פיתוח משחק בJavaFX ועבודת מחקר על התמיכה ברב-חוטיות בJavaFX

שם המגיש: נתנאל רון

שם המנחה: ד"ר יואב רודה

הקדמה:

בעבודה זו אעסוק בפיתוח משחק בJavaFX. המשחק יהיה דינאמי עבור משתתף יחיד, בו המשתמש ישחק את הדמות הראשית כאשר מטרת המשחק היא לצבור נקודות על ידי חיסול מפלצות, המשחק יכלול אנימציות עשירות לדמויות ותפריט משתמש אינטרקטיבי. בנוסף בעבודה זו אעסוק בתמיכת JavaFX ברב-חוטיות, אממש ואשווה בין האפשרויות השונות שספרייה זו נותנת בהקשר של המשחק הן מבחינת ביצועים והן מבחינת נוחות וטבעיות של הקוד בהקשר של משחק מחשב.

הסבר קצר על המשחק:

כפי שצוין בהקדמה, במשחק שיצרתי ראשית יהיה תפריט אינטרקטיבי- בתפריט קיימים 4 כפתורים שלכל אחד תפקיד אחר: כפתור Play- לאחר לחיצה עליו יתחיל המשחק, כפתור Top Ten- לאחר לחיצה עליו יופיע לוח התוצאות בו שמות ותוצאות העשירייה עם התוצאות הטובות ביותר, כפתור Credits- לאחר לחיצה עליו יופיע מסך הקרדיטים שיכיל את כל היוצרים של המשאבים בהם השתמשתי, כפתור Quit- לאחר לחיצה עליו המשחק יסגר. כפתורים אלה כולם מופעים של המחלקה GameButton מחלקה זו מתארת כפתורים עם רקע מסוים שמשנים את תצוגתם כאשר העכבר נכנס/יוצא מגבולתיהם. בעת לחיצה על כפתורי ה-Top Ten/Credits תופיע/תוסתר תת-סצנה שהיא מופע של המחלקה MenuSubScene- מחלקה היורשת מ-javafx.scene.SubScene שבה יש מתודה להגדרת התת-סצנה (כותרת ותוכן) ומתודה להזזת התת-סצנה (בעזרת Transition, ההזזה תתבצע כך שאם היא מופיעה אז היא תוסתר, ואם היא מוסתרת אז היא תופיע), תוכן תתי הסצנות נקרא מתוך קבצים.

בעת לחיצה על כפתור ה-Quit תיסגר הבמה שמכילה את התפריט הראשי ובכך יסגר המשחק. בעת לחיצה על כפתור ה-Play יווצר אובייקט מסוג Game, ותופעל מתודה להחלת המשחק על הבמה הנוכחית.

מחלקת Game היא עיקר המשחק ובעצם מתפעלת את כל מהלך המשחק. יש בה שימוש ב-AnimationTimer ראשי (מחלקה הקוראת למתודת ה-()handle שלה בכל 1/60 שניות ומאפשרת עדכון שוטף של סצנת המשחק) שקוראת בין היתר למתודת ()update, ושלל מתודות עזר לתפעול המשחק. במשחק לכל האובייקטים יש מחלקת אב משותפת- AnimatedGameObject שלמעשה מתארת כל אובייקט בר תזוזה בעל יכולת אנימציה, ממנה יורשות עוד שלושה מחלקות עיקריות- מחלקת GameCharacter- מחלקה המתארת אובייקטים מסוג שחקן, מחלקת Projectile – מחלקה המתארת את האובייקט שנוצר בעת התקפת השחקן ומחלקת Enemy – מחלקה המתארת את כל האויבים במשחק.

תזוזת השחקן והמתקפות שלו מתבצעות בעזרת מתודת updateLocation() אשר מקבלת את הזמן שחלף (בעזרת חישוב קל ממתודת handle()) ומחשבת את מיקומו ביחס למיקום הקודם שלו ובהתחשבות במהירותו (במקרה של השחקן, קיים גם כוח גרביטציה אשר מושך אותו למיקום מקסימלי בציר Y, וזאת משום שהשחקן מסוגל גם לקפוץ).

תזוזת האויבים מתבצעת על ידי PathTransition כך שנבחרת תנועה אקראית שלהם (ימין/שמאל וקפיצה/צניחה) בכל שניה.

בנוסף בכל קריאה למתודת update() מתבצעת בה בדיקה עבור כל מתקפה/אויב אם גבולותיהם נחתכים (ובכך בעצם ישנה פגיעה של מתקפת השחקן באויב) ובמידה ויש פגיעה, יורדות נקודות החיים של האויב והמתקפה מוסרת ובמקומה מופיע אנימציית פיצוץ קצרה. בנוסף גם נבדק חיתוך גבולות בין השחקן לאויבים, ובמידה וקיים אז יורדות לשחקן נקודות חיים ומוצגת אנימציה של עמעום השחקן (המסמלת גם את הזמן שבו השחקן לא יכול להיפגע שנית).

המשחק ייגמר כאשר יגמרו נקודות החיים של השחקן, ובכך יעצר המשחק ויופיע מסך בו יתבקש השחקן להקליד את שמו, תוצאתו ושמו ישמרו וייכתבו לתוך קובץ טקסט שימויין מחדש ויכיל לבסוף את 10 השמות והתוצאות הטובות ביותר.

במהלך המשחק יכול השחקן ללחוץ על כפתור ה-Esc לשם עצירת המשחק ובו יופיעו לו מקשי המשחק ואפשרות לחזור לשחק.

המשחק יכיל צלילים ומוזיקת רקע המשתנה בין התפריט הראשי למשחק עצמו.

<u>רב חוטיות ב-JavaFX</u>:

JavaFX תומכת ברב חוטיות בשלל דרכים, כאשר לשם המימוש קיימת מגבלה עיקרית שעליה יש להקפיד- כל שינוי ב-GUI (ממשק משתמש גרפי) יתבצע אך ורק מתוך חוט האפליקציה (Application Thread). לפיכך מימוש רב חוטיות בJavaFX ייעשה לרוב כאשר ישנם חישובים כבדים, אותם ניתן לחשב בנפרד ולדווח על התוצאה או על מהלך החישוב לחוט האפליקציה וזאת על מנת להקל על חוט האפליקציה ולאפשר לו להתעסק בעיקר במטרה לשמה הוא קיים- עדכון ה-GUI. כמובן שכל מימוש לצרכים שאינם עדכון ה-GUI יהיו באותו אופן כמו רב חוטיות בתוכנית Java רגילה.

בסעיפים הבאים אתאר בקצרה את השיטות העיקריות למימוש רב חוטיות על ידי שימוש בקוד של המשחק אותו יצרתי.

שיטה א' – יצירת חוט עם Runnable ושימוש במתודה Platform.runLater():

שיטה זו היא למעשה הבסיסית והפשוטה ביותר, היות וכל שינוי ב-GUI צריך להתבצע אך ורק בתוך חוט האפליקציה, מתודה זו תאפשר לכל חוט אחר שפועל ברקע לשלוח קטע קוד שיתבצע בתוך חוט האפליקציה כאשר חוט האפליקציה יתפנה לכך (הכנסתו לתוך תור עדיפויות של חוט האפליקציה, וכשיגיע תורו הוא יתבצע). כלומר, שיטה זו תקרא מתוך חוט כלשהו, כאשר כל הקוד שהיא עוטפת בתוכה יתבצע מתוך חוט האפליקציה- לדוגמא:

```
1.  private void displayDamage(String color,int dmg,Pane node) {
2.              new Thread(()->{
3.                          Label label=new Label(String.format("%d", dmg));
4.                          label.setTextFill(Paint.valueOf(color));
5.                          int miliTime=0;
6.                          Platform.runLater(()->{
7.                                      gamePane.getChildren().add(label);
8.                          });
9.                          while(miliTime<=400) {
10.
      label.setFont(Font.font("Impact",((miliTime+100)/100)*2+10));//increase size according to
    time passed
11.                                     label.setOpacity(label.getOpacity()-0.02);
12.                                     Platform.runLater(()->{
13.                                                 label.setTranslateY(node.getTranslateY()-30);
14.                                                 label.setTranslateX(node.getTranslateX()+10);
15.                                     });
16.                                     //count the time
17.                                     try {
18.                                                 Thread.sleep(16);
19.                                     } catch (InterruptedException e) {}
20.                                     miliTime+=16;
21.                         }
22.                         Platform.runLater(()->{
23.                                     gamePane.getChildren().remove(label);
24.                         });
25.             }).start();
26.   }
```

בדוגמא זו, המתודה displayDamage() מקבלת מחרוזת אותה תציג, צבע שבו המחורזת תוצג ואובייקט שאליו המחורזת תיצמד. המתודה תפעיל חוט שירוץ כחצי שניה תיצור תווית טקסט אשר תופיע למשתמש כאשר גודלה ועמימותה יגדלו כפונקציה של הזמן וכי היא תראה כמרחפת מעל האוביקט שנשלח. ספירת הזמן תתבצע על ידי Thread.sleep() והעלאת משתנה בהתאם (16 מילי שניות כדי לדמות קצב רענון של 60 פריימים בשניה). ניתן לראות שהמתודה יוצרת חוט אשר יפעל ברקע,וכי הפעולות היחידות שעטופות ב-Platform.runLater() הינם פעולות על ה-GUI והן הוספת התווית למסך, שינוי מיקומה במסך והסרתה.

שיטה ב' – שימוש במחלקה הגנרית-Task<V>

Task הינה מחלקה אבסטרקטית אשר בעת שימוש בה יש לדרוס את שיטת ה-call() שלה. את המופע המחלקה האנונימית שיצרנו עבור מחלקה זו אנו שולחים כפרמטר ל-new Thread() ובכך למעשה יוצרים חוט אשר מבצע את שיטת call(). עיקר ההבדל בין השימוש ב-Task לבין שימוש בכל Runnable אחר כפרמטר ליצירת חוט הוא ש-Task נבנה במיוחד לשימוש בJavaFX, זאת על ידי מימוש הממשק האבסטרקטי Worker<V> אשר מכיל Properties שאפשר להקשיב להם או לקשור אותם אך ורק מתוך חוט האפליקציה ובכך למעשה תוך כדי ריצת החוט שמפעיל את Task, ניתן יהיה לעקוב אחריו ולפעול בהתאם מתוך חוט האפליקציה. דוגמא פשוטה לכך היא בקוד הבא:

```
1.  private void displayHealth(int damage,AnimatedGameObject obj) {
2.          Task<Void> task=new Task<Void>() {
3.                  @Override
4.                  protected Void call() throws Exception {
5.                          for(int i=0;i<damage;i+=5) {
6.                                  updateProgress(i,damage);
7.                                  Thread.sleep(16);
8.                          }
9.                          return null;
10.                 }
11.         };
12.
     task.progressProperty().addListener((observable,oldvalue,newvalue)->{
13.                 synchronized(task) {
14.                         if(obj.life.get()>0)
15.                                 obj.life.set(obj.life.get()-5);
16.                 }
17.         });
18.         new Thread(task).start();
19.  }
```

מתודה זו מקבל מספר שלם המסמל את כמות הנזק, ואת האובייקט שלו נגרם הנזק ולמעשה מורידה מנקודות החיים של האובייקט כגודל הנזק באופן רציף (כלומר,בכל 1/60 שניות ירדו נקודות החיים של האובייקט ב-5 עד שהורד לו כל הנזק שנגרם). זאת נעשה על ידי הרצת חוט שבו יש Task שכל מה שהוא עושה הוא לעדכן את ה Progress Property שלו בכל 1/60 שניות. כפי שציינתי לעיל השימוש הקלאסי ב-Task הוא האזנה ל-Properties שלו, לכן כאן לאחר הגדרתו האזנתי לProgress Property , ובכל שינוי בו הורדתי את נקודות החיים של האובייקט ב-5. בקוד המשחק הגדרתי "בר חיים" שמאזין לחיים של כל אובייקט אליו הוא שייך ומשתנה בהתאם, בכך למעשה השגתי ירידה חלקה של בר החיים לפי הנזק שנגרם לאובייקט.

דוגמא נוספת:

```
1.   private void displayExplosion(int miliSeconds,String sprite_URL,int columns,int
    count,int offsetX,int offsetY,int height,int width,double startX,double startY) {
2.             ImageView img=new ImageView(new Image(sprite_URL));
3.             Task<Void> task=new Task<Void>() {
4.                     @Override
5.                     protected Void call() throws Exception {
6.                             img.setViewport(new Rectangle2D(offsetX ,offsetY,
    height, width));
7.                             img.setTranslateX(startX);
8.                             img.setTranslateY(startY);
9.                             Platform.runLater(()->{
10.                                     gamePane.getChildren().add(img);
11.                             });
12.                             for(int i=0;i<miliSeconds;i+=16) {
13.                                     double frac=(double)i/miliSeconds;
14.                                     final int
    index=Math.min((int)Math.floor(count*frac),count-1); //index is animation's
    current image,goes from 0 to count-1
15.                                     final int x=(index%columns)*width+offsetX;
16.                                     img.setViewport(new
    Rectangle2D(x,0,width,height)); //set new image at (x,0) with the given size
17.                                     Thread.sleep(16);
18.                             }
19.                             return null;
20.                     }
21.             };
22.             task.setOnSucceeded(e->{
23.                     gamePane.getChildren().remove(img);
24.             });
25.             new Thread(task).start();
26.   }
```

בדוגמא זו ה-Task למעשה מבצע אנימציה בהינתן Sprite Sheet ומידע עליו, אשר אורכת milliseconds מילי-שניות במקום (startX,startY) על המסך הראשי של המשחק. הקוד למעשה מחשב בכל 1/60 שניות את הפריים הבא של האנימציה (במידה ויש לעבור פריים-תלוי ביחס בין אורך האנימציה לכמות הפריימים) ומגדיר את התמונה בהתאם. נשים לב כי יש שימוש ב-Platform.runLater() בתוך ה-Task וזאת משום שהוספנו את האובייקט אל המסך הראשי (פעולה על GUI). למעשה הסיבה לשימוש ב-Task כאן נובעת משורות 22-24, הגדרתי אירוע כך שברגע שה-Task הסתיים בהצלחה,האובייקט יוסר מהמסך.

נקודה חשובה היא שכל Task שהוגדר הוא חד פעמי,כלומר ברגע שהוא בוצע לא ניתן לבצעו שוב. על מנת להגדיר Task רב פעמי, בחבילה javafx.concurrent קיימת המחלקה Service, שלמעשה מהווה Task רב פעמי.