פיתוח תוכנה מתקדם 2

אבני דרך 4,5 בפרויקט

ד"ר אליהו חלסצ'י
eliahukh@colman.ac.il

רקע

פרויקט זה בקורס פת"מ הוא חלון הראווה שלכם כשתרצו להציג את הניסיון התכנותי שצברתם. פרויקט זה מכיל את האלמנטים הבאים:

- שימוש בתבניות עיצוב וארכיטקטורה
- תקשורת וארכיטקטורת שרת-לקוח
- שימוש במבני נתונים ובמסד נתונים
- הזרמת נתונים (קבצים ותקשורת)
- השוואה, בחירה והטמעה של אלגוריתמים בתוך המערכת שניצור
- תכנות מקבילי באמצעות ת'רדים
- תכנות מוכוון אירועים, אפליקציית desktop עם GUI
- תכנות מוכוון אירועים, אפליקציית Web

בסמסטר זה (פת"מ 2) עליכם להגיש 2 אבני דרך, עבור צד הלקוח בפרויקט שלנו.

1. כתיבת מפרש קוד (interpreter) לשפת תכנות חדשה השולטת במל"ט.
2. אפליקציית דסקטופ לחישוב מסלול הטיסה והטסת המטוס.

לפרויקט קיימת בדיקה אוטומטית. כל הפרטים כגון אופן ותאריכי ההגשה ימצאו באתר הקורס.

בהצלחה!

אבן דרך 4 – מפרש קוד

היכרות עם סימולטור הטיסה

ברצוננו לכתוב מפרש לקוד שליטה במל"ט (מטוס ללא טייס). המטוס שלנו יטוס במרחב הווירטואלי של סימולטור הטיסה FlightGear. את סימולטור הטיסה תוכלו להוריד מ http://home.flightgear.org/ .

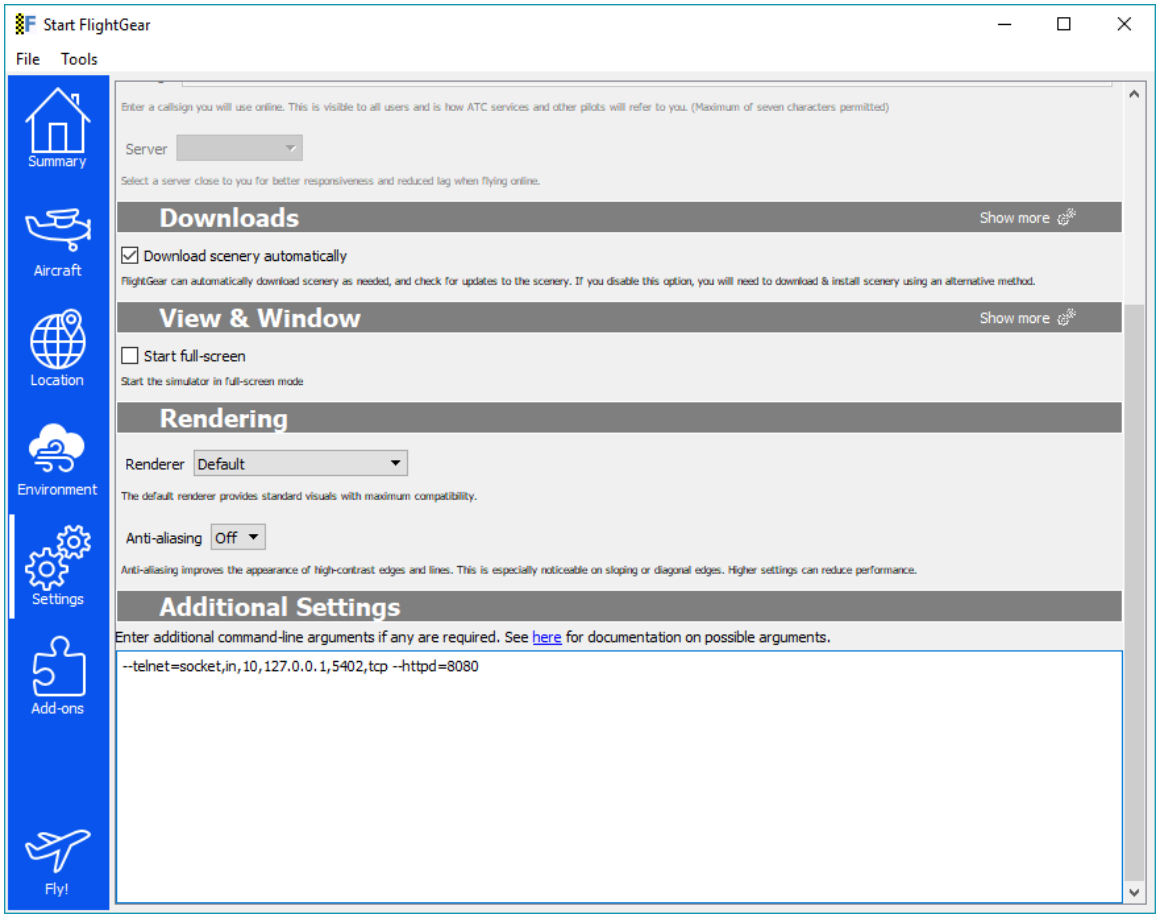
בין היתר, סימולטור זה מהווה גם שרת שאפשר להתחבר אליו כלקוח (ולהיפך). כך נוכל בקלות לשלוף מידע אודות הפרמטרים השונים של הטיסה בזמן אמת ואף להזריק לו פקודות שינהגו את המטוס.

במסך הפתיחה ניתן לגשת ל settings ולהוסיף הגדרות שבד"כ נכתבות ב command line בסעיף של Additional Settings (ראו צילום מסך בהמשך).

למשל ההגדרה --telnet=socket,in,10,127.0.0.1,5402,tcp

אומרת לסימולטור לפתוח ברקע שרת שניתן להתחבר אליו באמצעות כל telnet client. השרת מבוסס על socket, והוא נועד לקרוא מידע שיגיע מהלקוח (in) בקצב של 10 פעמים בשנייה, ב local host כלומר באותו המחשב (ip 127.0.0.1) על port 5402 מעל פרוטוקול tcp/ip.

למשל ההגדרה –httpd=8080 תפתח web server על פורט 8080.



הריצו את הסימולטור עם ההגדרות לעיל (Fly!).

לאחר שהסימולטור פעל, פיתחו את הדפדפן בכתובת http://localhost:8080 ותוכלו לראות את האפליקציה ה web-ית שבאה עם הסימולטור.

כעת, תפתחו telnet client בשורת הפקודה, על local host ופורט 5402 (בהתאם להגדרות שראינו בסימולטור)

הערה: בלינוקס ניתן ישר לפתוח מהטרמינל, בחלונות יש לוודא שהתקנתם telnet client, מי שלא התקין וצריך עזרה יכול לחפש בגוגל "install telnet client windows 10 command line " ולמצוא מדריכים.

כאמור, ב CMD של חלונות הקלידו telnet 127.0.0.1 5402 ובעצם התחברתם כלקוח לשרת שפתח הסימולטור על המחשב שלכם. הממשק הזה נוח מאד מכיוון שהוא בנוי בצורה של file system (קבצים ותיקיות). כתבו ls כדי לראות את "התיקיות והקבצים" במיקום הנוכחי שלכם. תוכלו להיכנס לתיקיה באמצעות הפקודה cd (שזה change directory) למשל cd controls. תטיילו קצת בין אינספור ההגדרות השונות כדי לקבל תחושה מה יש שם.

כעת הביטו בו זמנית בסימולטור וב telnet. בסימולטור אתם יכולים לשנות זוויות צפייה ע"י לחיצה על V ואף משחק עם העכבר. הביטו על המטוס ממבט אחורי. כעת בואו נזין פקודה להזזת מייצב הכיוון של המטוס (rudder):

לא משנה היכן אתם ב telnet, כתבו:

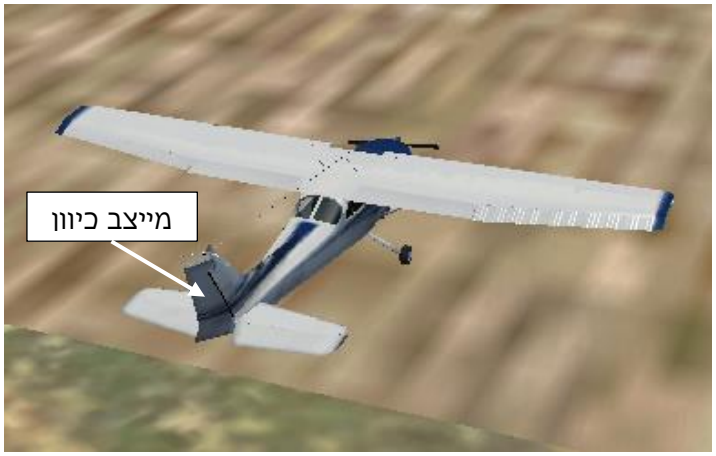
set controls/flight/rudder 1

ראו בסימולטור כיצד מייצב הכיוון זז עד הסוף ימינה.

באופן דומה כתבו

set controls/flight/rudder -1

וראו כיצד מייצב הכיוון של המטוס זז עד הסוף שמאלה. כל ערך בין 1- ל 1 יסובב את מייצב הכיוון בהתאמה.



חוץ מלתת פקודות להגאים של המטוס, ניתן גם לדגום את הערכים השונים שנמדדו ע"י מכשירי הטיסה, כמו כיוון, מהירות, גובה וכו'. כתבו ב telnet את השורה הבאה:

```
get /instrumentation/altimeter/indicated-altitude-ft
```

ניתן לראות בתגובת השרת את הערך של הגובה הנוכחי כפי שנמדד במכשיר טיסה שנקרא `altimeter`.

אבל, את הערכים של הטיסה אנו נדגום בצורה מחוכמת יותר. יחד עם ההפעלה של הסימולטור נוסיף את ההגדרה:

```
--generic=socket,out,10,127.0.0.1,5400,tcp,generic_small
```

משמעותה היא שהפעם הסימולטור יתחבר כלקוח לשרת (שאנו נבנה) באמצעות socket, לצורך פלט (out) בתדירות של 10 פעמים בשנייה, על ה local host בפורט 5400 מעל tcp/ip. הערכים שנדגום מוגדרים בקובץ שנקרא generic_small.xml המצורף כנספח לפרויקט. את הקובץ הזה עליכם לשתול במיקום בו התקנתם את FlightGear בתיקייה data/protocol. פתחו את קובץ ה XML כדי להתרשם אלו נתונים נדגום.

עם הגדרה זו נצטרך לפתוח את השרת שלנו **לפני** שנפתח את הסימולטור כדי שהוא יוכל להתחבר אלינו כלקוח. הסימולטור ישלח 10 פעמים בשנייה את הערכים שדגם מופרדים בפסיק (בדיוק כמו ב CSV) ולפי הסדר שהוגדרו ב XML. בהמשך אבן הדרך תכתבו שרת קטנטן שיאזין לערכים האלו.

מי שרוצה לקבל עוד קצת רקע לגבי שליטה במטוס יוכל לקרוא (בוויקפדיה למשל) אודות

- ההגאים: aileron, elevators, rudder (מייצב כיוון, מייצב גובה, מאזנות בהתאמה)
- וכיצד הם משפיעים בעת טיסה על ה roll, pitch, yaw בהתאמה.
  - ובעברית סבסוב, עלרוד, גלגול בהתאמה.

אין כל חובה לדעת להטיס מטוס בפרויקט זה, אך מעט רקע בהחלט יכול לעזור.

מפרש קוד לשליטה בטיסה

כאמור, ברצוננו לכתוב מפרש לשפת תכנות חדשה שמטרתה להטיס את המטוס שבסימולטור. נתחיל מלהגדיר קוד לדוגמא שמטרתו לגרום למטוס להמריא בצורה ישרה. בהמשך נסביר את המשמעות של שורות אלו וכיצד נכתוב מפרש שיריץ אותן.

קוד לדוגמא:

```
1.  openDataServer 5400 10
2.  connect 127.0.0.1 5402
3.  var breaks = bind "/controls/flight/speedbrake"
4.  var throttle = bind " /controls/engines/current-engine/throttle"
5.  var heading = bind "/instrumentation/heading-indicator/offset-deg"
6.  var airspeed = bind "/instrumentation/airspeed-indicator/indicated-speed-kt"
7.  var roll = bind "/instrumentation/attitude-indicator/indicated-roll-deg"
8.  var pitch = bind "/instrumentation/attitude-indicator/internal-pitch-deg"
9.  var rudder = bind "/controls/flight/rudder"
10. var aileron = bind "/controls/flight/aileron"
11. var elevator = bind "/controls/flight/elevator"
12. var alt = bind "/instrumentation/altimeter/indicated-altitude-ft"
13. breaks = 0
14. throttle = 1
15. var h0 = heading
16. while alt < 1000 {
17.    rudder = (h0 – heading)/20
18.    aileron = - roll / 70
19.    elevator = pitch / 50
20.    print alt
21.    sleep 250
22. }
23. print "done"
```

הסבר:

נרצה ששורה 1 תגרום לפתיחה של ת'רד ברקע, שפותח **שרת** המאזין על פורט 5400 וקורא שורה שורה בקצב של 10 פעמים בשנייה. את הערכים הנדגמים יש לאכסן במבנה נתונים שבאמצעותו נוכל לשלוף ב O(1) את הערך העדכני של משתנה כלשהו שבחרנו.

נרצה ששורה 2 תתחבר כלקוח לשרת שנמצא ב 127.0.0.1 ומאזין על פורט 5402.

בשורות 3-12 אנו מגדירים את המשתנים שאיתם נעבוד במהלך התוכנית. המשמעות של bind היא כריכה בין ערך המשתנה בתוכנית שלנו לבין מיקומו בסימולטור הטיסה.

כך למשל בשורה 14 כשביצענו השמה throttle = 1 שלחנו למעשה לסימולטור את הפקודה:

```
set /controls/engines/engine/throttle 1
```

וגרמנו למצערת להיפתח עד הסוף (כוח מלא למנוע כדי שהמטוס יתחיל לנוע)

נשים לב שהכריכה היא דו-כיוונית, למשל המשתנה `heading` כרוך למכשיר `heading indicator` שנמצא ב:

```
/instrumentation/heading-indicator/offset-deg
```

בכל פעם בתוכנית שנשתמש ב `heading` נקבל את הערך הנוכחי של הכיוון מסימולטור הטיסה. למשל, בשורה 15 המשתנה `h0` מקבל את ערכו הנוכחי של `heading` - כיוון הטיסה, כפי שנגדם באותו הרגע ע"י ת'רד שרץ ברקע שפתחנו בשורה 1.

בשורה 16 פתחנו לולאת while כל עוד המשתנה alt (הכרוך לגובה הטיסה כפי שנמדד ע"י ה altimeter) קטן מ 1000 רגל. כאמור הערך של alt מתעדכן באופן אוטומטי בזכות הכריכה שהגדרנו ובאמצעות הת'רד שפתחנו בשורה 1.

בתוך הלולאה אנו מעדכנים את הערכים

- של ה rudder כפונקציה של הכיוון
- של ה aileron כפונקציה של הגלגול
- ושל ה elevator כפונקציה של ה pitch

הערה: ערכים אלו ניתנו עבור המטוס הדיפולטיבי בסימולטור – Cessna C172p ובהחלט יכולים להיות שינויים בין גרסאות שונות כדי שהמטוס באמת יתייצב בהמראה.

כמו כן, בכל איטרציה אנו מדפיסים את הערך של alt וממתינים 250 מילישניות לפני המעבר לאיטרציה הבאה. בסוף הרוטינה אנו כותבים done.

אז איך ניגשים למשימה מפלצתית שכזו? ☺

תחילה נכיר תבנית עיצוב חשובה ופשוטה בשם Command Pattern. התבנית אומרת לנו להגדיר ממשק בשם Command עם מתודה doCommand()‎. כל פקודה במערכת שלנו (אצלנו זה פקודות שיש לפרש) תהיה מחלקה מהסוג של Command. כך Command פולימורפי יכול להיות פקודה ספציפית כלשהי, ונפעיל את כולן באותו האופן. לצרכים שלנו doCommand יכולה לקבל כפרמטר מערך של מחרוזות שיש לפרש.

הטריק התכנותי שנבצע הוא שנכניס את כל הפקודות למפה מבוססת hash כך שהמפתח הוא מחרוזת, והערך הוא אובייקט ספציפי מסוג Command. כך בהינתן המחרוזת נוכל לשלוף מידית את ה Command שצריך לפעול.

מנגנון העבודה:

צרו פונקציה בשם lexer שתפקידה לקרוא את הסקריפט שצריך לפרש (שורה בודדת מה console או קובץ שלם של פקודות) והיא תחזיר מערך של מחרוזות. כל מחרוזת היא מילה בתוכנית שיש לפרש.

כעת, כתבו פונקציה בשם parser שעוברת (כמעט) על כל מחרוזת במערך שיצר ה lexer. בהינתן מחרוזת, היא תשמש אותנו כמפתח שבאמצעותו נשלוף את אובייקט הפקודה המתאים מהמפה, ונזין לו את המחרוזות שהוא צריך כדי שיפרש את הפקודה ויבצע אותה.

לדוגמא:

המחרוזת הראשונה בתוכנית היא openDataServer. מפתח זה יגרום לשליפה של אובייקט שמימש את הממשק Command. נניח שקוראים לו OpenServerCommand. למתודה doCommand שלו נזין את שארית השורה כמערך של מחרוזות. בתורה, doCommand תודא שגודל המערך הוא 2 (שני פרמטרים בסקריפט) ושהמחרוזות מהוות ערך מספרי תקין. אחרת, נדפיס הודעת שגיאת סינטקס מתאימה. אם הערכים תקינים נפעיל את השרת שלנו ברקע על פרמטרים אלה. כמובן, כדאי שהשרת הזה יוגדר במחלקה אחרת, קראו לה DataReaderServer.

כעת, ה parser יזין את המחרוזת הבאה שיש לפרש (connect) ישלוף אובייקט פקודה מתאים, יריץ אותו, וחוזר חלילה עד לסוף הסקריפט.

<u>הגדרת משתנים:</u>

כפי שניתן לראות המילה var היא הטריגר להגדרת משתנים. הפקודה המתאימה צריכה לתחזק מפה מבוססת hash שבו המפתח הוא שם המשתנה והערך (מסוג double) הוא ערך המשתנה. תקראו למפה הזו symbolTable.

המשמעות של bind היא כפולה, מצד אחד יש לגרום לכך שאם המשתנה נדגם, כלומר מצד ימין של אופרטור ההשמה, אז ערכו יישלף מהמפה שיצר הת'רד שקיבל את הנתונים מהסימולטור (השרת הקטן שלנו). לעומת זאת אם המשתנה נמצא מצד שמאל של אופרטור ההשמה אז עלינו לשלוח לסימולטור כלקוח את הפקודה set למיקום המתאים עם הערך שנמצא מימין לאופרטור ההשמה. כפי שאתם מבינים אופרטור ההשמה "=" גם הוא אובייקט מסוג Command.

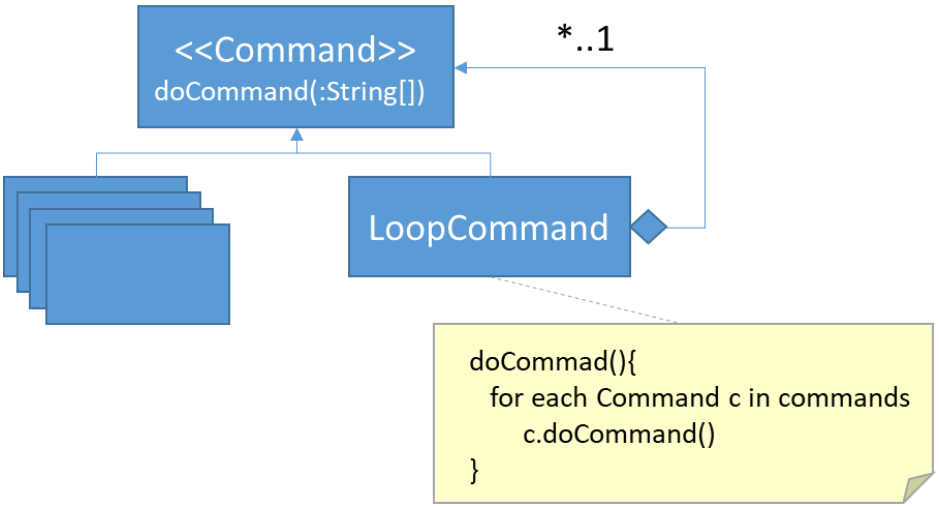
תנאים:

בשורה 16 המפתח הוא while לאחר מכן נצפה לסדרה של תנאים עד להופעת הסימן "}". לשם הקלה עליכם לצפות רק לתנאי אחד. אבל מי שרוצה להשקיע שיכניס גם סוגרים עגולים ופעולות AND ו OR.

התנאי יכול להיות מורכב מהאופרטורים <,>,<=,>=, ==,=! במשמעות הרגיל שאתם כבר מכירים.

לולאה:

היופי בתבנית העיצוב של Command זה שברגע שכל פקודה מוגדרת במחלקה, אין לנו בעיה ליצור פקודה שמורכבת מכמה פקודות בסיסיות יותר:



למשל, המחלקה LoopCommand יכולה להחזיק מערך דינאמי (ופולימורפי) של אובייקטי Command. כל אחד מהם יכול להיות אובייקט ספציפי של פקודה כלשהי או אפילו עוד אובייקט מהסוג של LoopCommand, כלומר עוד לולאה פנימית בסקריפט. המתודה doCommand של LoopCommand פשוט תפעיל את כל אובייקטי ה Command שנזין לה כל עוד התנאי מתקיים. נזין את כל אובייקטי ה Command שחזרו מהשורות ש"פירסרנו" עד להופעה של "}". בדוגמא לעיל מדובר בשורות 17 עד 21.

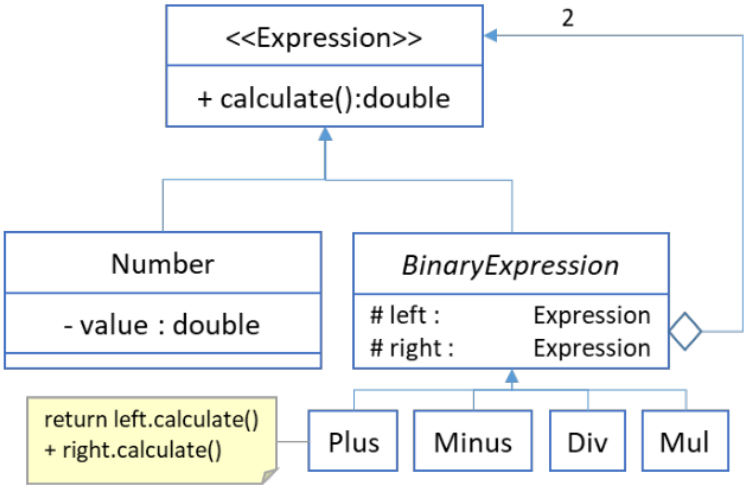
ביטויים:

ומה לגבי פיענוח ביטויים מתמטיים מורכבים כמו בשורות 17-19?

נשים לב שכל ביטוי יכול להיות ערך (קבוע או משתנה) או אופרטור אונרי הפועל על ביטוי, או אופרטור בינארי הפועל על שני ביטויים. כמובן שבתורם ביטויים אלה יכולים להיות שוב ערכים או אופרטורים... כלומר, מתקבל עץ של ביטויים בו כל קודקוד הוא או עלה (ערך בודד), או אופרטור עם בנים שהם קודקודים (עלים או אופרטורים). שוב מדובר בפולימורפיזם, יש לנו כמה סוגים של קודקודים, או ביטויים, כאשר הילדים של אופרטור כלשהו גם הם בעצמם ביטויים. כדי להתמודד עם זה אנו זקוקים לתבנית עיצוב בשם, איך לא, Interpreter.

הנה דוגמא:

ממשק בשם Expression מגדיר מתודה בשם Calculate המחזירה double. אובייקט מסוג Expression יכול להיות Number או BinaryExpression שלו יש שני משתנים בדיוק מהסוג של Expression: בשמות left, right. כמו כן ירשו אותו פעולות החישוב הבסיסיות כמו פלוס, מינוס, חילוק וכפל. לדוגמא, Plus יחזיר את left.calculate() + right.calculate() יהיה הביטוי שלהם עמוק כאשר יהיה...

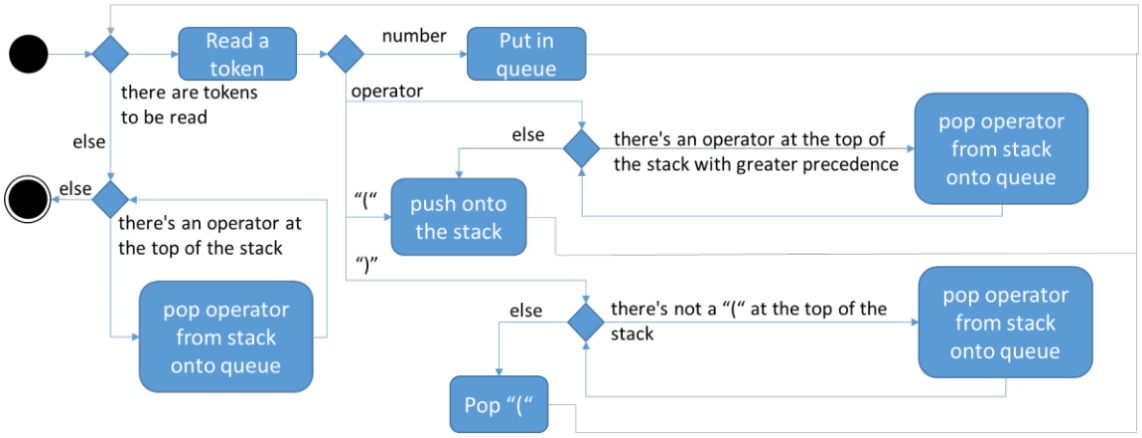


דוגמא לחישוב:

```
// 3+(4/2)*5
Expression* e=new Plus(new Number(3) , new Mul( new Div(new Number(4), new
Number(2)) , new Number(5)));
e->calculate();
```

איך מגיעים מביטוי בצורה של infix כמו 3+4 ליצירה של אובייקטים כמו Plus(Number(3),Number(4))?

לשם כך עליכם לממש אלגוריתם חביב בשם Shunting-yard של דייקסטרה. הנה Activity Diagram שמתארת את פעולתו:



בהינתן ביטוי infix , האלגוריתם מסדר את המספרים בתור, ומשתמש במחסנית כדי להכניס את האופרטורים לתור זה בסדר שמציג את הביטוי כ postfix. למשל עבור הדוגמא לעיל בסוף האלג' התור יראה כך: 342/5*+. כשנקרא את התור הפוך (כלומר, מימין לשמאל) נבין שעלינו לבצע חיבור של (הכפלה של 5 עם (חלוקה של 4 ב 2) עם 3. לפיכך, נוכל לייצר בהתאם לביטויים את המופעים של Number, Plus, Minus, Mul, Div ולחשב את תוצאות הביטוי.

:Main

למעשה פונקציית ה `main` צריכה לקרוא מהמשתמש שורה שורה, ועל כל שורה לשלוח ל `lexer` ואת הפלט שלו לשלוח ל `parser`. כך הקוד שהמשתמש כתב מתפרש וגורם לפעולות שונות בסימולטור.

אופציה שנייה, היא לפתוח קובץ טקסט ובתוכו הסקריפט המלא שאותו נפרש שורה שורה בדיוק באותו האופן.

למשקיעים: ה `main` תמיד תצפה לקלט של המשתמש, אך אחת הפקודות תהיה להריץ סקריפט שכתוב בקובץ מסוים. למשל:

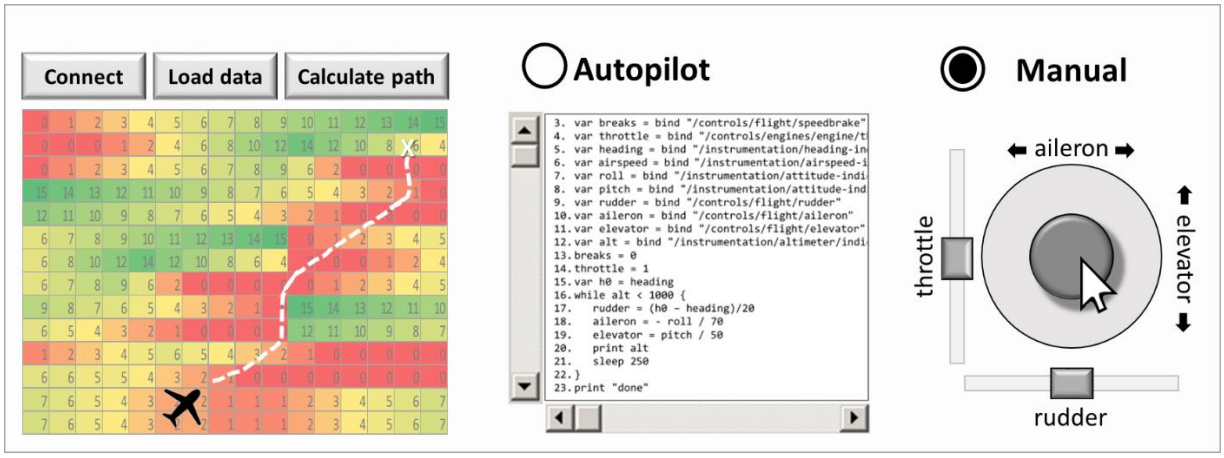
```
> run “d:/scripts/takeoff.fgs”
```

בהצלחה!

אבן דרך 5 – אפיליקציית desktop

באבן דרך זו עליכם לבנות אפליקציית דסקטופ בארכיטקטורת MVVM עם JavaFx.

התצוגה תכיל את האלמנטים הבאים:



נתאר את האפליקציה משמאל לימין:

- כפתור ה connect יתחבר לסימולטור הטיסה. בחלון popup נקלוט את ה IP וה port. האפליקציה תתחבר לסימולטור **כלקוח**.
- כפתור ה load data יפתח דיאלוג לפתיחת קובץ.
  - נפתח קובץ CSV המכיל נ.צ (נקודת ציון) של תא 0,0, גודל תא בקילומטר מרובע, וטבלת נתוני גובה של כל תא (במטרים). קובץ לדוגמה יינתן בהמשך.
  - מתחת תוצג מפה של הנתונים שנטענו. ככל שהגובה נמוך יותר התא יקבל צבע אדום יותר וככל שהגובה גבוה יותר צבע ירוק.
  - נדגום את מיקום המטוס בסימולטור בקצב של 4 פעמים בשנייה ונציג אייקון של המטוס בתא המתאים.
  - לחיצה על תא כלשהו תקבע יעד עבור המטוס שיסומן ב X.
- לחיצה על כפתור ה calculate path תציג חלון popup ובו קלט ה IP וה port של השרת שכתבתם סמסטר קודם הפותר בעיות חיפוש.
  - נתחבר לשרת שיחזיר לנו את המסלול הזול ביותר – ואותו נציג על המפה.

- מעתה, על בחירה של יעד חדש תגרום לחישוב מסלול מחדש ללא צורך בלחיצה על הכפתור calculate path.

- **המפה הצבעונית והפונקציונאליות שלה צריכים להיות מגולמים בפקד משתמש אחד.**
- בונוס 10 נקודות (רמת קושי גבוהה): פקודות הטסה ישלחו מהאפליקציה לסימולטור כך שהמטוס יטוס על פי המסלול שנקבע בגובה של X מטרים מעל לפני השטח.
- כפתור רדיו autopilot יגרום לאינטרפרטציה של הטקסט שנכתב ב textbox מתחתיו.
  - רצוי להוסיף גם כפתור load שיפתח דיאלוג לפתיחת קובץ טקסט שכזה
  - אין צורך לממש מחדש interpreter. חישבו כיצד הכי נכון לבצע reuse לזה שכתבתם סמסטר קודם.
- כפתור רדיו manual יעביר את השליטה לג'ויסטיק וירטואלי
  - הג'ויסטיק מצויר כעיגול בתוך מעגל בדומה לתמונה לעיל.
  - באמצעות העכבר נוכל לגרור את העיגול במגבלות המעגל
    - תנועה ימינה-שמאלה תשלוט על תנועת ה aileron בערכים 1..1- בהתאמה
    - תנועה למעלה-למטה תשלוט על תנועת ה elevator בערכים 1-..1 בהתאמה
    - שחרור העכבר יוביל את העיגול לאמצע (הכל 0)
  - סליידר אנכי ישלוט על המצערת: למעלה -1 למטה 0
  - סליידר אופקי ישלוט על ה rudder: ימין 1 שמאל 1-
  - כל הג'ויסטיק והפונקציונאליות שלו צריכים להיות מגולמים בפקד משתמש אחד.

חשוב מאד לשמור על עקרונות ה MVVM. יש להפריד בין המודל ל view באמצעות view model ולעשות שימוש ב data binding עד כמה שניתן.

בהצלחה!

תוך כדי סמסטר אפרסם לכם את פרטי ההגשה והבדיקה האוטומטית.