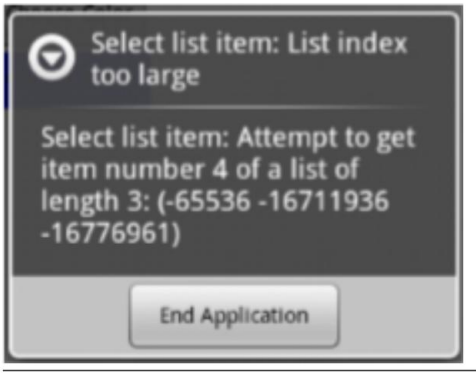
המשתמש עובר לצבע הבא ברשימה על ידי לחיצה על כפתור הצבע. בכל הקשה, יש להגדיל את האינדקס וצבע הרקע של הלחצן אמור להשתנות לפריט שנבחר כעת, כפי שמוצג באיור 19-7.



איור 19-8. כל לחיצה על הכפתור משנה את צבעה

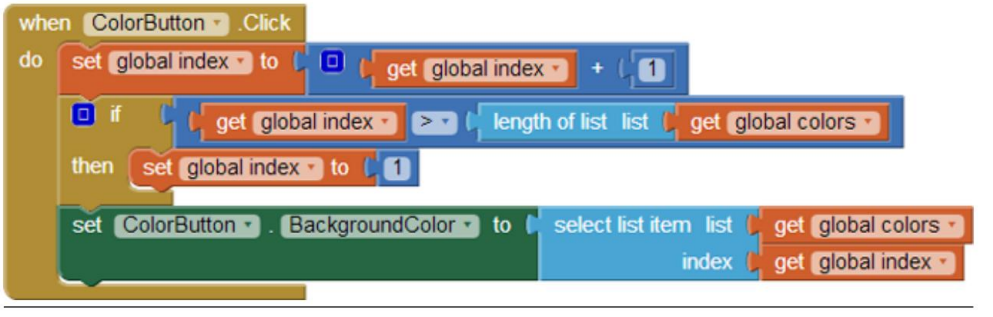
נניח שרקע הכפתור מוגדר בהתחלה לאדום. Component Designer-בפעם הראשונה שהמשתמש לוחץ על הכפתור, האינדקס משתנה מערכו ההתחלתי של 1 ל-2, וצבע הרקע של הכפתור משתנה לפריט השני ברשימה, ירוק. בפעם השנייה שהמשתמש מקיש עליו, האינדקס משתנה מ-2 ל-3, וצבע הרקע עובר לכחול.

אבל מה לדעתך יקרה בפעם הבאה שהמשתמש יקיש עליו?
אם אמרת שתהיה שגיאה, אתה צודק! האינדקס יהפוך ל-4 והאפליקציה תנסה לבחור את הפריט הרביעי ברשימה, אך ברשימה יש רק שלושה פריטים. האפליקציה תאלץ לסגור, או להפסיק, והמשתמש יראה הודעת שגיאה כמו זו באיור 19-8.



איור .19-9הודעת השגיאה המוצגת כאשר האפליקציה מנסה לבחור פריט רביעי מתוך רשימה של שלושה פריטים

ברור שההודעה הזו היא לא משהו שאתה רוצה שמשתמשי האפליקציה שלך יראו. כדי למנוע בעיה זו, הוסף בלוק אם כדי לבדוק אם הצבע האחרון ברשימה הושג. אם כן, ניתן לשנות את האינדקס בחזרה ל-1 כך שהצבע הראשון יוצג שוב, כפי שמוצג באיור .19-9



איור .19-10שימוש if-בכדי לבדוק אם ערך האינדקס גדול מאורך הרשימה

כאשר המשתמש מקיש על הכפתור, האינדקס מוגדל ולאחר מכן נבדק כדי לראות אם הערך שלו גדול מדי. המדד מושווה לאורך הרשימה, לא ;3כך האפליקציה שלך תעבוד גם אם תוסיף פריטים לרשימה. על ידי בדיקה אם האינדקס גדול מאורך הרשימה שלך (לעומת בדיקה אם הוא גדול מהמספר הספציפי ,(3ביטלת תלות בקוד באפליקציה שלך. תלות קוד היא מונח תכנות המתאר קוד שמוגדר באופן ספציפי מדי וחסר גמישות. לפיכך, אם אתה משנה משהו במקום אחד -בדוגמה שלנו כאן, אתה מוסיף פריטים לרשימה שלך -תצטרך לחפש כל מופע שבו אתה משתמש ברשימה הזו ולשנות אותה במפורש.

כפי שאתה יכול לדמיין, תלות מסוג זה עשויה להסתבך מהר מאוד, ו
הם בדרך כלל מובילים להרבה יותר באגים שתוכל לרדוף אחריהם, גם כן. למעשה, ה

דוגמה: מעבר ברשימת צבעי צבע

העיצוב עבור אפליקציית הצבע שלנו מכיל עוד תלות בקוד כפי שהוא מתוכנת כעת. אתה יכול להבין מה זה?

אם שינית את הצבע הראשון ברשימה שלך מאדום לצבע אחר, האפליקציה לא יעבוד כמו שצריך אלא אם כן זכרתם לשנות גם את ה- Button.BackgroundColor הראשוני שהגדרתם. Component Designer-בהדרך לבטל את התלות בקוד היא להגדיר את הצבע הראשוני ColorButton.BackgroundColorלצבע הראשון ברשימה ולא לצבע ספציפי. מכיוון ששינוי זה כרוך בהתנהגות המתרחשת כאשר האפליקציה שלך נפתחת לראשונה, אתה עושה זאת ב- Screen.Initializeמטפל באירועים המופעל כאשר אפליקציה מופעלת, כפי שמוצג באיור 19-10.

איור 19-11. הגדרת צבע הרקע של הכפתור לצבע הראשון ברשימה כאשר האפליקציה מופעלת

יצירת טפסי קלט ונתונים דינמיים

אפליקציית Colorהקודמת כללה רשימה סטטית: כזו שהאלמנטים שלה מוגדרים על ידי המתכנת (אתה) והפריטים שלו לא משתנים אלא אם כן אתה משנה את הבלוקים עצמם. עם זאת, לעתים קרובות יותר, אפליקציות עוסקות בנתונים דינמיים: מידע שמשתנה על סמך משתמש הקצה שמזין פריטים חדשים, או פריטים חדשים שנטענים ממסד נתונים או מקור מידע אינטרנטי. בחלק זה, אנו דנים באפליקציית Note Takerלדוגמה, שבה המשתמש מזין הערות בטופס ויכול להציג את כל ההערות הקודמות שלה.

הגדרת רשימה דינמית

אפליקציות כגון Note Takerמתחילות ברשימה ריקה. כאשר אתה רוצה רשימה שמתחילה ריקה, אתה מגדיר אותה באמצעות בלוק צור רשימה ריקה , כפי שמתואר באיור 19-11.

איור 19-12. הבלוקים להגדרת רשימה דינמית אינם מכילים פריטים מוגדרים מראש

הוספת פריט

בפעם הראשונה שמישהו מפעיל את האפליקציה, רשימת ההערות ריקה. אך כאשר המשתמש מקליד נתונים מסוימים בטופס ומקיש על שלח, הערות חדשות יתווספו לרשימה. הטופס עשוי להיות פשוט כמו זה שמוצג באיור 19-12.

איור 19-13. שימוש בטופס להוספת פריטים חדשים לרשימת ההערות

כאשר המשתמש מקליד הערה ומקיש על כפתור שלח, האפליקציה קוראת להוספה פריטים לרשימה פונקציה לצרף את הפריט החדש לרשימה, כפי שמוצג באיור 19-13.

איור 19-14. קורא הוסף פריטים לרשימה כדי להוסיף את ההערה החדשה כאשר המשתמש מקיש על הכפתור Submit

אתה יכול למצוא את הוספת הפריטים לבלוק הרשימה במגירת הרשימה. זהירות: יש גם בלוק הוספה לרשימה , אבל זה הוא בלוק נדיר למדי המשמש לצרף רשימה שלמה אחת לאחרת.

הצגת רשימה

התוכן של משתני רשימה, כמו כל המשתנים, אינו גלוי למשתמש. הבלוקים באיור 19-13 מוסיפים פריטים לרשימה בכל פעם שמופעל SubmitButton.Click, אך המשתמש לא יקבל משוב שהרשימה גדלה עד שתתכנת בלוקים נוספים כדי להציג בפועל את תוכן הרשימה.

הדרך הפשוטה ביותר להציג רשימה בממשק המשתמש של האפליקציה שלך היא להשתמש באותה שיטה שבה אתה משתמש להצגת מספרים וטקסט: שים את הרשימה במאפיין Text של רכיב תווית , כפי שמוצג באיור 19-14.

איור 19-15. הצגת הרשימה למשתמש על ידי הצבתה בתווית.

למרבה הצער, השיטה הפשוטה הזו להצגת רשימה אינה אלגנטית במיוחד; זה מציב את רשימה בסוגריים, כאשר כל פריט מופרד ברווח ולא בהכרח באותה שורה. לדוגמה, אם המשתמש יקליד, "האם אי פעם אסיים את הספר הזה?" בתור ההערה הראשונה, ו"אני שוכח איך הבן שלי נראה!" בתור השני, האפליקציה תציג את רשימת ההערות בדומה למה שאנו רואים באיור 19-15.

איור 19-16. ערכים אלה רשומים באמצעות עיצוב ברירת מחדל

בפרק 20, אתה יכול לראות דרך מתוחכמת יותר להציג רשימה.

הסרת פריט מרשימה

אתה יכול להסיר פריט מרשימה באמצעות בלוק הסר פריט רשימה , כפי שמוצג באיור 19-16.

איור 19-17. הסרת פריט מרשימה

הבלוקים באיור 19-16 מסירים את הפריט השני מהרשימה הנקראת הערות.
עם זאת, בדרך כלל, לא תרצה להסיר פריט קבוע (למשל, 2), אלא תספק מנגנון למשתמש לבחור את הפריט להסרה.
אתה יכול להשתמש ברכיב ListPicker כדי לספק למשתמש דרך לבחור פריט. ListPicker מגיע עם כפתור משויך. כאשר לוחצים על הכפתור, ListPicker מציג את הפריטים של רשימה שממנה המשתמש יכול לבחור אחד. כאשר המשתמש בוחר פריט, האפליקציה יכולה להסיר אותו.
קל לתכנת את ListPicker אם אתה מבין את אירועי המפתח שלו, BeforePicking ו-AfterPicking, ומאפייני המפתח שלו, Elements, Selection ו-SelectionIndex (ראה טבלה 19-1).

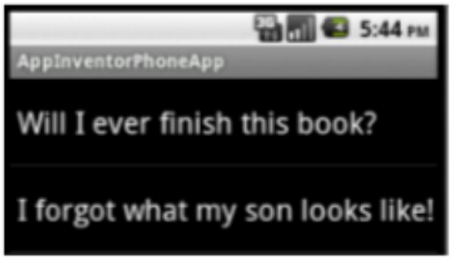
טבלה 19-1. האירועים והמאפיינים העיקריים של רכיב ListPicker

תכונה	מִקרֶה
לפני בחירה: מופעל בעת לחיצה על הכפתור. אלמנטים: רשימת האפשרויות.	
AfterPicking: מופעל כאשר המשתמש עושה בחירה. בחירה: בחירת המשתמש.	
SelectionIndex: מיקום בחירה.	

המשתמש מפעיל את אירוע ListPicker.BeforePicking על ידי הקשה על ListPicker's כפתור משויך. במטפל האירועים , ListPicker.BeforePicking תגדיר את המאפיין ListPicker.Elements למשתנה רשימה כך שהנתונים ברשימה יוצגו. עבור האפליקציה Taker, Note תגדיר את Elements למשתנה הערות שמכיל את רשימת ההערות שלך, כפי שמוצג באיור 19-17.

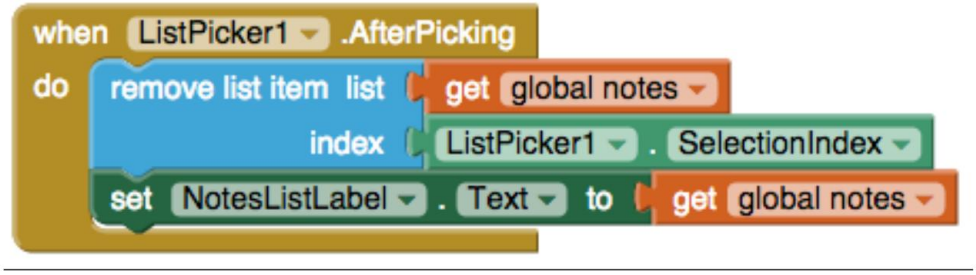
איור 19-18. המאפיין Elements של ListPicker1 מוגדר לרשימת ההערות

עם בלוקים אלה, הפריטים של הערות הרשימה יופיעו ב- ListPicker. אם היו שני הערות, זה היה מופיע כפי שמוצג באיור 19-18.



איור 19-19. רשימת ההערות מופיעה ב- ListPicker

כאשר המשתמש בוחר פריט ברשימה, הוא מפעיל את ListPicker.AfterPicking מִקרֶה. במטפל אירועים זה, אתה יכול לגשת לבחירת המשתמש במאפיין ListPicker.Selection .
עם זאת, המטרה שלך בדוגמה זו היא להסיר פריט מהרשימה, והסרת פריט מבלוק הרשימה מצפה לאינדקס, לא לפריט. המאפיין Selection של ListPicker הוא הנתונים בפועל (ההערה), לא האינדקס. לכן, עליך להשתמש במאפיין SelectionIndex במקום זאת מכיוון שהוא מספק לך את האינדקס של הפריט הנבחר. יש להגדיר אותו כאינדקס של בלוק הסר פריט רשימה , כפי שמוצג באיור 19-19.

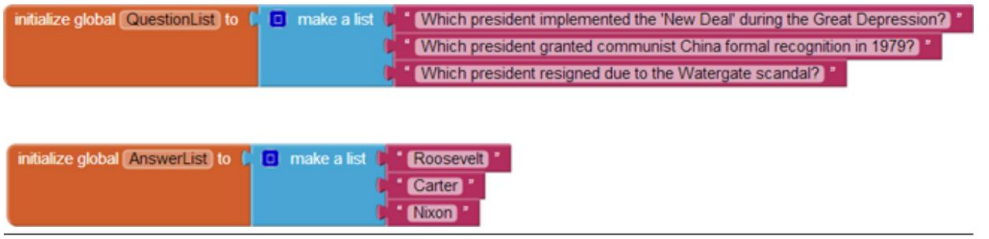


איור 19-20. הסרת פריט באמצעות ListPicker.SelectionIndex

רשימות של רשימות

הפריטים ברשימה יכולים להיות מכל סוג, כולל מספרים, טקסט, צבעים או ערכים בוליאניים (true/false). אבל, הפריטים של רשימה יכולים גם, בעצמם, להיות רשימות. מבני נתונים מורכבים כאלה נפוצים. לדוגמה, ניתן להשתמש ברשימת רשימות כדי להמיר את חידון הנשיאים (פרק 8) לחידון רב-ברירה. בואו נסתכל שוב על הבסיס

מבנה חידון הנשיאים, שהוא רשימה של שאלות ורשימת תשובות, כפי שמוצג באיור 19-20.



איור 19-21. רשימת שאלות ורשימת תשובות

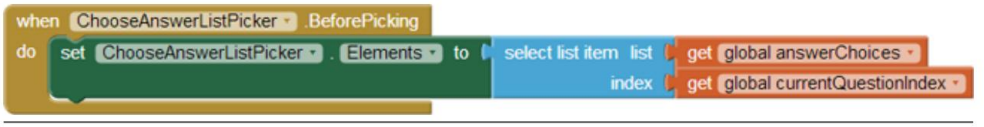
בכל פעם שהמשתמש עונה על שאלה, האפליקציה בודקת אם היא נכונה על ידי השוואת התשובה לפריט הנוכחי ברשימת התשובות.

כדי להפוך את החידון לבחירה מרובה, עליך לשמור רשימה נוספת, כזו המאחסנת את האפשרויות עבור כל תשובה לכל שאלה. אתה מציין נתונים כאלה על ידי הצבת שלושה בלוקים של יצירת רשימה בתוך גוש יצירת רשימה פנימי , כפי שמוצג באיור 19-21.

איור 19-22. רשימה של רשימות נוצרת על ידי הוספת יצירת בלוקים של רשימה כפריטים בתוך בלוק יצירת רשימה פנימית

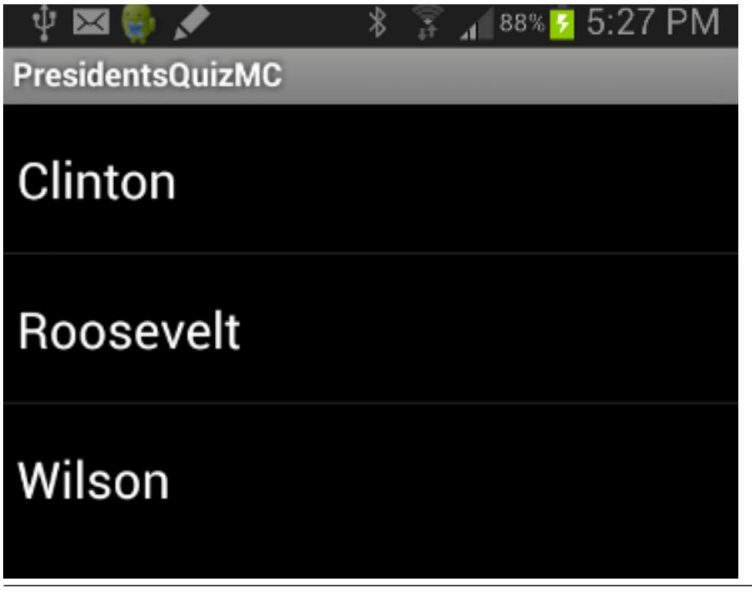
כל פריט במשתנה answerChoicesהוא בעצמו רשימה המכילה שלושה פריטים. אם תבחר פריט מתוך answerChoices, התוצאה היא רשימה. כעת, לאחר שמילאת את התשובות מרובות הברירות שלך, כיצד תציג זאת למשתמש?

כמו באפליקציית Note Taker, אתה יכול להשתמש ב- ListPickerכדי להציג בפניו את האפשרויות המשתמש. אם האינדקס היה נקרא currentQuestionIndex, האירוע ListPicker.BeforePickingיופיע כפי שמוצג באיור 19-22.



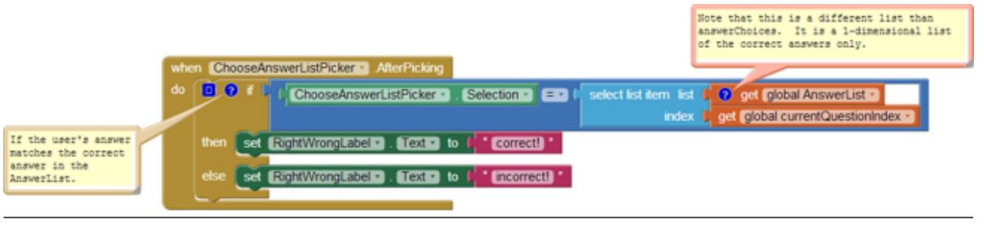
איור 19-23. שימוש בבורר הרשימות כדי להציג אחת מרשימת אפשרויות התשובות ל-
מִשׁתַמֵשׁ

בלוקים אלה יקחו את רשימת המשנה הנוכחית של אפשרויות תשובות ויאפשרו למשתמש
לבחור ממנו. אז אם currentQuestionIndex היה 1, ListPicker מציג רשימה כמו זו באיור 19-23.



איור 19-24. אפשרויות התשובה שהוצגו למשתמש עבור השאלה הראשונה

כאשר המשתמש בוחר, אתה בודק את התשובה עם הבלוקים המוצגים באיור 19-24.



איור 19-25. בודקים האם המשתמש בחר בתשובה הנכונה

בלוקים אלו, הבחירה של המשתמש מה- ListPickerמושווה לתשובה הנכונה, המאוחסנת ברשימה שונה, AnswerList(מכיוון answerChoices-שמספק רק את האפשרויות ואינו מציין את התשובה הנכונה).

סיכום

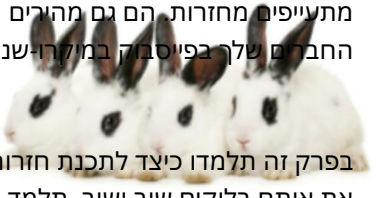
רשימות משמשות כמעט בכל אפליקציה שאתה יכול לחשוב עליה. הבנת איך הם עובדים היא בסיסית לתכנות. בפרק זה, חקרנו את אחת מתבניות התכנות הנפוצות ביותר: שימוש במשתנה אינדקס שמתחיל בתחילת הרשימה ומתגבר עד שכל פריט רשימה מעובד. אם אתה יכול להבין ולהתאים אישית את הדפוס הזה, אתה אכן מתכנת!

לאחר מכן כיסינו כמה מהמנגנונים האחרים למניפולציה ברשימות, כולל טפסים טיפוסיים לאפשר למשתמש להוסיף ולהסיר פריטים. תכנות כזה דורש רמה נוספת של הפשטה, מכיוון שאתה צריך לדמיין את הנתונים הדינמיים לפני שהם באמת קיימים. אחרי הכל, הרשימות שלך ריקות עד שהמשתמש מכניס בהן משהו. אם אתה יכול להבין את זה, אולי אפילו תחשוב לעזוב את העבודה היומיומית שלך.
סיימנו את הפרק בהכנסת מבנה נתונים מורכב, רשימה של רשימות.
זהו ללא ספק מושג קשה, אבל חקרנו אותו על ידי שימוש בנתונים קבועים: אפשרויות התשובות עבור חידון רב-ברירה. אם שלטת בזה ובשאר הפרק, המבחן הסופי שלך הוא זה: צור אפליקציה שמשתמשת ברשימת רשימות אבל עם נתונים דינמיים. דוגמה אחת תהיה אפליקציה שבעזרתה אנשים יכולים ליצור חידונים מרובי-ברירה משלהם, ולהרחיב עוד יותר את אפליקציית MakeQuizבפרק .10בהצלחה!

בזמן שאתה חושב על איך תתמודד עם זה, הבינו שהחקירה שלנו על רשימות לא נעשה. בפרק הבא, אנו ממשיכים בדיון ומתמקדים באיטרציה של רשימה עם טוויסט: החלת פונקציות על כל פריט ברשימה.

בלוקים חוזרים

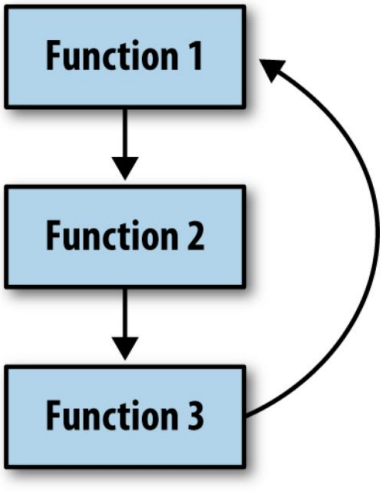
אם יש משהו שמחשבים טובים בו, זה לחזור על דברים -כמו ילדים קטנים, הם אף פעם לא מתעייפים מחזרות. הם גם מהירים מאוד ויכולים לעשות דברים כמו לעבד את כל רשימת החברים שלך בפייסבוק במיקרו-שנייה.


איור 1-20

בפרק זה תלמדו כיצד לתכנת חזרות עם בלוקים חוזרים מיוחדים במקום להעתיק ולהדביק את אותם בלוקים שוב ושוב. תלמד כיצד לשלוח הודעת SMS לכל מספר טלפון ברשימה וכיצד להוסיף רשימה של מספרים. תלמד גם שחסימות חוזרות יכולות לפשט משמעותית אפליקציה.

שליטה בביצוע אפליקציה: הסתעפות ו בלולאה

בפרקים הקודמים, למדת שאתה להגדיר התנהגות של אפליקציה עם קבוצה של מטפלים באירועים -אירועים והפונקציות שצריך להורג בתגובה. למדת גם שהתגובה לאירוע היא לרוב לא רצף ליניארי של פונקציות ויכולה להכיל בלוקים שמתבצעים רק בתנאים מסוימים.



איור .2-20בלוקים חוזרים גורמים לתוכנית לעבור לולאה

בלוקים חוזרים הם הדרך השנייה שבה an האפליקציה מתנהגת בצורה לא ליניארית. ממש כאילו ועוד אם בלוקים מאפשרים לתוכנית להסתעף, בלוקים חוזרים מאפשרים לתוכנית להסתעף; כלומר, לבצע קבוצה של פונקציות ואז לקפוץ חזרה למעלה בקוד ולעשות זאת שוב, כפי שמוצג באיור .1-20כאשר אפליקציה מופעלת, מונה תוכניות הפועל מתחת למכסה המנוע של האפליקציה עוקב אחר הפעולה הבאה שתתבצע. עד כה בדקתם אפליקציות שבהן מונה התוכניות מתחיל בראש מטפל באירועים ו(בתנאי)

מבצע פעולות מלמעלה למטה. עם בלוקים חוזרים, מונה התוכנית מתגבש בלולאות, ומבצע ברציפות את אותן פעולות.

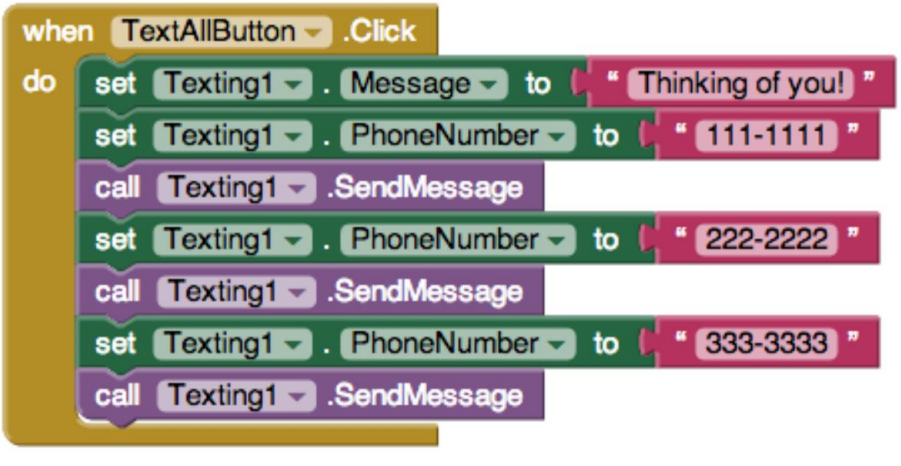
App Inventor מספק מספר בלוקים חוזרים, כולל עבור כל וזמן , בהם נתמקד בפרק זה. foreach משמש לציון פונקציות שיש לבצע בכל פריט ברשימה. אז אם יש לך רשימה של מספרי טלפון, אתה יכול לציין שיש לשלוח טקסט לכל מספר ברשימה

בלוק ה- while הוא כללי יותר מהחסימה לכל אחד. עם זה, אתה יכול לתכנת בלוקים שחוזרים ללא הרף עד שמצב שרירותי כלשהו משתנה. אתה יכול להשתמש בלוקים בעוד כדי לחשב נוסחאות מתמטיות כגון הוספת ה-n המספרים הראשונים או חישוב הפקטוריאלי של n. אתה יכול גם להשתמש בזמן כאשר אתה צריך לעבד שתי רשימות בו זמנית; עבור כל תהליכים רק רשימה בודדת בכל פעם.

איטרציה של פונקציות ברשימה עם עבור כל אחת

פרק 18 מדגים אפליקציה שמתקשרת באופן אקראי למספר טלפון אחד ברשימה. התקשרות אקראית לחבר אחד עשויה להצליח לפעמים, אבל אם יש לך חברים כמו שלי, הם לא תמיד עונים. אסטרטגיה שונה תהיה לשלוח "חושב עליך!" שלח הודעה לכל החברים שלך וראה מי מגיב ראשון (או הכי מקסים!).

אחת הדרכים ליישם אפליקציה כזו היא פשוט להעתיק את הבלוקים לשליחת טקסט ליחיד מספר ולאחר מכן הדבק אותם עבור כל חבר שאליו תרצה לשלוח הודעת טקסט, כפי שמוצג באיור 20-2.



איור 20-3. העתקה והדבקה של הבלוקים עבור כל מספר טלפון להודעות טקסט

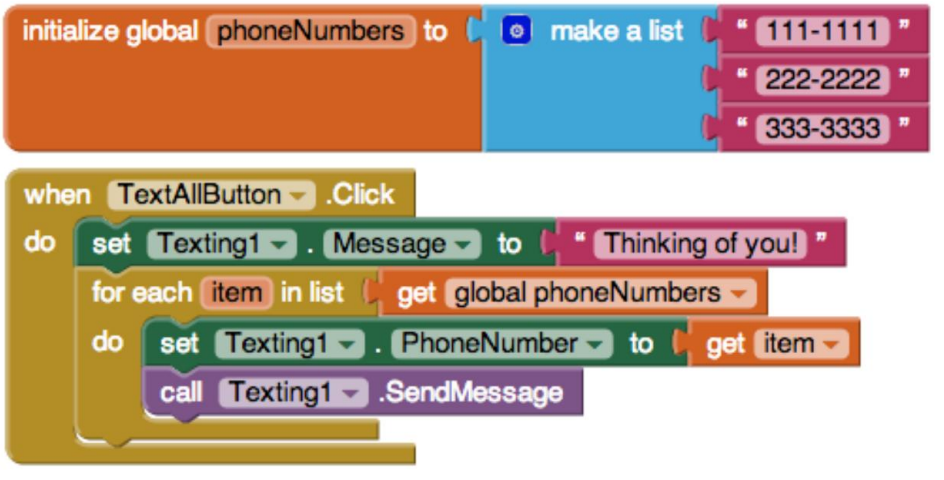
שיטת העתק-הדבק מסוג "כוח אכזרי" היא fine אם יש לך רק כמה בלוקים לחזור עליהם. אבל, אם אתה מתמודד עם כמויות גדולות של נתונים או נתונים שישתנו, אתה לא

רוצה לשנות את האפליקציה שלך בשיטת העתק-הדבק בכל פעם שאתה מוסיף או מסיר מספר טלפון מהרשימה שלך.

עבור כל בלוק מספק פתרון טוב יותר. אתה מגדיר משתנה phoneNumbers

עם כל המספרים ולאחר מכן עטפו aעבור כל בלוק סביב עותק בודד של הבלוקים שברצונכם לבצע. איור 20-3מציג את הפתרון עבור כל פתרון הודעות טקסט א

קְבוּצָה.



איור .20-4שימוש ב- עבור כל בלוק כדי לבצע את אותם בלוקים עבור כל פריט ברשימה

אתה יכול לקרוא את הקוד הזה בתור, "עבור כל פריט (מספר טלפון) ברשימה מספרי טלפון, הגדר את מספר הטלפון של אובייקט הטקסט לפריט ושלח את הודעת הטקסט."

בחלק העליון של כל בלוק, אתה מציין את הרשימה שתעובד. לבלוק יש גם משתנה מציין מיקום שמגיע עם עבור כל אחד. כברירת מחדל, מציין מיקום זה נקרא "פריט". אתה יכול להשאיר אותו כך או לשנות את שמו. משתנה זה מייצג את הפריט הנוכחי המעובד ברשימה.

אם רשימה כוללת שלושה פריטים, הבלוקים הפנימיים יבוצעו שלוש פעמים. אומרים שהבלוקים הפנימיים כפופים, או מקוננים בתוך, עבור כל בלוק. אנו אומרים שמונה התוכנית "מתגלגל" בחזרה כשהוא מגיע לבלוק התחתון בתוך

לכל אחד.

מבט מקרוב על לולאה

הבה נבחן את המכניקה של כל בלוקים בפירוט, מכיוון שהבנת לולאות היא בסיסית לתכנות. כאשר המשתמש מקיש על TextAllButtonועל

מטפל באירועים מופעל, הפעולה הראשונה שבוצעה היא הגדר Texting1.Messageלחסימה, מה שמגדיר את ההודעה ל"חושב עליך!" בלוק זה מבוצע פעם אחת בלבד.
לאחר מכן מתחיל עבור כל בלוק. לפני ביצוע הבלוקים המקוננים של aעבור כל אחד מהם, פריט משתנה מציין המיקום מוגדר למספר הראשון ברשימת מספרי הטלפון .(1111–111) זה קורה אוטומטית; עבור כל אחד משחררים אותך מהצורך להתקשר לפריט ידנית לפריט בחר רשימה. לאחר שהפריט הראשון נבחר לפריט המשתנה, הבלוקים בתוך עבור כל אחד מבוצעים בפעם הראשונה. ה
מאפיין SMS1.PhoneNumberמוגדר לערך של הפריט (1111–111)וההודעה נשלחת.

לאחר הגעה לבלוק האחרון בתוך aעבור כל אחד (גוש ,(Texting.SendMessage האפליקציה "נכנסת" חזרה לראש ה- forכל אחד ומכניסה אוטומטית את הפריט הבא ברשימה (2222–222)לפריט המשתנה. לאחר מכן חוזרות על שתי הפעולות בתוך עבור כל אחת , ושולחות את ה"חושב עליך!" טקסט למספר .2222–222לאחר מכן, האפליקציה חוזרת בלולאה ומגדירה את הפריט לפריט האחרון ברשימה .(3333–333)הפעולות חוזרות על עצמן בפעם השלישית, שליחת הטקסט השלישי.
מכיוון שהפריט הסופי ברשימה עבר עיבוד, ה- עבור כל לולאה נעצר ב-
הנקודה הזו. בשפת תכנות, אנו אומרים שהשליטה "קופצת" מחוץ ללולאה, מה שאומר שמונה התוכניות ממשיך להתמודד עם הבלוקים שמתחת לכל אחד. בדוגמה זו, אין בלוקים מתחתיו, כך שהמטפל באירוע מסתיים.

כתיבת קוד בר תחזוקה

למשתמש של האפליקציה, עבור כל פתרון שתואר זה עתה מתנהג בדיוק כמו שיטת "הכוח האכזרי" של העתקה ואז הדבקת בלוקים של הודעות טקסט. מנקודת מבטו של מתכנת, לעומת זאת, הפתרון עבור כל פתרון ניתן לתחזוקה וניתן להשתמש בו גם אם הנתונים (רשימת הטלפונים) מוזנים באופן דינמי.
תוכנה הניתנת לתחזוקה היא תוכנה שניתן לשנות בקלות מבלי להכניס באגים. עם הפתרון עבור כל אחד , אתה יכול לשנות את רשימת החברים שנשלחו להם טקסטים על ידי שינוי רק במשתנה הרשימה -אינך צריך לשנות את ההיגיון של התוכנית שלך (מטפל באירועים) כלל. הפוך זאת לשיטת ,brute-forceמחייבת אותך להוסיף בלוקים חדשים במטפל האירוע כאשר חבר חדש נוסף.
בכל פעם שאתה משנה את ההיגיון של תוכנית, אתה מסתכן בהחדרת באגים.
לא פחות חשוב, עבור כל פתרון יעבוד גם אם רשימת הטלפונים הייתה דינמית; כלומר, כזה שבו משתמש הקצה יכול להוסיף מספרים לרשימה. בניגוד לדוגמא שלנו, שיש לה שלושה מספרי טלפון מסוימים הרשומים בקוד, רוב האפליקציות עובדות עם נתונים דינמיים שמגיעים ממשתמש הקצה או ממקור אחר. אם עיצבת מחדש את האפליקציה הזו כך שמשתמש הקצה יוכל להזין את מספרי הטלפון, תצטרך להשתמש ב-a עבור כל פתרון, מכיוון שכאשר אתה כותב את התוכנית, אתה לא יודע אילו מספרים לשים בפתרון ה-force.-eturb

שימוש עבור כל אחד כדי להציג רשימה

כאשר אתה רוצה להציג את הפריטים של רשימה בטלפון, אתה יכול לחבר את הרשימה למאפיין הטקסט של תווית, כפי שמוצג באיור .20-4

איור .20-5הדרך הפשוטה להציג רשימה היא לחבר אותה ישירות לתווית

כאשר אתה מחבר רשימה ישירות למאפיין טקסט של תווית, פריטי הרשימה מוצגים בתווית כשורה אחת של טקסט, מופרדים ברווחים ומוכללים בסוגריים: (333–3333 222–2222 111–1111)

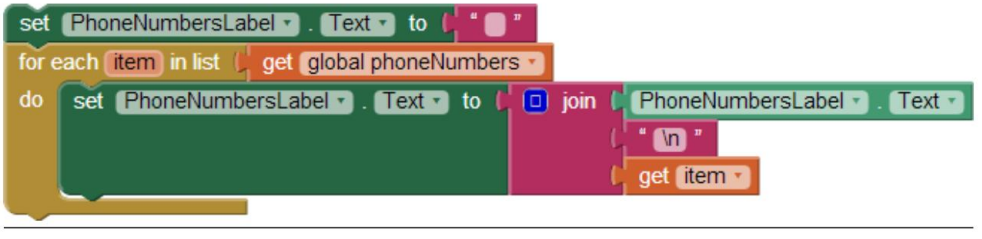
המספרים עשויים להשתרע על יותר משורה אחת או לא, תלוי כמה יש. המשתמש יכול לראות את הנתונים ואולי להבין שזו רשימה של מספרי טלפון, אבל זה לא מאוד אלגנטי. פריטי רשימה מוצגים בדרך כלל בשורות נפרדות או עם פסיקים המפרידים ביניהם.

כדי להציג רשימה נכונה, אתה צריך בלוקים שהופכים כל פריט רשימה ליחיד ערך טקסט עם העיצוב הרצוי. אובייקטי טקסט מורכבים בדרך כלל מאותיות, ספרות וסימני פיסוק. עם זאת, טקסט יכול גם לאחסן תווי בקרה מיוחדים, שאינם ממפים לתו שאתה יכול לראות. כרטיסייה, למשל, מסומנת ב-\t. (למידע נוסף על תווי בקרה, עיין בתקן Unicode לייצוג טקסט בכתובת http://www.unicode.org/standard/standard.html.)

ברשימת מספרי הטלפון שלנו, אנו רוצים תו חדש, המסומן ב-\n.
כאשר \nמופיע בגוש טקסט, זה אומר "עבור לשורה הבאה לפני שתציג את הפריט הבא." לפיכך, אובייקט הטקסט "111–1111\n222–2222\n333–3333" יופיע כך:

```
111–1111
222–2222
333–3333
```

כדי לבנות אובייקט טקסט כזה, אנו משתמשים ב-a עבור כל בלוק ו"מעבדים" כל פריט על ידי הוספתו יחד עם תו חדש למאפיין PhoneNumberLabel.Text , כפי שמוצג באיור .20-5



איור A .20-6עבור כל בלוק המשמש להצגת רשימה עם פריטים בשורות נפרדות

בואו נתחקה אחר הבלוקים כדי לראות איך הם עובדים. כפי שנדון בפרק ,15מעקב מראה כיצד כל משתנה או תכונה משתנה עם ביצוע הבלוקים. עם aעבור כל אחד, אנו רואים את הערכים לאחר כל איטרציה; כלומר, בכל פעם שהתוכנית עוברת את עבור כל לולאה.

לפני כל אחד , ,PhoneNumbersLabelמאותחל לטקסט הריק. כאשר עבור כל אחד מתחיל, האפליקציה ממקמת אוטומטית את הפריט הראשון ברשימה (111-1111)בפריט המשתנה של מציין המיקום. הבלוקים של כל אחד לאחר מכן מצטרפים עם PhoneNumbersLabel.Text(הטקסט הריק), \nוהפריט, ומגדירים את התוצאה ל- .PhoneNumbersLabel.Textלפיכך, לאחר האיטרציה הראשונה של ה- עבור כל אחד מהם, המשתנים הרלוונטיים מאחסנים את הערכים המוצגים בטבלה 20-1.

טבלה .20-1הערכים לאחר האיטרציה הראשונה

פריט	PhoneNumbersLabel.Text
111–1111	\n111–1111

מכיוון שהושגה התחתית של כל אחד , לולאות בקרה מגובות, ו הפריט הבא ברשימה (222–2222)מוכנס לפריט המשתנה. כאשר חוזרים על הבלוקים הפנימיים, הטקסט משרשר את הערך של (\n111– 1111) PhoneNumbersLabel.Textעם ,\nולאחר מכן עם פריט, שהוא כעת .222–2222לאחר איטרציה שנייה זו, המשתנים מאחסנים את הערכים המוצגים בטבלה 20-2.

טבלה .20-2הערכים לאחר האיטרציה השנייה

פריט	PhoneNumbersLabel.Text
2222–222n	\n111–1111222–2222

הפריט השלישי של הרשימה ממוקם אז בפריט, והגוש הפנימי חוזר על עצמו א פעם שלישית. הערך הסופי של המשתנים, לאחר איטרציה אחרונה זו, מוצג ב טבלה 20-3.

טבלה .20-3ערכי המשתנים לאחר האיטרציה הסופית

פריט	PhoneNumbersLabel.Text
3333–333n	\n111–1111333–3333 \2222–222n

לכן, לאחר כל איטרציה, התווית הופכת גדולה יותר ומכילה עוד מספר טלפון אחד (ועוד שורה חדשה אחת). בסוף הערך עבור כל אחד, PhoneNumbersLabel.Text מוגדר כך שהמספרים יופיעו כדלקמן: 111–1111

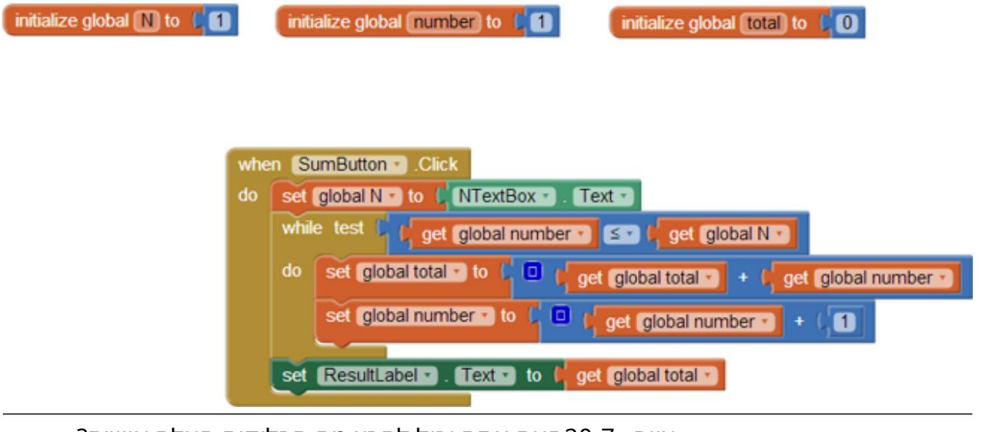
222–2222
333–3333

בלוק ה-while-do

בלוק ה- while-do הוא קצת יותר מסובך לשימוש מאשר עבור כל אחד. היתרון של בלוק ה- do while- טמון בכלליותו: עבור כל חזרה על פני רשימה, אבל בעוד יכול לחזור כל עוד כל תנאי שרירותי נכון.
כפי שלמדת בפרק 18, תנאי בודק משהו ומחזיר ערך של נכון או לא נכון. בלוקים while-do כוללים מבחן מותנה, בדיוק כמו אם בלוקים. אם הבדיקה של זמן מה מוערכת כנכונה, האפליקציה מבצעת את הבלוקים הפנימיים, ולאחר מכן חוזרת בלולאה ובודקת מחדש את הבדיקה. כל עוד הבדיקה מוערכת לאמיתה, הבלוקים הפנימיים חוזרים על עצמם. כאשר הבדיקה מוערכת כ-eslaf, האפליקציה יוצאת מהלולאה (כמו שראינו עם עבור כל בלוק) וממשיכה עם הבלוקים מתחת ל- while-do.

שימוש while-do בכדי לחשב נוסחה

הנה דוגמה לבלוק while-do שחוזר על פעולות. מה לדעתך עושים הבלוקים באיור 20-6? דרך אחת לגלות זאת היא לעקוב אחר כל בלוק (ראה פרק 15 למידע נוסף על מעקב), מעקב אחר הערך של כל משתנה תוך כדי תנועה.



איור 20-7. האם אתה יכול להבין מה הבלוקים האלה עושים?

הבלוקים בתוך לולאת while-doיחזרו על עצמם בעוד שמספר המשתנה קטן או שווה למשתנה N.
עבור אפליקציה זו, Nמוגדר למספר שמשתמש הקצה מקליד בתיבת טקסט .(NTextBox)נניח
שהמשתמש מקליד 3.המשתנים של האפליקציה ייראו כמו טבלה 20-4כאשר תגיע לראשונה לבלוק ה-
while-do .

טבלה .20-4ערכי משתנים כאשר while-do-המושגת לראשונה

	Nסך הכל	מספר
		3 1

הבלוק while-doבודק קודם את התנאי: האם מספר קטן או שווה ל-(□) ?N
בפעם הראשונה ששואלים שאלה זו, הבדיקה נכונה, ולכן הבלוקים המקוננים בתוך בלוק while-do
מבוצעים. סך מוגדר לעצמו (0)פלוס מספר ,(1)והמספר מוגדל. לאחר האיטרציה הראשונה של
הבלוקים בתוך ,while-doהמשתנה
הערכים הם כמפורט בטבלה 20-5.

טבלה .20-5ערכי המשתנים לאחר האיטרציה הראשונה של הבלוקים בתוך בלוק while

	Nסך הכל	מספר
3	2	1

באיטרציה השנייה, הבדיקה "N□רפסמ"עדיין נכונה ,(3□2)כך שהבלוקים הפנימיים
מוצאים להורג שוב. סך מוגדר לעצמו (1)פלוס מספר .(2)מספר מוגדל.
כאשר איטרציה שנייה זו מסתיימת, המשתנים מכילים את הערכים המפורטים בטבלה 20-6.

טבלה .20-6ערכי המשתנים לאחר האיטרציה השנייה

	Nסך הכל	מספר
3	3	3

האפליקציה חוזרת שוב בלולאה ובודקת את המצב. שוב, זה נכון ,(3□3)אז הבלוקים מבוצעים בפעם
השלישית. כעת, הסכום מוגדר לעצמו (3)בתוספת מספר ,(3)כך שהוא הופך ל-6. מספר מוגדל ל-4, כפי
שמוצג בטבלה 20-7.

טבלה .7-20הערכים לאחר האיטרציה השלישית

מספר	סך הכל N	
6	4	3

לאחר איטרציה שלישית זו, האפליקציה חוזרת שוב פעם אחת לראש הזמן
לַעֲשׂוֹת. כאשר הבדיקה "N רפסמ"פועלת הפעם, היא בודקת ,3 4אשר מוערכת לשווא.
לפיכך, הבלוקים המקוננים של while-doאינם מבוצעים שוב, ומטפל האירועים משלים.

אז מה עשו הבלוקים האלה? הם ביצעו את אחת הפעולות המתמטיות הבסיסיות ביותר:
ספירת מספרים. לא משנה מה המספר שהמשתמש יקליד, האפליקציה תדווח על סכום
המספרים ,N..1כאשר Nהוא המספר שהוזן. בדוגמה זו, Nהוא ,3כך שהאפליקציה הגיעה
עם סך של .1+2+3=6אם המשתמש היה מקליד ,4האפליקציה הייתה מחשבת .10

סיכום

מחשבים טובים בלחזור על אותה פונקציה שוב ושוב. חשבו על כל חשבונות הבנק שמעובדים
לצבירת ריבית, כל הציונים המעובדים לחישוב ממוצעי הציונים של התלמידים, ועוד אינספור
דוגמאות יומיומיות שעבורן מחשבים משתמשים בחזרה כדי לבצע משימה.

פרק זה חקר שניים מהבלוקים החוזרים של .App Inventorעבור כל בלוק
מחיל קבוצה של פונקציות על כל רכיב ברשימה. על ידי שימוש בו, אתה יכול לעצב קוד עיבוד
שעובד על רשימה מופשטת במקום נתונים קונקרטיים. קוד כזה ניתן לתחזוקה יותר; ואם
הנתונים לעיבוד הם דינמיים, זה נדרש.
בהשוואה לכל אחד, while-doהוא כללי יותר: אתה יכול להשתמש בו כדי לעבד רשימה, אבל אתה
יכול להשתמש בו גם לעיבוד סינכרוני של שתי רשימות או לחשב נוסחה. עם ,while-doהחסימות
הפנימיות מבוצעות ברציפות כל עוד מצב מסוים נכון. לאחר ביצוע החסימות בתוך הזמן , לולאות בקרה
מגובות ומצב הבדיקה נוסה שוב. רק כאשר הבדיקה מוערכת כ-eslaf מסתיים בלוק ה- . while-do