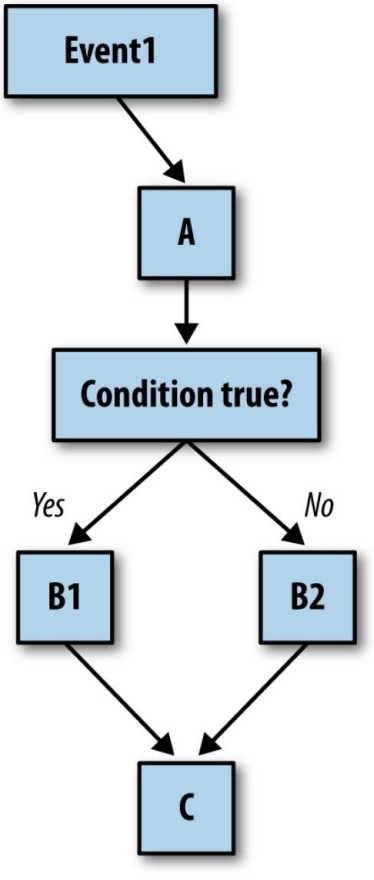
תכנות האפליקציה שלך לקבלת החלטות: בלוקים מותנים

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 המרכזי 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

איור 18-1

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.

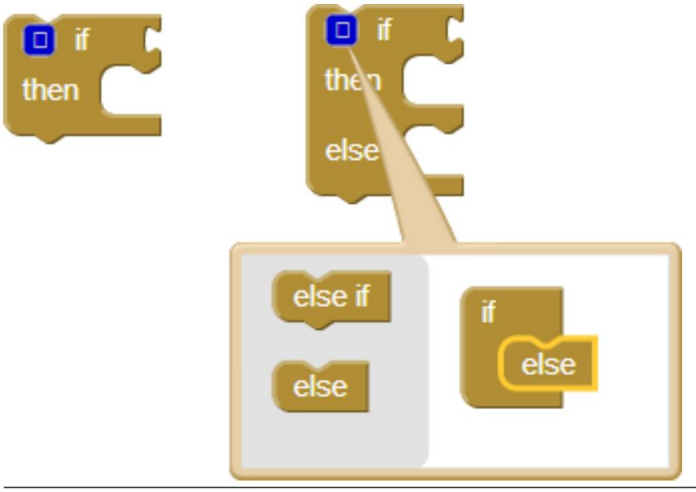
שקול את התרשים באיור .18-1כאשר מתרחש אירוע, פונקציה (בלוק) Aמבוצעת. לאחר מכן, מתבצעת מבחן החלטה. אם הבדיקה נכונה, מבוצע .B1אם הוא שקר, B2מבוצע. בכל מקרה, שאר המטפל באירועים (C)הושלם.

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

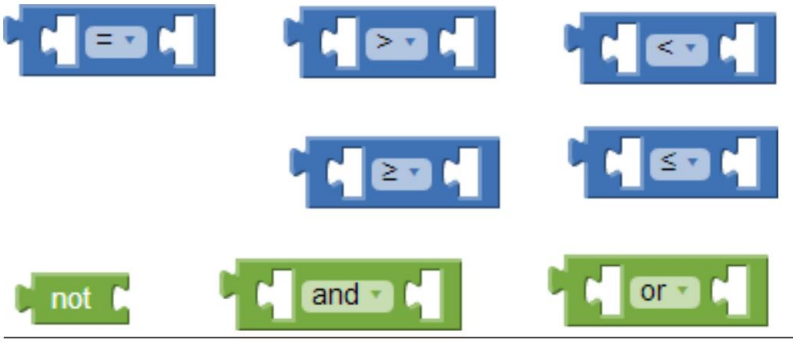
תנאי בדיקה עם if Blocks if I-else

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



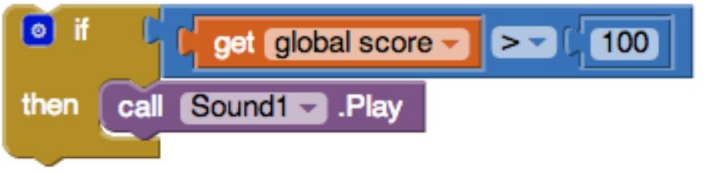
איור 18-3. חסימות אם ואחרות אם מותנות

אתה יכול לחבר כל ביטוי בוליאני לשקעי הבדיקה של בלוקי if- . else if ביטוי בוליאני הוא משוואה מתמטית שמחזירה תוצאה של נכון או לא נכון. הביטוי בודק את הערך של מאפיינים ומשתנים על ידי שימוש באופרטורים יחסיים ולוגיים כמו אלה המוצגים באיור 18-3.



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

הבלוקים שתשים בתוך שקע ה"אז" של בלוק if יבוצעו 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.

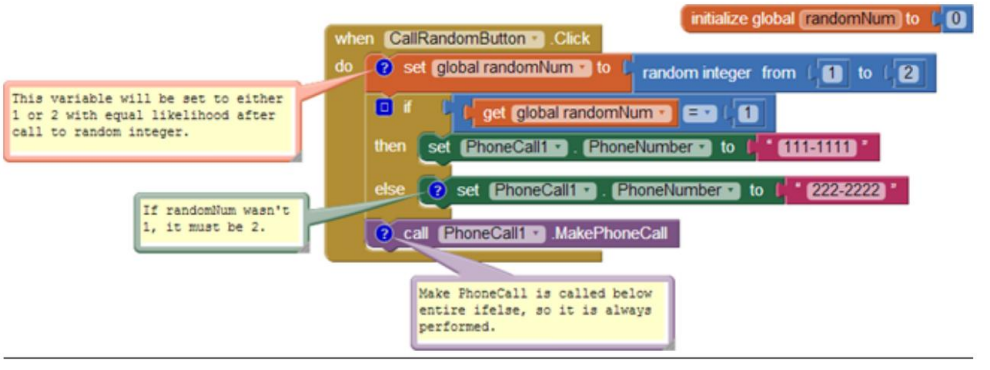


איור 18-5. ביטוי בוליאני המשמש לבדיקת הערך של הציון המשתנה

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 שווא 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, והוא מתקשר לחבר אקראי. באיור 18-5, בלוק שלם אקראי משמש ליצירת מספר אקראי ולאחר מכן בלוק if else מתקשר למספר טלפון מסוים בהתבסס על אותו מספר אקראי.



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

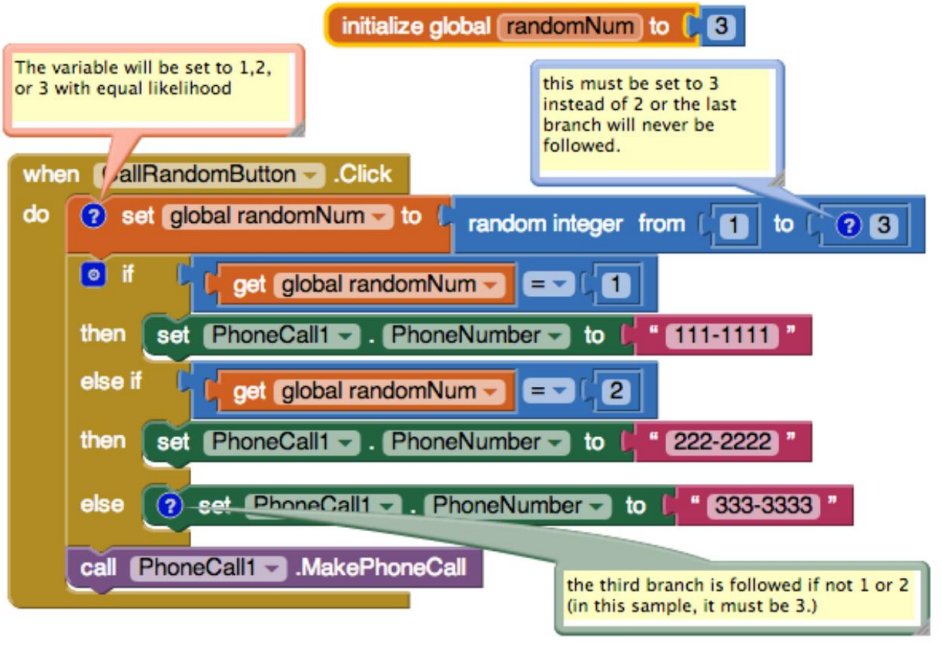
בדוגמה זו, מספר שלם אקראי נקרא עם ארגומנטים 1ו-2, כלומר המספר האקראי המוחזר יהיה 1או 2בסבירות שווה. המשתנה RandomNum מאחסן את המספר האקראי המוחזר.

לאחר הגדרת RandomNum, הבלוקים משווים אותו למספר 1במבחן . 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– .2222האפליקציה 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. לשם כך, 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- elseif מבוצע, 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.

שים 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

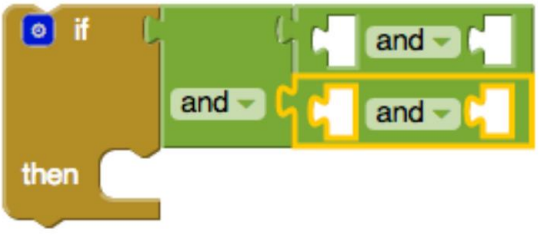
מלבד תנאי קינון, ניתן גם לציין בדיקות מותניות בודדות שהן מורכבות יותר מבדיקת שוויון פשוטה. לדוגמה, שקול אפליקציה שרוטטת כאשר הטלפון שלך (וכנראה אתה!) עוזב בניין או גבול כלשהו. אפליקציה כזו עשויה לשמש אדם על תנאי כדי להזהיר אותו כאשר הוא מתרחק מדי מגבולותיו החוקיים. הורים עשויים להשתמש בו כדי לעקוב אחר 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 אחד מהבלוקים וה- andבתוך שקע ה-"tset" של ,ifואת האחרים בתוך ,block and frst-ה כפי שמוצג באיור .18-7



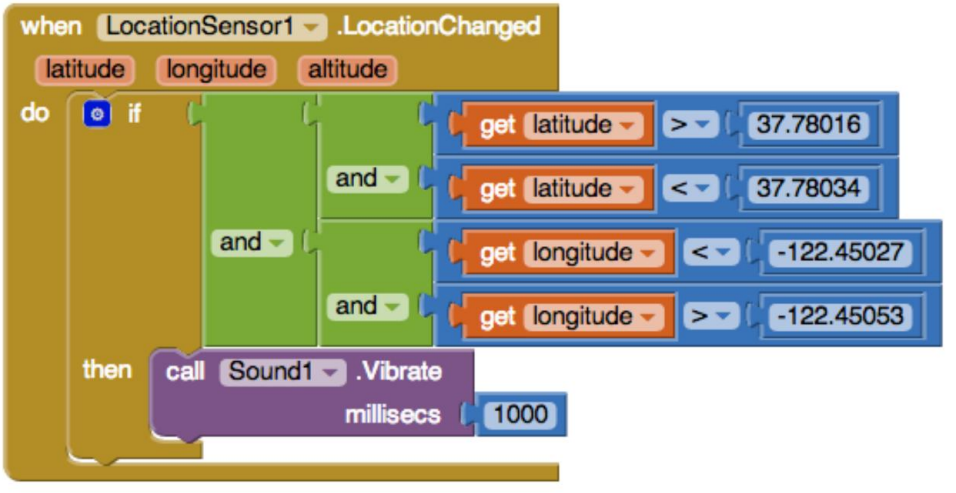
איור .18-8מבחן אם יכול לבדוק תנאים רבים באמצעות ו, או, ובלוקים יחסיים אחרים

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

איור .18-9בלוקים 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

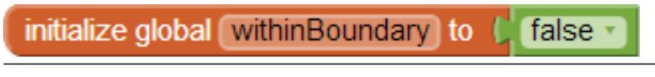
תכנות תנאים מורכבים



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

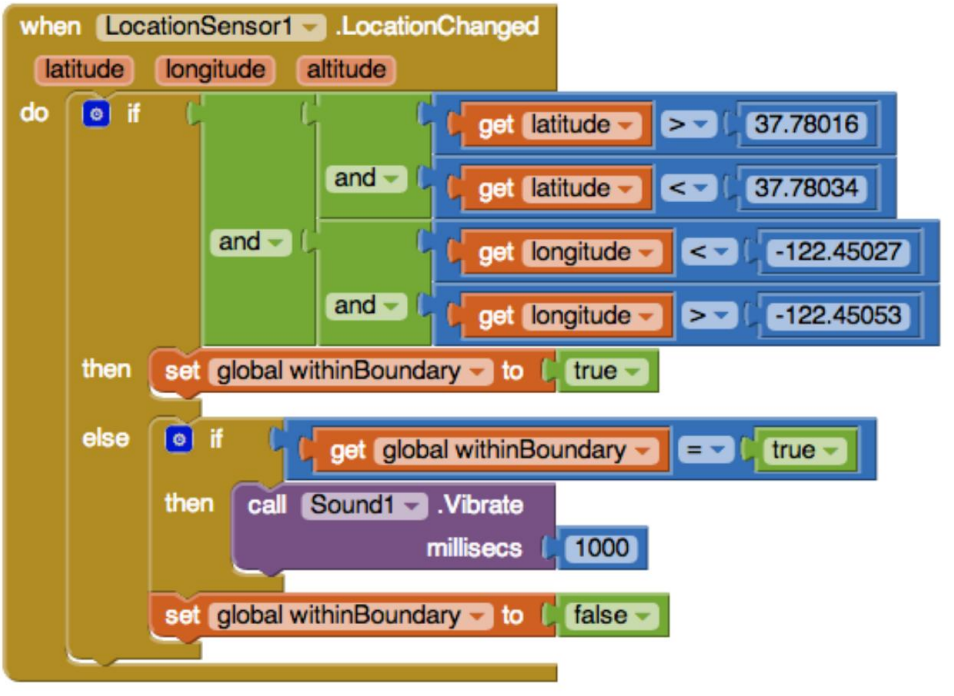
עם בלוקים אלה, בכל פעם שה- LocationSensor מקבל קריאה חדשה ושלה המיקום נמצא בתוך הגבול, הטלפון רוטט.
בסדר, עד כה זה די מגניב, אבל עכשיו בואו ננסה משהו אפילו יותר מסובך לתת לך מושג על היקף סמכויות קבלת ההחלטות של האפליקציה. מה אם היית רוצה שהטלפון ירטוט רק כשהגבול נחצה מבפנים אל חוץ? לפני שתתקדם, חשב כיצד 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. insideBoundary הוא דוגמה למשתנה בוליאני - במקום לאחסן מספר או טקסט, הוא מאחסן אמת או שקר. עבור דוגמה זו, היית מאתחל אותו ל-eslaf, כפי שמוצג באיור 18-10, כלומר המכשיר אינו נמצא במרכז המדע הרני של USF.



איור 18-11. insideBoundary מאותחל ל-eslaf

כעת ניתן לשנות את הבלוקים כך שהמשתנה insideBoundary מוגדר על כל אחד מהם שינוי מיקום, וכך הטלפון רוטט רק כשהוא זז מבפנים אל מחוץ לגבול. אם לנסח זאת במונחים שאנו יכולים להשתמש בהם 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 18-11 מציג את הבלוקים המעודכנים.



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

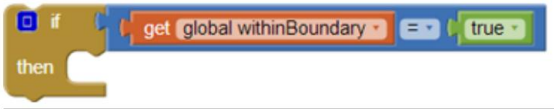
הבה נבחן את הבלוקים הללו יותר מקרוב. כאשר חיישן המיקום מקבל קריאה, LocationSensor מגדירה את המשתנה insideBoundary ל-true. כמיוון שאנו רוצים שהטלפון ירטוט רק כשאנחנו מחוץ לגבול, לא מתרחשת רטט בסניף הראשון הזה.

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 -אתה יכול לחשוב מה זה? אנחנו צריכים גם לאפס בתוך הגבול ל-eslaf כדי שהטלפון לא ירטוט שוב בקריאה הבאה של החיישן.

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

תכנות תנאים מורכבים



איור 18-13. האם אתה יכול לדעת אם שני המבחנים האלה 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 את הצד ההגיוני של המוח שלך לא מעט, אבל זה גם יכול להיות כיף גדול.