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 מכאבי הגדילה האלה, 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 האלה אולי

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 שהעצה של הלוחש לא חלה על תוכנה.
למעשה, זה הפוך ממה שאתה צריך לעשות. ההיסטוריה של התוכנה עמוסה בפתרונות מעולים שאין להם 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 למתכנתים מעולים. רוב פרויקטי התוכנה נתקלים בבעיות רבות כל כך עד שהם לעולם 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 המוצלחת 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 הוא 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 לגבי פרטים או שלמות או ממשק 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 ואז 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 אז 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 מקשה אחת בכל פעם, תוך כדי בדיקה. אתה תמצא באגים, בסדר, אבל זעירים שאתה יכול בקלות להחליק. והתהליך יהפוך למספק 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 שאתה מתמודד 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.

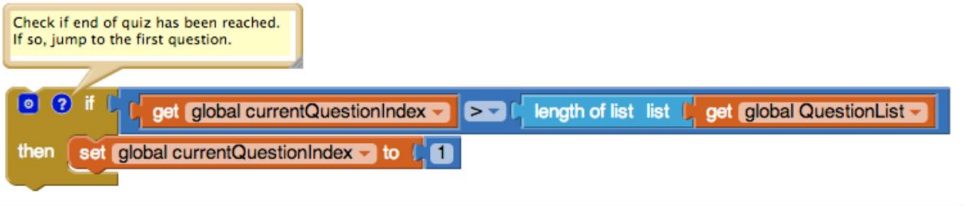
ספרים שלמים נכתבו על מתודולוגיות שונות של עיצוב תוכניות. יש אנשים שמשתמשים בדיאגרמות כגון תרשימים או תרשימי מבנה לעיצוב, בעוד שאחרים מעדיפים טקסט וסקיצות בכתב יד. יש אנשים המאמינים שכל "עיצוב" צריך

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 התוכניות שאין לו שום קשר לשפת תכנות מסוימת. התמודדות בו זמנית גם 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, וודאו שברור לכם מה אתם רוצים שהיא תעשה, ותעדו את מה שמצאתם 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. באתה יכול להוסיף App Inventor-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)ולהתמודד 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, תוכל 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 פניה כלפי חוץ, 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 התוכניות שעוקב אחר איזו פקודה 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 איך התוכנית תתנהג בפועל. כאשר אתה מוכן, אתה עובר למצב בדיקה: משחק את התפקיד של משתמש הקצה ובדוק את 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.

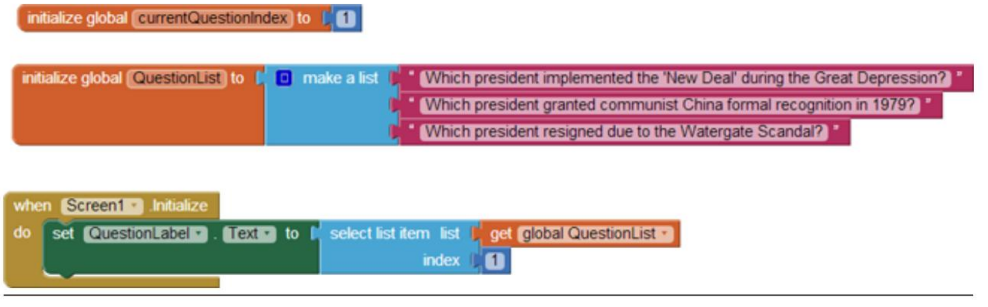
כאשר אתה מתחיל לתכנת, יש לך רק מודל חלקי של איך מחשב התוכנית עובדת - כל התהליך נראה כמעט קסום. אתה מתחיל עם כמה אפליקציות פשוטות: לחיצה על כפתור גורמת לחתול מיאו! לאחר מכן אתה עובר ליישומים מורכבים יותר, עוברים על כמה מדריכים, ואולי מבצעים כמה שינויים כדי להתאים אותם. המתחיל מבין חלקית את פעולתן הפנימית 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 להתחקות אחר הביצוע של אפליקציה פשוטה כלשהי, מייצג על הנייר בדיוק מה קורה בתוך המכשיר כאשר כל בלוק מתבצע. דמיינו את המשתמש מפעיל מטפל כלשהו באירועים ואז עברו והראו את ההשפעה של כל בלוק: כיצד משתנים המשתנים והמאפיינים באפליקציה? כיצד משתנים הרכיבים בממשק המשתמש? כמו קריאה מדוקדקת 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.

אתה תתחיל להבין בהדרגה שהכללים השולטים בתהליך כולו אינם מכריעים כפי שחשבת במקור.

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



איור 15-3. הגדרת הטקסט QuestionLabel בלפריט הראשון 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 מבט של משתמש, אלא פנימית, בתוך האפליקציה כאשר היא מאתחלת. אם השלמת חלק מהמדריכים, אתה בטח יודע את זה, אבל אולי לא חשבת על זה במונחים מכניים. כאשר אפליקציה מתחילה:

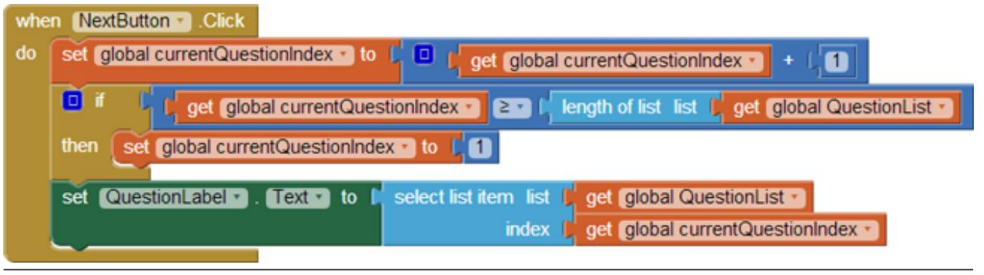
1. כל מאפייני הרכיב מוגדרים על סמך הערכים ההתחלתיים שלהם .Designer ב-Component
2. כל ההגדרות והאתחולים המשתנים מבוצעים.
3. הבלוקים במטפל האירוע Screen.Initialize מבוצעים.

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, לספוג 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

התחקות אחר אפליקציה צעד אחר צעד, על הנייר, היא אחת הדרכים להבין תכנות; זו 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 אלה מאפשרים למתכנת לבצע את הפעולות הבאות:

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

צופה במשתנים

הערכים של מאפיינים ומשתנים של רכיבים אינם גלויים כאשר אתה בודק אפליקציה .Inventor App-בטכניקת איתור באגים נפוצה היא להוסיף בלוקים להצגת 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 Watch-הכדי לבחון משתנים במהלך הפעלת אפליקציה, אבל כלי אחר בשם Do Itמאפשר לך להתנסות בנפרד

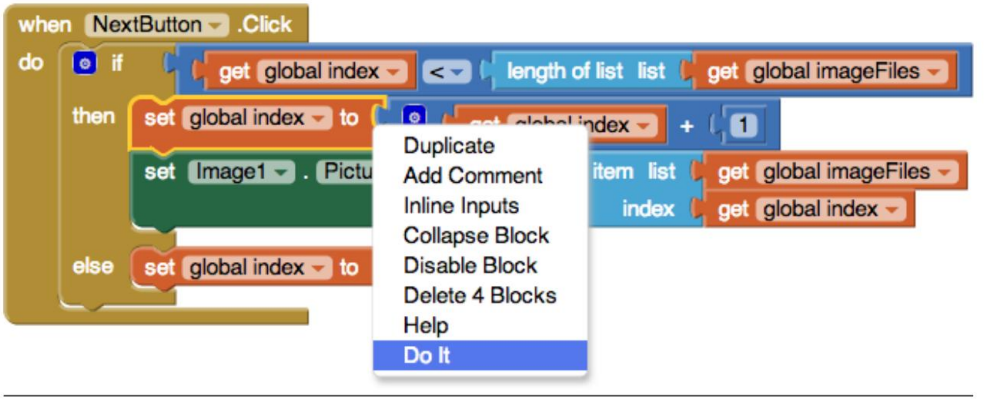
בלוקים מחוץ לרצף הביצוע הרגיל. לחץ לחיצה ימנית על כל בלוק ובחר עשה זאת; החסימה תתבצע. אם הבלוק הוא ביטוי שמחזיר ערך, App Inventorיציג את הערך הזה בתיבה מעל הבלוק.

האם זה שימושי מאוד לאיתור בעיות לוגיקה בלוקים שלך. שקול שוב את מטפל האירועים NextButton.Clickשל 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- currentQuestionIndexכדי לראות איך 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 Watch-השלך 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 Nextכדי להגיע לשאלה השנייה. לאחר מכן, במקום ללחוץ שוב על NextButtonולבצע את כל המטפל באירועים במכה אחת, תוכל להשתמש Do It-בכדי לבצע את החסימות בתוך ה- NextButton.Clickמטפל באירועים, אחד בכל פעם. תתחיל בלחיצה ימנית על שורת הבלוקים העליונה (התוספת של currentQuestionIndex) ובחירה Do 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. כשתהיו מוכנים, תוכלו לבחור את שורת הבלוקים הבאה (בדיקת אם) ולבחור Do It-בכך 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 באופן מלא מבלי לדאוג לגבי הבעייתיים שבהם. 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 App Inventor-בהוא כמה זה קל. האופי הוויזואלי שלו גורם לך להתחיל לבנות אפליקציה מיד, ואתה לא צריך לדאוג להרבה 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 and Blocks Editor Designer-לולהתחיל 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.