

תכנות מונחה עצמים – תרגיל 2

תאריך ההגשה : 18.02.2024 בשעה 23:55

תרגיל זה מומלץ להגיש בזוגות.

0. הקדמה

בתרגיל זה עליכם לממש משחק דמוי "[Arkanoid](#)" שנקרא לו "Bricker".

לצורך משימה זו, אנחנו מספקים לכם מנוע-משחק בדמות הספריה DanoGameLab (בגרסה 1.1.0). בעזרת ספריה זו ניתן ליצור עצמים של המשחק (GameObjects) שאותם ינהל המנוע בעזרת מנהל המשחק, (ה-)GameManager. בתרגיל זה אתם תתכנתו את המשחק באמצעות המנוע הזה.

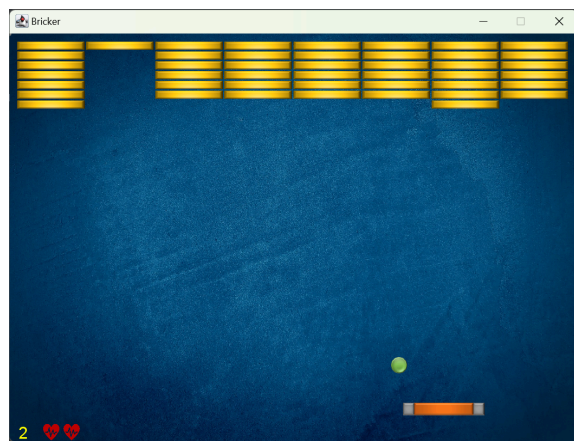
בתרגיל זה נעבוד בשני שלבים, בחלק הראשון נבנה משחק בסיסי ובחלק השני נוסיף למשחק אפקטים מיוחדים. מומלץ לעבור על כל הוראות התרגיל במלואן לפני תחילת העבודה. החלק הראשון של התרגיל ניתן בצורה מודרכת ומלווה בסרטונים. חלק זה יעזור לכם ללמוד איך לעבוד עם הספריה של DanoGameLab. בנוסף לסרטונים באתר, תוכלו למצוא במודל גם [מדריך](#) מקוצר לעבודה עם הספריה. בחלק השני של התרגיל תצטרכו אתם לחשוב על העיצוב ולהחליט בעצמכם אילו מחלקות ליצור ובאילו תבניות עיצוב להשתמש בהתאם לעקרונות שלמדנו.

המשחק צפוי להיראות בערך כך:

חלק ב- המשחק המורחב:



חלק א- המשחק הבסיסי:



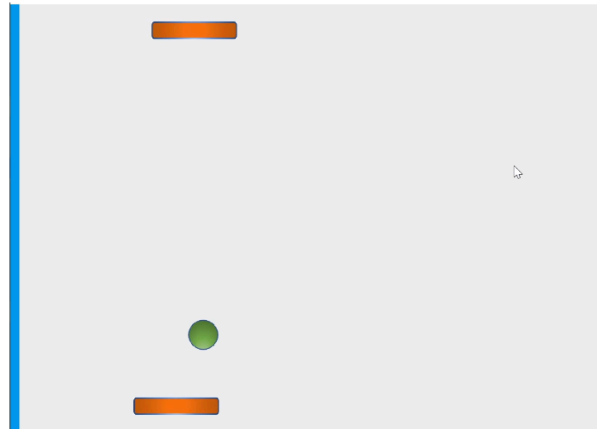
במהלך המשחק השחקן מזיז את הדיסקית שזזה שמאלה וימינה, הכדור צריך לשבור את כל הלבנים, ואסור שהכדור יפול אל מעבר המסך.

במשחק הבסיסי: כשלבנה נשברת היא נעלמת וזהו. כשכולן נשברו - ניצחנו. במשחק המורחב: כל מיני דברים עשויים לקרות כל פעם שלבנה נשברת: ייתכן שיופיעו כדורים מסיחים, דיסקית נוספת, ייפלו חיים נוספים, או שהמצלמה תשנה את הפוקוס שלה.

שימו לב! בתרגיל זה צפויים לכם שני אתגרים חדשים, גם ללמוד איך לעבוד עם ספריה חדשה, וגם בפעם הראשונה בקורס אתם צריכים לחשוב על עיצוב הקוד בחלק מהתרגיל בעצמכם!

1. חלק ראשון

בחלק זה של התרגיל נבנה את המשחק הבסיסי. המדריך של חלק זה יתבסס על סדרת סרטונים שעוסקים בהכנת משחק מאוד דומה ל-Bricker שנקרא לו Ping והוא נראה כך:



גם ב-Ping השחקן משחק בדיסקית התחתונה ואסור לתת לכדור ליפול, גם כאן הכדור מקפץ וכן הלאה, רק שבמקום לבנים שנשברות בחלק העליון, ישנה שם דיסקית יריבה שמשחק המחשב. אם דיסקית היריב נותנת לכדור לחלוף על פניה אל מעל למסך, השחקן ניצח. אנחנו נעקוב אחרי סדרת הסרטונים ובגדול נעתיק ממנה באופן גס, פרט להבדלים שיצוינו לצד כל סרטון.

היכנסו לאתר קמפוס פרק 4 חלק [JavaX: Bricker 4.4](#) ועקבו אחר ההוראות ליצירת המשחק מהמדריך להתקנת הספריה ועד לסיום תת הפרק. **שימו לב למספר הסעיף בראש כל לשונית לצורך מענה על שאלות בפורום.**

המבנה של ה-API של התרגיל שלכם בסוף החלק הראשון צריך להיות לפי ההגדרות הבאות (שימו לב שמצויינים ב-API רק שדות ושיטות שהן public):

`package bricker.gameobjects:`

1) Class Ball

- constructor
- void onCollisionEnter(GameObject other, Collision collision)
- int getCollisionCounter()

2) Class Paddle

- constructor
- void update(float deltaTime)

3) Class Brick

- constructor

- void onCollisionEnter(GameObject other, Collision collision)
- 4) **A class or classes that implement a graphic and a numeric life counters.**

package bricker.brick_strategies:

5) Interface CollisionStrategy

- onCollision(GameObject thisObj, GameObject otherObj)

6) Class BasicCollisionStrategy

- constructor
- onCollision(GameObject thisObj, GameObject otherObj)

package bricker.main:

7) Class BrickerGameManager

- public static void main(String[] args)
- constructor
- void initializeGame(ImageReader imageReader, SoundReader soundReader, UserInputListener inputListener, WindowController windowController)
- void update(float deltaTime)
- **Possibly more methods according to your design.**

הערות חשובות:

1. הסרטונים במדריך הם קטעים חתוכים מתוך סרטוני יוטיוב וביניהם קטעי טקסט עם הוראות. לבטיחותכם והנאתכם, צפו בהם דרך קמפוס, לא דרך יוטיוב.
2. הגדלים והמיקומים המופיעים בסרטים גם בגדר המלצה. אתם יכולים לשנות מעט את הנתונים כדי לשפר את חווית המשחק או אם תראו בכך צורך בשביל החלק השני. לא תיבדקו על קבועים מדויקים.

2. חלק שני:

2.1. ההתנהגויות של הלבנים

לאחר שבחלק הראשון מימשתם את המשחק Bricker לפי המדריך באתר קמפוס, בחלק זה תממשו התנהגויות מיוחדות לחלק מהלבנים שיקרו כאשר הכדור יתנגש בלבנים. כל הלבנים במשחק יעלמו כאשר הכדור פוגע בהן, אך חצי מהלבנים יגרמו לדברים נוספים לקרות כאשר הן נעלמות.

הערה חשובה! קראו את התרגיל עד הסוף לפני שאתם מתחילים במימוש!

בחלק הזה של התרגיל ניתן לכם חופש עיצובי, אך אם עברתם על כל התרגיל מראש, ועיצבתם נכון את המחלקות שלכם בהתאם לעקרונ ה-Open-Close, השינויים בחלק הזה צפויים להיות רק בהוספת מחלקות ושינויים

למחלקת מנהל המשחק, כלומר לא אמור להיות צורך בשינויים ב-API של המחלקות שיצרתם בחלק הראשון מלבד המחלקה `BrickerGameManager`.

ההתנהגויות של הלבנים יהיו לפי ההתפלגות הבאה (הסבר מפורט של ההתנהגויות המיוחדות בחלקים הבאים):

- בהסתברות $\frac{1}{2}$: התנהגות רגילה - ההתנהגות אותה ממשותם כבר בחלק הראשון (הלבנה נשברת ונעלמת).
- בהסתברות 1/10: כדורים נוספים.
- בהסתברות 1/10: דיסקית נוספת.
- בהסתברות 1/10: שינוי מצלמה.
- בהסתברות 1/10: החזרת פסילה.
- בהסתברות 1/10: התנהגות כפולה.

סה"כ בתוחלת, כחצי מהלבנים יהיו רגילות וכחצי עם התנהגות מיוחדת, כאשר לכל אחת מההתנהגויות המיוחדות תהיה הסתברות שווה להופיע במשחק.

2.2. התנהגויות מיוחדות

2.2.1. כדורים נוספים

בעת שבירת לבנה בעלת התנהגות זו, ייווצרו במרכז המיקום של הלבנה (במקומה) 2 כדורים לבנים להם נקרא "Puck", התכונות של כל Puck:

- 2.2.1.1. התמונה שלו היא `mockBall.png` הנמצאת בקבצי `assets`.
- 2.2.1.2. משתמש באותו סאונד של הכדור הרגיל.
- 2.2.1.3. גודלו $\frac{3}{4}$ מגודל הכדור הרגיל גם באורך וגם ברוחב.
- 2.2.1.4. הכיוון ההתחלתי שלו יהיה אלכסון רנדומלי. שימו לב שיתכן ושני הכדורים שתיצרו יגרילו את אותו אלכסון ולכן זה יראה כאילו נוצר רק כדור אחד. מצב זה תקין.
- 2.2.1.5. הוא יכול להתנגש ב-pucks האחרים, בדסקית, בלבנים בקירות וכן בכדור המרכזי עצמו (ball).
- 2.2.1.6. אם הוא יצא מגבול המשחק, הוא אינו משפיע על כמות הפסילות של השחקן ויש לדאוג להסיר אותו מרשימת האובייקטים של המשחק.

2.2.2. דיסקית נוספת

בעת שבירת לבנה בעלת התנהגות זו, תיווצר דיסקית נוספת על המסך. התכונות של הדיסקית הנוספת:

- 2.2.2.1. זהה לדיסקית הרגילה (באותה גודל ואותה תמונה של הדיסקית הרגילה).

2.2.2.2. היא תיווצר במרכז ציר ה-X של המסך, ובגובה של חצי מהמסך.

2.2.2.3. תזוז גם היא לפי תנועות השחקן בדומה לדיסקית המקורית.

2.2.2.4. הדיסקיות לא חייבות להיות מקבילות. כלומר, אם הדיסקית המקורית הייתה צמודה לקיר השמאלי, ואז הופיעה הדיסקית המשנית באמצע, כשנלחץ ימינה ונביא את הדיסקית המקורית לאמצע המסך, הדיסקית המשנית תגיע לקיר הימני. אם נמשיך ללחוץ ימינה רק הדיסקית המקורית תמשיך לזוז כיוון שהדיסקית המשנית כבר הגיע לקצה המשחק.

2.2.2.5. בכל רגע נתון יכולה להיות רק דיסקית נוספת אחת. כלומר, סה"כ יכולות להיות רק שתי דיסקיות ברגע נתון במשחק. אם כבר קיימת דיסקית משנית, לא ניצור חדשה.

2.2.2.6. לאחר 4 פגיעות בה, הדיסקית הנוספת תיעלם מהמסך.

2.2.3. שינוי מצלמה

בעת שבירת בלבה המחזיקה בהתנהגות זו ע"י הכדור הראשי, מצלמת המשחק תעקוב אחרי הכדור. כדי לשנות את המצלמה למעקב אחרי הכדור, נשתמש בשורות קוד הבאות:

```
setCamera(  
    new Camera(  
        ball, //object to follow  
        Vector2.ZERO, //follow the center of the object  
        windowController.getWindowDimensions().mult(1.2f), //widen the frame a bit  
        windowController.getWindowDimensions() //share the window dimensions  
    )  
);
```

כדי לאתחל חזרה את המצלמה, נשתמש בקוד הבא:

```
gameManager.setCamera(null);
```

הכללים של שינוי המצלמה:

2.2.3.1. המצלמה תשתנה רק אם הכדור הראשי פגע בלבנה. אם הכדורים הנוספים (Puck) פגעו בלבנה, לא יקרה כלום. המצלמה תעקוב אחרי הכדור, ורק כאשר הוא התנגש ב-4 עצמים, המצלמה תחזור לקדמותה.

2.2.3.2. התנהגות המצלמה תופעל רק במידה וההתנהגות לא פעילה כבר
(ניתן לבדוק ע"י התנאי `gameManager.camera() == null`)

הכוונה לעיצוב:

2.2.3.3. חשבו איך לעצב את הקוד כך שיהיה ניתן לבדוק איזה כדור פגע בלבנה ולשנות את המצלמה רק אם זה הכדור הראשי.

2.2.3.4. חשבו על מנגנון שיעקוב אחרי ההתנגשויות של הכדור וידאג להחזיר את המצלמה למצב הרגיל לאחר 4 התנגשויות. מומלץ להשתמש בשיטה `getCollisionCounter` של הכדור שהגדרנו בחלק הראשון של התרגיל.

2.2.4. החזרת פסילה

בעת שבירת לבנה המחזיקה בהתנהגות זו, ממרכז הלבנה ייפול אובייקט לב, אשר על הדיסקית "לאסוף" על מנת להחזיר פסילה. כללים של החזרת פסילה:

- 2.2.4.1. גודל הלב הנופל יהיה כגודל הלב שהגדרתם בהצגת מספר החיים וישתמש באותה התמונה.
- 2.2.4.2. הלב הנופל יופיע במיקום של מרכז הלבנה שנעלמה, ויפול במהירות קבועה של 100 בכיוון ציר ה-Y (כלומר רק למטה).
- 2.2.4.3. כמות הפסילות המקסימלית במשחק תהיה 4 (כאשר עדיין הכמות ההתחלתית היא 3). ייתכן ותצטרכו בשלב זה לשנות את הקוד של המחלקות (או המחלקה) שמממשות את הצגת מספר החיים, כדי להתאים את המחלקות להתנהגות החדשה.
- 2.2.4.4. הלב הנופל יכול להתנגש רק עם הדיסקית המקורית (ולא הדיסקית הנוספת). אם הלב לא מתנגש עם הדיסקית ויוצא המסך, יש להסירו מרשימת האובייקטים של המשחק.

הכוונה לעיצוב:

- 2.2.4.5. כדי לדאוג שהלב מתנגש רק עם הדיסקית המקורית ניתן לדרוס את הפונקציה `shouldCollideWith` שקיימת ל-`gameObject` (שאותו מחלקת הלב תירש). פונקציה זו מאפשרת להגדיר בעת זיהוי התנגשות האם ההתנגשות חוקית.
- 2.2.4.6. חשבו איך יהיה נכון לבדוק שאכן התנגשתם בדיסקית המקורית בלי שימוש ב-`instanceOf`. טיפ: להשתמש בתכונה `tag` של `GameObject`.

2.2.5. התנהגות כפולה

עבור לבנה בעלת התנהגות זו יוגרלו שתי התנהגויות מיוחדות מבין 5 ההתנהגויות האפשריות (כולל עוד שכבה של התנהגות כפולה) והיא תפעיל את שתיהן כאשר הכדור מתנגש בה.

כללים של התנהגות כפולה:

- 2.2.5.1. ללבנה נתונה יהיו **לכל היותר** 3 התנהגויות מיוחדות- למשל, לבנה תחזיק התנהגות של כדורים נוספים, התנהגות של דיסקית נוספת והתנהגות של החזרת פסילה, (וכמובן יהיה לה את ההתנהגות הבסיסית של הסרת לבנה כמו כל לבנה).
- שימו לב!** מקרה המקסימום שבו יש 3 התנהגויות כפולות נוצר מכך שהגרלנו עבור לבנה התנהגות כפולה, ואז **לאחת** מההתנהגויות המיוחדות הגרלנו שוב התנהגות כפולה. לא ניתן להרגיל שוב התנהגות כפולה מעבר לכך.
- 2.2.5.2. התנהגות מסוימת יכולה לחזור על עצמה בהתנהגות כפולה, למשל ייתכן שתיבחר התנהגות של כדורים נוספים פעמיים לאותה לבנה (במקרה זה ייווצרו 4 כדורים).

טיפ לעיצוב: התנהגות כפולה היא בעצם התנהגות מיוחדת חדשה ש"עוטפת" שתי התנהגויות מיוחדות אחרות. השתמשו בתבנית עיצוב מתאימה שלמדנו המתאימה למצב הזה. שימו לב שמימוש של התנהגות אחת לא אמור להיות מושפע מהוספת מימוש של התנהגות אחרת, ובפרט, ההתנהגויות המיוחדות האחרות לא אמורות להיות מושפעות מכך שיש התנהגות שהיא כפולה אלא זו התנהגות נפרדת.

2.3. קריטריונים והכוונה לעיצוב נכון.

בחלק השני של התרגיל זה יש לכם מעט חופש עיצובי, ולא הגדרנו לכם API שאותו יש לממש. לעומת זאת, ישנם כמה קריטריונים לעיצוב מתקבל:

2.3.1. הימנעו מריבוי מחלקות מיותרות. לדוגמא, יצירת מחלקה לכל שילוב של התנהגויות אפשרי, הוא לא עיצוב תקין.

2.3.2. חישבו על API מינימלי ועל מגדירי נראות לכל פונקציה ואיבר במחלקות שלכם.

2.3.3. את התרגיל ניתן לפתור בעזרת עץ ירושה שאינו גבוה (כלומר אין צורך באבא-בן-נכד-נין וכו'). ניתן להשתמש בכלים שלמדתם בקורס (ירושה, ממשקים, הכלה וכו'). יש להסביר את השימוש בהם ב-README.

2.3.4. חישבו על קריאות הקוד, אפשרות התחזוק שלו והרחבתו כאשר אתם מעצבים את הפיתרון שלכם.

2.3.5. מותר להרחיב את ה-API של חלק 1 אם אתם יודעים להסביר למה זה נצרך בעיצוב שלכם. אם תבחרו לשנות את ה-API הנתון, תתבקשו לתעד ב-README את הסיבה לכך, וכיצד זה תורם לעיצוב הפיתרון שלכם.

כעת ננסה לתת מעט הכוונה לעיצוב אפשרי:

2.3.6. את ההתנהגות הבסיסית כבר יצרתם. חשבו על הדרך הנכונה להגדיר את ההתנהגויות הנוספות, מה מבנה המחלקות שלהן, מה ה-API שלהן, ואיך יוצרים אובייקטים שלהן.

2.3.7. חשבו מבחינה עיצובית איפה יהיה המקום הנכון בקוד להגריל את ההתנהגויות של כל לבנה.

2.3.8. חשבו כיצד ניתן ליצור כמה התנהגויות במקביל ללבנה אחת. נסו לבנות את התוכנית כך שיהיה קל להרחיב אותה לדרישות עתידיות.

2.3.9. חשבו כיצד לעצב את הקוד כדי לכלול את ההגבלה של 3 התנהגויות מיוחדות ללבנה. עצבו את הקוד כך שאם תתבקשו להרחיב את התוכנה כך שיהיה ניתן ליצור לבנה עם יותר מ-3 התנהגויות, תוכלו לעשות את השינוי במינימום שינויי קוד.

2.3.10. זכרו את ה-Coding Style של הקורס, והיצמדו להוראות.

3. הוראות הגשה

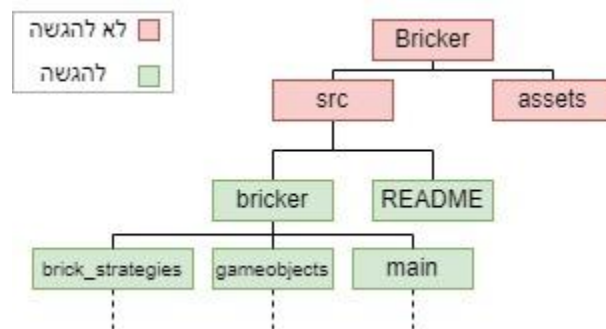
3.1. הגישו את התרגיל כקובץ jar/tar/zip בשם **ex2** (והסיומת בהתאם).

3.2. בתוך קובץ זה, ימצאו הקבצים הבאים:

3.2.1. התיקיה bricker המכילה את קבצי הפתרון שלכם לפי ה-API המפורט בסוף חלק 1, בתוספת המחלקות שהוספתם עבור חלק 2. הקפידו על חלוקה לחבילות. בכל קובץ תופיע רק מחלקה אחת ששמה כשם הקובץ.

3.2.2. קובץ בשם README (ללא סיומת) לפי ההוראות בסעיף 3.4.

3.2.3. כלומר, זה עץ ההגשה של הפרוייקט כאשר יש להגיש את התיקיות והקבצים **הירוקים**:



שימו לב - כל חבילה שהוספתם לקוד צריכה להיות תת חבילה של החבילה bricker.

3.3. אין להגיש את קבצי ה-assets (דאגו לא לשנות את השמות של הקבצים בהם אתם משתמשים בקוד שיהיו זהים לקבצים שהורדתם).

3.4. קובץ ה-README:

3.4.1. בשורה הראשונה בקובץ יופיע שם המשתמש CSE שלכם. אם אתם מגישים בזוג, יש להפריד בין שמות המשתמש עם פסיק (שניהם באותה שורה).

3.4.2. בשורה השניה מספר תעודת הזהות. אם אתם מגישים בזוג, יש להפריד בין מספרי הזהות עם פסיק (שניהם באותה שורה).

3.4.3. השורה השלישית ריקה.

3.4.4. החל מהשורה הרביעית ענו בקובץ על השאלות הבאות לפי הסדר:

1. כתבו איזה עיצוב בחרתם בחלק 1.7 כדי לאפשר העלמת לבנים במשחק, והסבירו מה הם **היתרונות והחסרונות** של הבחירה העיצובית שלכם.

2. הסבירו איך מימשתם את הצגת מספר החיים של השחקן (גרפי ונומרי) מחלק 1.8 של התרגיל. הסבירו בקצרה מה תפקידה של כל מחלקה שהוספתם בקוד.

3. הסבירו בקצרה כיצד מימשתם כל אחת מהתנהגויות המיוחדות (מלבד הכפולה). הסבירו על תפקידן של מחלקות שהוספתם כדי לממש את ההתנהגות, אם הוספתם.

4. הסבירו כיצד מימשתם את ההתנהגות הכפולה בחלק 2.2.5. התייחסו בהסבר לעיצוב של הקוד, וכן כיצד הגבלתם בקוד את כמות ההתנהגויות ל-3.
5. אם עשיתם בחלק 2 של התרגיל שינויים ב-API שמימשתם בחלק 1 שלו, הסבירו בקצרה למה השינוי הזה היה הכרחי לעיצוב שלכם.

4. בדיקת ההגשה

- 4.1. לאחר שהגשתם את התרגיל בתיבת ההגשה הקוד שלכם יעבור טסט `presubmit` ויווצר הקובץ `submission.pdf`.
- 4.2. וודאו שהקובץ נוצר בהצלחה.
- 4.3. הקובץ מכיל פלט של הטסט המוודא שהקוד שלכם מתקמפל, ומפרט על שגיאות בסיסיות. השתמשו בפלט שבקובץ על מנת לתקן שגיאות בתרגיל שימנעו מאיתנו להריץ את הטסטים הסופיים. (זהו טסט קדם הגשה ולא הטסט הסופי של התרגיל).
- 4.4. ניתן להגיש את התרגיל שוב ושוב ללא הגבלה עד למועד ההגשה. (ההגשה האחרונה היא ההגשה הסופית).
- 4.5. שימו לב: על פי נהלי הקורס חובה לעבור את הטסט ה `presubmit` ללא שגיאות. קובץ הגשה שלא עובר בהצלחה את הטסט יקבל ציון 0 ולא ייבדק!
- 4.6. ניתן לחלופין להריץ ישירות את ה `presubmit` על ידי הרצת הפקודה הבאה (במחשבי בית הספר):
`~oop1/ex2_presubmit <path to your file>`
- 4.7. שימו לב שפקודה זו לא מגישה את התרגיל בפועל אלא רק מריצה את ה `presubmit`.
- 4.8. חובה לעבור תמיד גם על הפלט של `submission.pdf` לאחר ההגשה בתיבת ההגשה לוודא שהכל תקין!

בהצלחה!