

# תוכנה 1

## תרגיל מספר 6

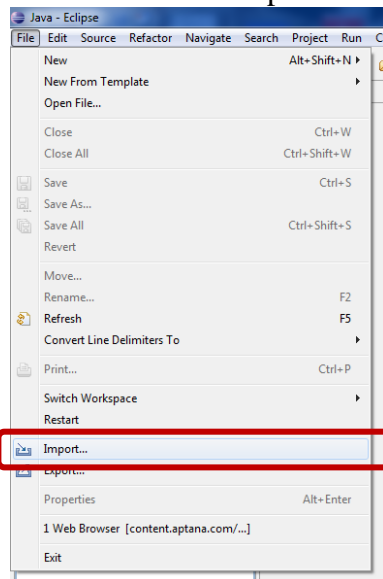
### הנחיות כלליות:

- קראו בעיון את קובץ נוהלי הגשת התרגילים אשר נמצא באתר הקורס.
- הגשת התרגיל תעשה במערכת ה VirtualTAU בלבד (<http://virtual2002.tau.ac.il/>).
- יש להגיש קובץ zip יחיד הנושא את שם המשתמש (לדוגמא, עבור המשתמש zvainer יקרא הקובץ zvainer.zip) קובץ ה zip יכיל:
  - א. קובץ פרטים אישיים בשם details.txt המכיל את שמכם ומספר ת.ז. הזהות שלכם.
  - ב. קבצי ה java של התוכניות אותם התבקשתם לממש.
  - ג. קובץ טקסט עם העתק של כל קבצי ה java
  - ד. קובץ טקסט בשם answers עם התשובות לשאלות

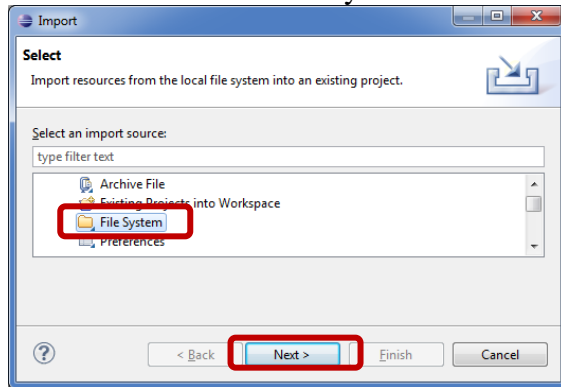
חשוב! בתרגיל זה עליכם להשלים קוד בפרוייקט אותו אנו מספקים.

לשם כך, הורידו את הקובץ BoidsApplication.zip מאתר הקורס. פתחו את תוכנו לתוך ספרייה מקומית. לאחר מכן, יבאו את הפרוייקט ל-Eclipse בעזרת הפקודות הבאות:

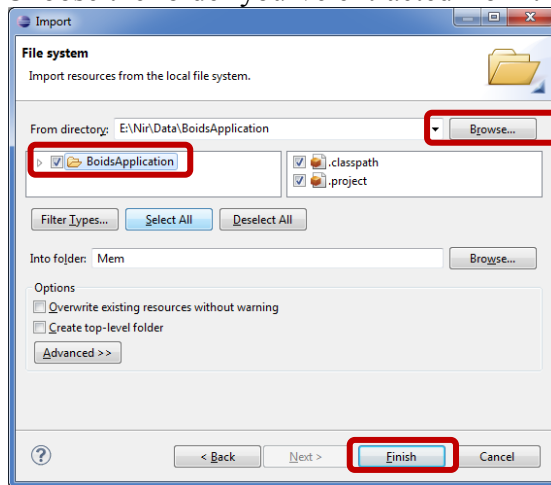
1. Create a new project in Eclipse
2. Click on “File -> Import...”



3. Choose “General -> File System” and then click “Next”



4. Choose the folder you've extracted from the ZIP file.



5. Click “Finish” and overwrite files if prompted to do so.

עליכם להגיש את הקוד הסופי. כלומר, לאחר מימוש כל השאלות (מימוש מחלקה חדשה בחלק א' והשלמת הפונקציות הנדרשות בחלקים ב' וג'), יש לכווץ את כל הפרוייקט לתוך קובץ ZIP ולהעלות כפתרון ל-VirtualTAU.

ניתן להוסיף פונקציות עזר, אולם אסור לשנות את הגדרות הפונקציות המופיעות בשאלות.

## חלק א':

מטרת חלק זה היא לממש ספרייה עבור פעולות ווקטוריות בסיסיות. הספרייה תשמש אותנו בהמשך התרגיל. שימו לב, יש לממש את כל הפונקציות הנתונות בחלק זה ללא שימוש בספריות נוספות או בשירותי ספרייה של Java, מלבד שירותים מתמטיים (במחלקה Math).

נבנה את המחלקה Point, המייצגת וקטור ו/או נקודה דו-מימדיים. מכיוון שנשתמש במחלקה זו בהמשך יש ליצור אותה כחלק מהחבילה boids. בתרגיל זה נתייחס למונחים ווקטור ונקודה כמונחים שקולים. כאשר נדבר על וקטור, נבין כי המחלקה מייצגת כוון ומרחק מראשית הצירים. כאשר נדבר על נקודה או מיקום, המחלקה תציין מיקום זה.

מידע נוסף אודות אריתמטיקה ווקטורית ניתן למצוא בכתובת:

[http://en.wikipedia.org/wiki/Euclidean\\_vector](http://en.wikipedia.org/wiki/Euclidean_vector)

- יש לממש בנאים למחלקה Point. הבנאים יאפשרו ייצור עצם חדש בהינתן קואורדינטות התחלה. הבנאי הסטנדרטי ייצור נקודה בראשית הצירים.

דוגמאות:

```
Point p1 = new Point();
System.out.println(p1);
> (x=0.00, y=0.00)

Point p2 = new Point(100, 50);
System.out.println(p2);
> (x=100.00, y=50.00)
```

- יש לממש שאילתות והשמות בסיסיות למחלקה Point.

להלן כמה דוגמאות:

```
p1.setX(10);
p2.setY(50);
System.out.println(p1);
> (x=10.00, y=50.00)

System.out.println(p1.getX());
> 10.0
System.out.println(p1.getY());
> 50.0
```

נדרש לממש את הפונקציות הבאות :

```
public double getX() {  
}  
  
public double getY() {  
}  
  
public void setX(double x) {  
}  
  
public void setY(double y) {  
}
```

בנוסף, לנוחיותנו, נממש פונקציות להזזת כל מימד בנפרד :

```
public void addX(double x) {  
}  
  
public void addY(double y) {  
}
```

3. כעת, נממש פעולות מתקדמות יותר עבור המחלקה Point.

א. ממשו את הפונקציה norm() המחשבת את הנורמה האוקלידית של הווקטור הנתון.

דוגמאות :

```
Point p = new Point(1,2);  
System.out.println(p);  
> (x=1.00, y=2.00)  
System.out.println(p.norm());  
> 2.23606797749979
```

חתימת הפונקציה :

```
public double norm() {  
}
```

ב. ממשו את הפונקציה distance המחשבת את המרחק האוקלידי בין שתי נקודות :

```
Point p = new Point(1,2);  
Point q = new Point(3,5);  
System.out.println(p);  
> (x=1.00, y=2.00)  
System.out.println(q);  
> (x=3.00, y=5.00)  
System.out.println(p.distance(q)); (x=1.00, y=2.00)  
> 3.605551275463989
```

חתימת הפונקציה :

```
public double distance(Point other) {  
}
```

4. נממש פעולות בין שתי נקודות שונות. כל פעולה תמומש בשני אופנים, האחד יחזיר אובייקט חדש המייצג את הפעולה הנתונה (בדומה לפעולת האופרטור +) ואילו השני ישנה את האובייקט עליו נקרא השירות (בדומה לפעולת האופרטור +=).

א. ממשו את הפעולות add ו-plus המבצעות חיבור ווקטורי.

```
/*
 * @pre other != null
 * @post x == $prev(getX()) + other.getX()
 * @post y == $prev(getY()) + other.getY()
 */
public void add(Point other) {
}

/*
 * @pre other != null
 * @post $ret.getX() == getX() + other.getX()
 * @post $ret.getY() == getY() + other.getY()
 * @post $ret != this
 */
public Point plus(Point other) {
}
```

ב. ממשו את הפעולות subtract ו-minus המבצעות חיסור ווקטורי. השלימו את חוזה הפונקציות:

```
public void subtract(Point other) {
}

public Point minus(Point other) {
}
```

ג. ממשו את הפעולות scale ו-times המבצעות הכפלה בסקאלר. השלימו את החוזים:

```
public void scale(double factor) {
}

public Point times(double factor) {
}
```

ד. ממשו את הפונקציה rotate, המסובבת את הווקטור הנתון סביב ראשית הצירים. שימו לב שהזווית נתונה ברדיאנים. דוגמאות:

```
Point r = new Point(1,1);
System.out.println(r);
> (x=1.00, y=1.00)
r.rotate(Math.PI/2);
System.out.println(r);
> (x=-1.00, y=1.00)
r.rotate(-Math.PI/2);
System.out.println(r);
> (x=1.00, y=1.00)
```

חתימת הפונקציה:

```
public void rotate(double theta) {
}
```

ה. ממשו את הפונקציה rotateBy, המסובבת את הווקטור הנתון סביב נקודה נתונה. גם כאן הזווית נתונה ברדיאנים.

**רמז:** אנו כבר יודעים לשובב נקודה סביב ראשית הצירים (סעיף קודם). אם כך, ניתן להזיז את שתי הנקודות במידה שווה, עד אשר הנקודה other מתלכדת עם ראשית הצירים. כעת, ניתן לשובב את הנקודה סביב הראשית בזווית הנדרשת. לבסוף, יש לבטל את ההזזה שביצענו, ע"י כך שנבצע הזזה הפוכה להזזה המקורית (כך ש-other חוזרת למקומה המקורי).  
דוגמא:

```
Point r = new Point(1,1);
Point s = new Point(2,2);
System.out.println(s);
> (x=2.00, y=2.00)
s.rotateBy(Math.PI/2, r);
System.out.println(s);
> (x=0.00, y=2.00)
```

חתימת הפונקציה:

```
public void rotateBy(double theta, Point other) {
}
```

1. ממשו את הפונקציה dot המבצעת מכפלה פנימית בין שני ווקטורים. דוגמא:

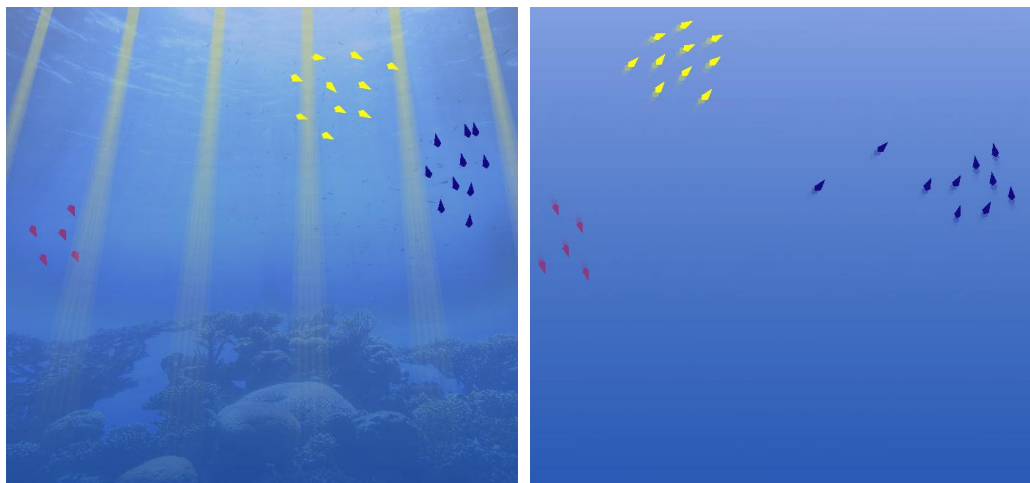
```
Point x = new Point(1,0);
Point y = new Point(0,1);
System.out.println(x);
> (x=1.00, y=0.00)
System.out.println(y);
> (x=0.00, y=1.00)
System.out.println(x.dot(y));
> 0.0
```

חתימת הפונקציה:

```
public double dot(Point other) {
}
```

## חלק ב':

בחלק זה נרצה לממש סימולציה של חיות. החיות פועלות עפ"י רשימת חוקים פשוטה (שתפורט בהמשך). בתחילה נממש התנהגות חיה בודדת ואח"כ נרחיב את המימוש להתנהגות חיות שהן חלק מלהקה.



שימו לב:

- יש להוריד מאתר הקורס את הקוד ההתחלתי לשם ביצוע חלק זה. לאחר שתשלימו את המימוש, תוכלו להריץ את האפליקציה המבצעת את הסימולציה.
- יש להשלים את מימוש המחלקה Point מחלק א' בטרם פותרים סעיף זה.
- מערכת הצירים בעולם הסימולציה היא:
  - ציר x גדל מימין המסך לשמאלו.
  - ציר y גדל מחלקו התחתון של המסך כלפי למעלה.

נממש תחילה את המחלקה Boid שתייצג חיה בודדת, בהמשך נרחיב את המחלקה כך שחיה תוכל להיות חלק מלהקה. חיה היא גוף נקודתי שנע במרחב, לכן נמדל חיות בעזרת מיקום ומהירות. בנוסף, לכל חיה יהיה מצביע אל עצם המתאר את הסביבה בה היא נמצאת (מופע של המחלקה Scene).

כל חיה נעה עפ"י חוקי התנהגות פשוטים. כל חוקי ההתנהגות שנתאר משפיעים על המהירות בלבד! ע"י שינוי המהירות (גודל וכוון) משפיעה החיה על מיקומה. למחלקה Boid יש מתודה move. המתודה תקרא ע"י תוכנת הסימולציה על מנת לדמות מעבר של פרק זמן קצר. בכל פרק זמן כזה, מיקום החיה יתעדכן בהתאם לווקטור המהירות הנוכחי שלה. זהו המקום היחיד בו משתנה מיקומה של החיה.



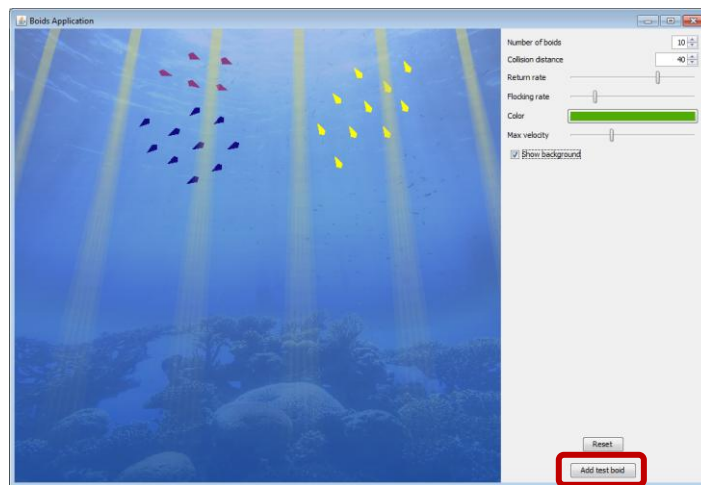
תחילה נתבונן בכלל הפשוט ביותר והוא שכל חיה שואפת להישאר בתחומי הסביבה המוגדרים. היא עושה זאת עפ"י הכלל הבא :

- אם החיה יוצאת מתחומי הסביבה לאורך ציר  $x$ , אזי היא מקטינה את מהירותה לאורך ציר זה ומגדילה את מהירותה לאורך ציר  $y$  ב-20%.
- בדומה, אם החיה יוצאת מתחומי הסביבה לאורך ציר  $y$  היא מקטינה את מהירותה לאורך ציר זה ומגדילה את מהירותה לאורך ציר  $x$  ב-20%.

בכמה יש להקטין את המהירות לאורך הציר בו חרגה החיה מהסביבה? זהו פרמטר הנקבע ע"י המשתמש. כל הפרמטרים הנוגעים לחיה בודדת ונקבעים ע"י המשתמש, ימצאו במחלקה `BoidParameters`. את גודל התיקון הנדרש ניתן להשיג באמצעות השאילתה `getReturnRate`.

1. ממשו את הפונקציה `stayInScene()`, המממשת את הכלל שתואר. הפונקציה נקראת מתוך המתודה `move`, לפני השורה המעדכנת את מיקומו של הפרט.

שימו לב, בשלב זה ניתן כבר לבדוק שההתנהגות אכן עובדת, ע"י הרצת המחלקה `BoidsApplication` ולחיצה על הכפתור "Add test boid". אם המימוש נכון, חיות אשר יוצאות מתחומי המסך יסתובבו. ייתכן שחלק ממסלול החזרה יהיה מחוץ לגבולות החלון.



2. עמ"נ לקבל סימולציה התואמת יותר למציאות, נרצה להגביל את מהירותן של החיות. ממשו את הפונקציה `constrainVelocity`, שתגביל את גודלו של וקטור המהירות כך שגודלו לא יעלה על זה המוגדר בפרמטר `getMaxVelocity`.

החיות שברשותנו אינן מגיבות לחיות אחרות. כלומר, ייתכן שהם יתנגשו אחת בשנייה במהלך תנועתן. ננסח כלל המאפשר לחיה להתרחק מחיות אחרות הקרובות אליה יתר על המידה. הכלל מנוסח מראות עיניה של חיה נתונה:

לכל חיה אחרת  $b$ :

- אם  $b$  רחוקה ממני (כלומר מרחקה ממני גדול מ-  $\text{getMinDistance}()$ , אתעלם ממנה.
- אחרת, אחשב את הכוון הנדרש להימנעות בדרך הבאה:
  - יהי  $p$  המיקום שלי ו- $q$  המיקום של  $b$ .
  - הכוון ממני אל  $b$  הוא:  $q-p$ , לכן ארצה לזוז בכוון ההפוך, לכוון זה.

$$c = \sum_i \frac{q_i - p}{\|q_i - p\|}$$

הכוון המשוקלל על פני כל החיות הקרובות נתון על ידי:

(כאשר  $\|p\|$  היא הנורמה של הווקטור  $p$ .)

כיצד נדע אלו חיות אחרות קיימות? נשתמש בשאילתת  $\text{getAllBoids}$  של המחלקה  $\text{Scene}$ .

3. ממשו את הפונקציה  $\text{avoidOtherBoids}$ , המממשת את התנהגות החיה עפ"י הכלל הנ"ל.

### חלק ג' – תסמונת העדר

בחלק זה נרצה להרחיב את התנהגותם של פרטים כך שינועו בלהקות. כל חיה בלהקה מתנהגת כמו חיה שאיננה חלק מלהקה (כלומר כל החוקים הקודמים חלים עליו), אולם ישנם חוקים נוספים המשפיעים על תנועתה. המחלקה  $\text{Flock}$  מייצגת להקה.

הכלל הבסיסי ביותר המשפיע על חברי להקה הוא רצונם להישאר ביחד. לשם כך, כל אחד מחברי הלהקה מנסה לנוע לכוון מרכז המסה של הלהקה כולה.

1. ממשו את הפונקציה  $\text{getCenterOfMass}$  במחלקה  $\text{Flock}$  המחזירה את מרכז המסה של חברי הלהקה. מרכז המסה של להקה שמיקומי חבריה נתונים על ידי:  $p_1, p_2, \dots, p_n$  מוגדר ע"י:

$$M = \frac{1}{n} \sum_i p_i$$

2. ממשו את הפונקציה stayWithFlock של המחלקה Boid. כאשר Boid הוא חלק מלהקה (כלומר, flock != null) אזי הוא מתקן את כוון מהירותו לעבר מרכז המסה של הלהקה. קצב התיקון (כלומר גודלו של הווקטור המחושב לצורך שינוי המהירות) נתון בפרמטר getFlockingRate של המחלקה Flock.

**רמז:** על מנת להתקדם מנקודה  $a$  לנקודה  $b$  יש לזוז בכוון הווקטור  $b-a$  (כי  $a+(b-a) = b$ ).

כלל נוסף המשפיע על חברי הלהקה הוא רצונם לשמור על מהירות דומה (כוון וגודל). כל אחד מחברי הלהקה מתקן את מהירותו ביחס למהירות הממוצעת של כל חברי הלהקה.

3. ממשו את הפונקציה getAverageVelocity במחלקה Flock המחזירה את המהירות הממוצעת של חברי הלהקה. המהירות הממוצעת של הלהקה שמהירויות חבריה נתונים ע"י  $v_1, v_2, \dots, v_n$

$$V = \frac{1}{n} \sum_i v_i \quad \text{מוגדרת בביטוי:}$$

4. ממשו את הפונקציה maintainFlockVelocity של המחלקה Boid. כאשר Boid הוא חלק מלהקה הוא מתקן את כוון מהירותו כך שיפנה לעבר הכוון של המהירות הממוצעת של חברי הלהקה. קצב התיקון נתון בפרמטר getFlockingRate של המחלקה Flock.

בשלב זה ניתן לבדוק את המימוש. בחלקו הימני של חלון האפליקציה<sup>1</sup> ניתן לשנות את ערכי הפרמטרים השונים. בעזרה לחיצה וגרירה של העכבר בחלקו המרכזי של חלון האפליקציה, ניתן לייצר להקות עפ"י הפרמטרים שהוגדרו (הלהקות יוצרו במקום ובכוון אותו סימנתם עם העכבר).

## ב ה צ ל ח ה !

---

<sup>1</sup> תמונת הרקע הורדה מהאתר : <http://greenupforlife.files.wordpress.com/2010/12/ocean-floor.jpg>