תיכון ע"ש ב.אוסטרובסקי

מצוינות • מנהיגות • יזמות • חדשנות

פרויקט 5 יח"ל
בתכנון ותכנות מערכות
הגנת סייבר

בנייתמנוע בפיזיקה דו-מימדי:

שם התלמיד	:	נדב
שם משפחה	:	אלון
תעודת זהות	:	322952631
כיתה	:	י"ב 4

תיכון אוסטרובסקי רעננה

מורה מלווה : אתי בררו

קיץ 2019

תוכן עניינים

מבוא

יצרתי מנוע פיזיקה דו מימדי שמציג עצמים במרחב ואת ההתנגשויות שלהם אחד עם השני. לפרוייקט שלי יצרתי UI בסיסי וכמות פונקציות בשביל שהמשתמש יוכל להשפיע על העצמים במרחב.

בחרתי בפרוייקט הזה בגלל שרציתי לקבל ניסיון בעבודה עם מנוע פיזיקה וUI, כי אני חושב שלא לא תהיה לי הזדמנות אחרת לעבוד על פרוייקט גדול שמשלב אמצעים כאלו, כי רוב העבודות בתחום כוללות עבודה אחרת. אני כרגע עושה תואר ראשון במדעי המחשב באוניברסיטה הפתוחה, כדי לסיים אותו לפני הצבא ולקבל תפקיד טוב יותר בתחום. תמיד התעניינתי במדעי המחשב והיה לי מאוד חשוב לקבל תפקיד בנושא בשירות הצבאי שלי. לפי מה שידוע לי, לא משתמשים הרבה במנועי משחק במקצועות אחרים ממפתחי משחקים, ורציתי לחוות יצירה של מנוע פיזיקה כדי לצבור ניסיון על כך. העבודה הזאת הייתה הזדמנות לחוות יצירת מנוע פיזיקה שכזה.

כל הפיזיקה ורוב המתמטיקה במנוע מומשה לבד. בגלל שחלק מהמטרה שלי מהפרוייקט הייתה לאגור ניסיון בעבודה על פרוייקט גדול מאפס, רציתי לממש את כל המתמטיקה המסובכת לבד במקום להסתמך על ספריות אלגברה לינארית שיש ב-Java השתמשתי בספריות הסטנדרטיות של Java בשביל גרפיקה.

הדברים המיוצגים במנוע הם:

כדורים
קירות
כבידה

סביבת עבודה

סביבת העבודה שבחרתי לעבוד בא הייתה eclipse. בלימודי השפה עבדתי עם BlueJ, אך החלטתי שהסביבה הזאת לא מתאימה לפרוייקט בסדר הגודל הזה, אז למדתי איך עובדים עם eclipse כדי לעבוד ביעילות גבוהה יותר.

ניהול גרסאות התבצע בgithub, שם תועדה רוב העבודה על הפרוייקט.

מושגים בפרוייקט

המתמטיקה:
המתמטיקה בפרוייקט מסתמכת על מתמטיקת וקטורים, כלומר אלגברה לינארית, בשביל לייצג את הגדלים הווקטורים בפיזיקה כמו מהירות וכוח. מימשתי את המתמטיקה הוקטורית לבד, ואת כל הפונקציות המתמטיות שהיו דרושות לי.

הפיזיקה:
יצרתי קובץ יחיד שמאחד את כל הפונקציות הפיזיקליות כמו חישוב התנגשויות, חישוב כוחות, חישוב אנרגיה וכ"ו.

העצמים:
יצרתי אובייקט מופשט של "עצם", שאובייקטים שיורשים ממנו צריכים להשפיע על עצמים אחרים.
עצמים שיש במנוע הם:
projectile, כדור שנע במרחב. יש לו מיקום במרחב, מסה, מהירות ורדיוס.
wall, קיר מרובע סטטי שכדורים יכולים להתנגש בו. יש לו שני נקודות שמתארות את הנקודות החשובות שלו.
roundwall, קיר עגול סטטי שכדורים יכולים להתנגש בו. יש לו מיקום במרחב ורדיוס.

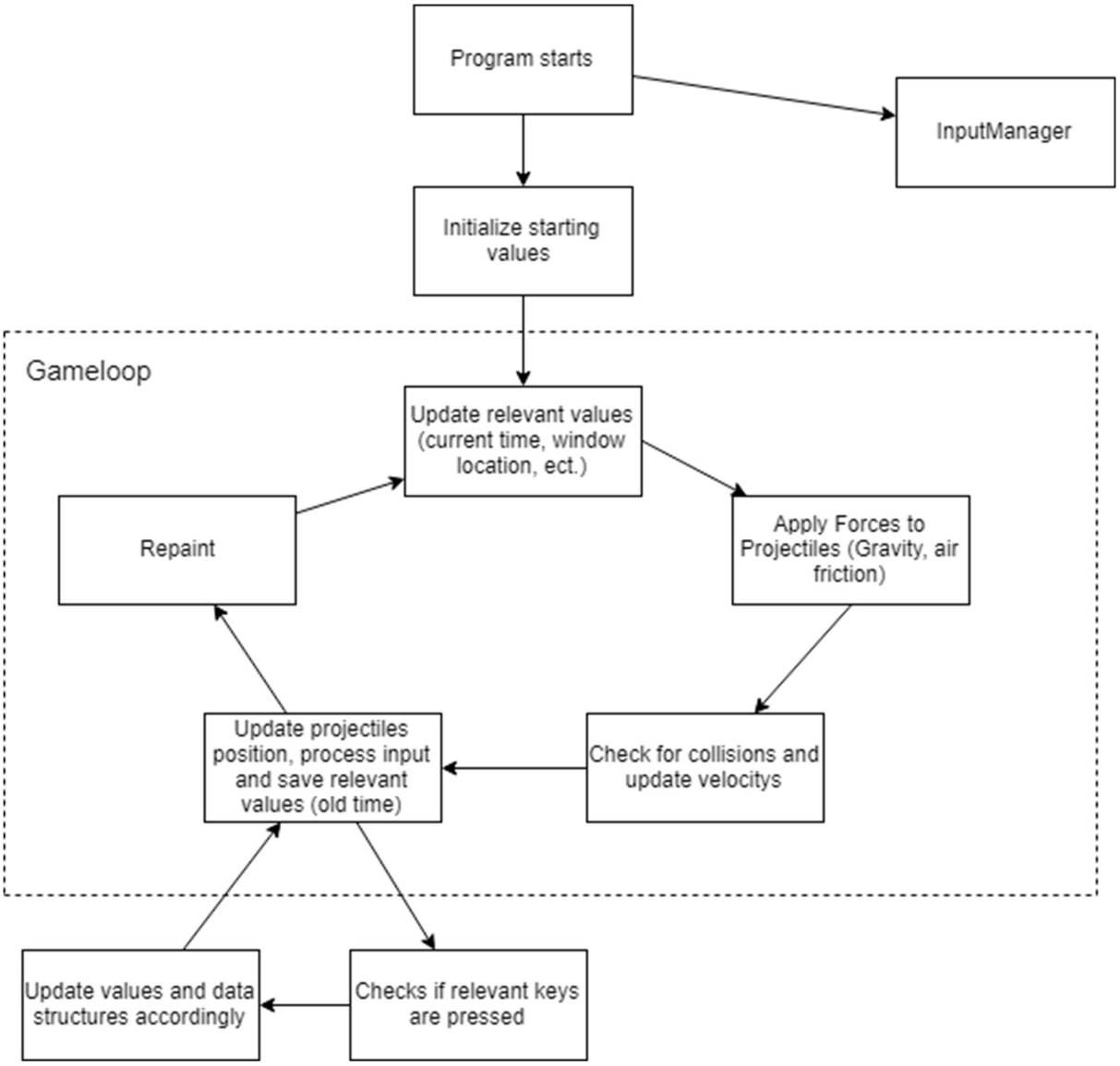
הקלט:
קובץ עיבוד קלט שמשתמש בספריות של java בשביל להקשיב למקלדת ולעכבר.

הגרפיקה:
רוב הגרפיקה ממומשת בקובץ putstuff ששם דברים על המסך, בשימוש בספריית הגרפיקה של Java. החלון הראשי של התכנה ממש את jpanel של java.

המain:
בmain ממומש הgameloop, ושם מוגדרים רוב הערכים והרשימות של המנוע עצמו.

תיאור פעולת התוכנה

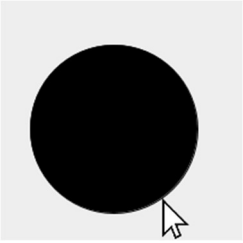
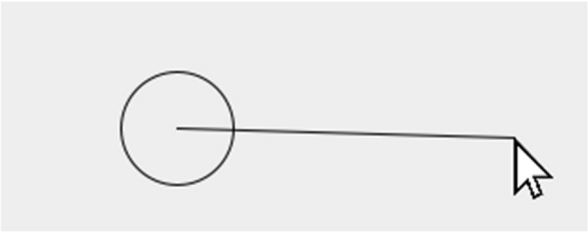
תיאור כללי:



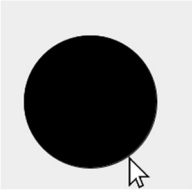
המערכת שבניתי מורכבת מלולאת משחק שרצה כל עוד החלון פתוח. כל התחלת לולאה, שומרים את הנתונים הרלוונטיים לחזרה הבאה, ומעדכנים את ערכי האובייקטים לפי המהירות\ הכוח המופעל עליהם, ומיקום כל האובייקטים. בסוף כל חזרה המסך נצבע מחדש לפי הנתונים החדשים.

תיאור הפעולות השונות למשתמש:

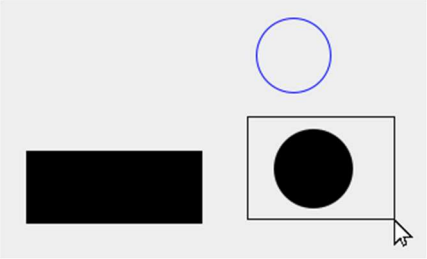
1. **הוספת כדור:**
המשתמש יכול להוסיף כדור בגודל שיבחר ע"י לחיצה על B במקלדת, וגרירת העכבר.
אלטרנטיבית, ע"י לחיצה על V, אפשר ליצור כדור חדש בגודל של הכדור האחרון שנוצר,
עם מהירות התחלתית ע"י גרירת העכבר בכיוון ההפוך לכיוון המהירות הרצוייה.

2. **הוספת קיר:**
המשתמש יכול להוסיף קיר מלבני ע"י לחיצת W או קיר מעגלי ע"י לחיצת M, וגרירת העכבר.

3. **מחיקת אובייקטים**:
המשתמש יכול למחוק אובייקטים מהמסך, ע"י לחיצת E והקפת האובייקטים הנמחקים במלבן שנוצר.




4. **שינוי הכבידה:**
בלחיצה על G, הכבידה תכבה ותדלק. בלחיצה על H ועל מקש חץ, ניתן לשנות את כיוון הכבידה.

5. **הילוך איטי**:
בלחיצה על S, הכדורים ינועו בהילוך פי 2 יותר איטי מבדרך כלל. לחיצה שנית על S תחזיר את המצב להילוך רגיל.

6. **הקפאת המצב**:
בלחיצה על F, הכדורים יפסיקו לזוז.

7. **החזרת הזמן אחורה וקדימה**:
כאשר המצב קפוא (F נלחץ), ניתן ללחוץ על המקשים > ו< בשביל להעביר את המצב עשירית שנייה אחורה, עד 10 שניות לפני הקפאת המצב, או קדימה עד המצב הקפוא.

8. **הפסקת המצב**:
בלחיצה על P, המשחק יעצור לגמרי, והמשתמש לא יוכל להשפיע על המצב בכלל.

תיאור מפורט:

קובץ ראשי:

כשהמשתמש מתחיל את התוכנה, הmain יוצר jframe חדש, שיהיה החלון של התוכנה.
מבוצע איתחול של כמה ערכים שדרושים בשביל התוכנה, ושמירת ערכים מסויימים כמו זמן המערכת וגודל החלון.
אחר כך, מבוצע שימוש בtimer בשביל ליצור gameloop שחוזר על עצמו בקצב קבוע.

```
public static void main(String[] args)
{
    for (int i = 0 ; i < key.length ; i++) {
        key[i] = false;}
    inintilizeProj();
    oldT = System.currentTimeMillis();

    JFrame frame = new JFrame("game");


    frame.add(attempt);
    frame.setSize(1800, 1000);  //setting window size
    frame.setVisible(true);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    oldHeight = 1000;
    oldWidth = 1800;



    //int flipXcnt = 1;
    //int flipYcnt = 1;
    TimerTask gameloop = new gameloop(attempt);
    Timer timer = new Timer(true);
    attempt.add(ipText);
    timer.scheduleAtFixedRate(gameloop, 0, FPS);    //setting fps
}
```

לולאת המשחק מתחילה בבדיקת גודל המסך, ואיתחול מחדש של ערכי הקירות אם גודל המסך שונה. זאת משום שלא ניתן להשאיר את הקירות במקום כאשר משנים את גודל המסך.
שומרים את הזמן החדש, ואת הזמן שעבר בין הפעם האחרונה שבדקנו זמן לעכשיו במילישניות, בשביל לחשב את הפיזיקה כתלוי בכך. אם מצב המשחק עצור, אז השינוי בזמן יהיה 0 כך שכדורים לא ינועו.

לאחר מכן, הכבידה פועלת על כל הכדורים שיש במנוע, ולכל קיר שיש במנוע יש בדיקת התנגשות עם כל כדור במנוע.

```
public void run()
{

    windowLocation = attempt.getLocation();
    if (oldHeight != attempt.getHeight() || oldWidth != attempt.getWidth())
        initilizeWall();
    oldHeight = attempt.getHeight();
    oldWidth = attempt.getWidth();

    long newT = System.currentTimeMillis(); //gets new time from the system.
    if (CurMode == mode.PAUSE)
        oldT = System.currentTimeMillis();  //if paused, make it so time doesnt pass


    long deltaT = newT - oldT;              //gets the difference of the times between last frame and now.

    if (CurMode != mode.PAUSE)
        ipText.setText("");

    // Upply gravity and friction to all projectiles.
    if (!pro.isEmpty())
        Physics.upplyG(pro, proSize);
    //Physics.upplyFric(pro, 2);


    for (int i = 0; i < proSize; i++) // Check all combination of items that can collide with each other
    {
        for (int j = 0; j < wallSize; j++)
        {
            if (Physics.areColliding((Item)pro.get(i),(Item)walls.get(j)))
            {
                totalBounce++;
                System.out.println("bounce wall");
                Physics.collision(pro.get(i),(Item)walls.get(j));
            }
        }
    }
```

לכל שני כדורים יש בדיקת התנגשות אחד עם השני. לאחר הוספת כל הכוחות וכל ההתנגשויות, מוסיפים לכל כדור את המרחק שעבר בזמן הזה לפי ערך המהירות שלו. לאחר מכן יש בדיקה של כל כדור אם הוא מחוץ לגבולות המגרש, והחזרתו אם כן.
לאחר מכן יש ציור מחדש של המסך ועיבוד מחדש של הקלט, ושמירת הזמן הנכחי.

```
    for (int i = 0; i < proSize; i++) // Check all combination of items that can collide with each other
    {

        for (int j = i+1; j < proSize; j++)
        {

            if (Physics.areColliding((Item)pro.get(i),(Item)pro.get(j)))
            {
                totalBounce++;
                System.out.println("bounce");
               Physics.collision(pro.get(i),pro.get(j));
            }
        }
    }

    for (int i = 0; i < proSize; i++) // apply speed to projectiles.
    {
        pro.get(i).cord1._x += deltaT*pro.get(i)._vel.getX()/1000;  //divide by 1000 because messured by milliseconds.
        pro.get(i).cord1._y += deltaT*pro.get(i)._vel.getY()/1000;
        totalDist += deltaT*pro.get(i)._vel.getSize()/1000;

        if (pro.get(i).cord1._x < ((Wall)walls.get(1)).cord2._x)
            pro.get(i).cord1._x = ((Wall)walls.get(1)).cord2._x + pro.get(i)._rad - 0.5;

        if (pro.get(i).cord1._x > walls.get(2).cord1._x)
            pro.get(i).cord1._x = walls.get(2).cord1._x - pro.get(i)._rad + 0.5;

        if (pro.get(i).cord1._y < walls.get(0).cord1._y)
            pro.get(i).cord1._y = walls.get(0).cord1._y + pro.get(i)._rad - 0.5;

        if (pro.get(i).cord1._y > ((Wall)walls.get(3)).cord2._y)
            pro.get(i).cord1._y = ((Wall)walls.get(3)).cord2._y - pro.get(i)._rad + 0.5;
    }


    at.repaint();       //repaint the screen
    processInput();
    oldT = newT;        //update oldT to newT to remember this frames time for the next frame.
}
```

הקלט עובד כך: כאשר המקלדת או העכבר עושים משהו, inputmanager שמקשיב להם מעדכן את הערכים שלהם. יש משתנה למיקום העכבר, ומערך בוליאני של אם כל מקש במקלדת לחוץ ברגע מסוים. כאשר inputmanager רואה שמקש לחוץ הוא מעדכן את המערך כדי לשקף זאת, וגם כשהוא רואה שמקש שוחרר.
פונקציית עיבוד הקלט היא פונקציה שמבצעת דברים מסויימים כתלוי במקשי המקלדת הלחוצים ברגע הזה. כאשר היא מקבלת את המקשים של החלפת מצב, היא משנה את ערך המשתנה curmode, וכאשר היא רואה שהעכבר נלחץ או שוחרר היא שומרת את הערכים שלו ומבצעת את מה שנדרש כתלוי במצב הנכחי.

דוגמא:

כשהעכבר נלחץ, הפונקציה שומרת את המיקום הנוכחי שלו בstartLocation. כאשר יש ערך בstartLocation (כלומר העכבר נלחץ ועוד לא שוחרר) וגם העכבר לא לחוץ עכשיו (כלומר העכבר בדיוק שוחרר), וגם המצב הנכחי זה כדור, אז זה יוסיף כדור באמצע שני הנקודות startLocation וendLocation, עם רדיוס של חצי המרחק ביניהם.

```
if (mousePressed && startLocation == null)
    startLocation = new Coord(mouseLocation);

if (!mousePressed && startLocation != null)
{
    endLocation = new Coord(mouseLocation);
    switch(CurMode)
    {
        case BALL:
        {
            double rad = Physics.CoordDist(startLocation, endLocation)/2;
            Coord cent = Physics.findMiddle(startLocation, endLocation);
            pro.add(new Proj(cent, rad));
            proSize++;
            break;
        }
```

קבצי מתמטיקה:

מחלקה שמתארת וקטור במישור:

```
public class Vect
{
 private double _size;
 private float _dir;

 /**
  * Constructor for vector using size and direction in radians.
  * to use this constructor you must use a float as d.
  */
 public Vect(double s, float d)
 {
     if (s < 0)  // Size must not be negetive.
     {
         System.out.println("Error: size smaller then 0 given to vector constructor.");
         s = 0;
     }
     else if (s > 0) // Makes sure the angle is in [0,2pi] to ease calculations.
     {
         while (d > 2*Math.PI)
         {
             d -= 2*Math.PI;
         }
         while (d < 0)
         {
             d += 2*Math.PI;
         }
     }
     else
         d = 0;

     _size = s;
     _dir = d;
 }
```

במימוש וקטורים שלי, הערכים הנשמרים הם הכיוון והגודל של הוקטור ולא ערכי הx והy. קיימים גם יוצרים שמשתמשים בx וy, וניתן להשיג אותם בעזרת פונקציות במחלקה. מימוש זה גורם לכל כיוון מדוייק להיות לא מדוייק.

הסבר: בגלל שהערך שנשמר הוא הכיוון, והכיוון נשמר ברדיאנים, כיוונים כמו היישר למעלה יהיו אי-רציונלים ולכן יהיה אי-דיוק מסויים כאשר ננסה לקבל אותם.

מחלקה שמתארת מטריצה דו-מימדית:

```
public class Matrix
{
    public Vect _v1;    //vertical vectors
    public Vect _v2;

    public Matrix(double x1, double y1, double x2, double y2)
    {
        _v1 = new Vect(x1,y1);
        _v2 = new Vect(x2,y2);
    }

    public Matrix(Vect v1, Vect v2)
    {
        _v1 = new Vect(v1);
        _v2 = new Vect(v2);
    }
```

המטריצה נבנית ומתוחזקת ע"י וקטורי השורה שלה, אך יש באמצעותה לקבל את וקטורי העמודה שלה עם פונקציות שבה.

מחלקה שבא נשמרות פונקציות המתמטיקה על וקטורים:

```
public abstract class Vec_Math
{
    //dot product between 2d vectors.
    public static double dot_prod(Vect A, Vect B)
    {
        return A.getX() * B.getX() + A.getY() * B.getY();
    }

    //scalar multiplication.
    public static void sizeMult (Vect A, double s)
    {
        A.sizeMult(s);
    }

    //returns a new vector instead of changing the given vector.
    public static Vect retSizeMult (Vect A, double s)
    {
        Vect temp = new Vect(A);
        temp.sizeMult(s);
        return temp;
    }
```

פונקציות חשובות במחלקה:

public static Vect vectAdd(Vect A, Vect B)

מחזיר את סכום שני הוקטורים.

public static void flipYaxis(Vect A)

הופך את הוקטור הנתון על ציר y.

public static Vect transform(Matrix a, Vect b)

מבצע טרנספורמציה לינארית על הוקטור ע"י המטריצה ומחזיר את הוקטור המתקבל.

קבצי אובייקטים:

מחלקה שמתארת נקודה במישור:

```
public class Coord
{
    public double _x;
    public double _y;

    public Coord(double x, double y)
    {
        _x = x;
        _y = y;
    }


    public Coord(Coord c)
    {
        _x = c._x;
        _y = c._y;
    }

    public Coord intoJcoord()
    {
        return new Coord(_x, attempt.attempt.getHeight() -_y);
    }
}
```

מחלקה אבסטרקטית שמתארת עצם במישור שמשפיע על עצמים אחרים:

```
public abstract class Item
{
    Coord cord1;


    public Item()
    {
        cord1 = new Coord(0,0);
    }
    public Item(double x, double y)
    {
        cord1 = new Coord(x,y);
    }

    public Item(Coord c)
    {
        cord1 = new Coord(c);
    }

    public abstract boolean isCol(Item other);  // Items on the screen must be able to interact with eachother.
}
```

מחלקה שמתארת כדור עם מהירות, רדיוס ומסה:

```
public class Proj extends Item
{

    // X and Y from Item represent top left of cube containing the ball
    public Vect _vel = new Vect(0,0);
    public double _rad;
    public double _mass;



    public Proj(Coord c, double rad)
    {
        super(c);
        _rad = rad;
        _mass = Math.PI * Math.pow(_rad,2);
    }
```

פונקציות נוספות במחלקה:

public boolean isCol(Item other)

פונקציה שמקבלת עצם אחר ובודקת אם הוא והעצם האחר מתנגשים (לכדור אחר יש השוואת המרחק של מרכזיהם לבין סכום הרדיוסים שלהם, ולקיר יש השוואת כל הקואורדינטות של הכדור לפינות של הקיר ובדיקת התנגשות).

מחלקה שמתארת קיר מלבני במישור:

```
public class Wall extends Item
{
    // X and Y of Item are the top left point. W and Z here are the bottom right point.
    public Coord cord2;
    /**
     * Constructor for objects of class Wall
     */


    public Wall(double x, double y, double w, double z)
    {
        super(x,y);
        cord2 = new Coord(w,z);
    }
```

פונקציות נוספות במחלקה:

public boolean isCol(Item other)

כדומה למימוש בכדור.

מחלקה שמתארת קיר עגול במישור:

```
public class RoundWall extends Item
{

    public double _rad;

    //constructor using x,y coordinates and a radius.
    public RoundWall(double x, double y, double rad)
    {
        super(x,y);
        _rad = rad;
    }

    public RoundWall(Coord c, double rad)
    {
        super(c);
        _rad = rad;
    }
```

פונקציות אחרות במחלקה:

public boolean isCol(Item other)

כדומה למימוש בכדור.

קובץ הפיזיקה

```
public class Physics
{
    public static Vect grav = new Vect(900, (float)(3*Math.PI/2));
    public static double airFric = 0.8;

    public Physics()
    {
        grav = new Vect(900, (float)(3*Math.PI/2));
    }
```

הקובץ בו פונקציות החישוב הפיזיקליות נמצאות, ובנוסף אליהן כמה פונקציות נספחות אחרות.
הקובץ כולל את הפונקציות הבאות:

public static void upplyF(Proj p, Vect f)
מפעיל כוח על כדור (משנה את המהירות של הכדור בהתאם לכוח ולמסת הכדור).

public static void upplyG(LinkedList<Proj> p, int n)
מפעיל את הכבידה הנכחית על רשימת כדורים.

public static void collision(Proj a, Item b)
מחשב ומיישם את המהירות החדשה של עצמים לאחר התנגשותם.

public static boolean isOverlap(Proj a, Proj b)
+
public static void fixOverlap(Proj a, Proj b)
בודק אם יש חפיפה בין שני כדורים, ומתקן אם יש. (קיימים גרסאות שונות לכל שני אובייקטים שעלולים לחפוף).

קובץ עיבוד נתונים

```
public class InputManager
{
    //static int action;

    public class MyActionListener implements KeyListener, MouseListener, MouseMotionListener {

        @Override
        public void keyTyped(KeyEvent e)
        {}
        @Override
        public void keyPressed(KeyEvent e) {
            switch (e.getExtendedKeyCode()) {
            case (KeyEvent.VK_UP):
                attempt.key[attempt.keyCode.UP.code] = true;
                break;
            case (KeyEvent.VK_DOWN):
                attempt.key[attempt.keyCode.DOWN.code] = true;
                break;
```

מממש את הממשקים הסטנדרטיים של ג'אווה לקבלת קלט דרך העכבר והמקלדת.
קבלת קלט מהמקלדת עובד כך: יש מערך בוליאני בקובץ הראשי שמתאר כל לחצן במקלדת ואם הוא לחוץ. כשהinputlistener "שומע" שמקש מקלדת נלחץ, הוא שם בתא המתאים במערך את הערך אמת, וכאשר מקש מקלדת נעזב הוא שם בתא המתאים שקר.
כדומה לכך, כשהוא שומע שהעכבר זז ונלחץ הוא שולח את המקום למשתנים המתאימים בקובץ הראשי.

קובץ הגרפיקה

```
public abstract class Putstuff
{
    public static void putProj(Proj p, Graphics2D g2d)
    {
        int x = (int)p.cord1.intoJcoord()._x;
        int y = (int)p.cord1.intoJcoord()._y;
        g2d.drawOval(x - (int)p._rad , y - (int)p._rad, (int)p._rad*2, (int)p._rad*2);
    }

    public static void putWall(Wall w, Graphics2D g2d)
    {
        int x = (int)w.cord1.intoJcoord()._x;
        int y = (int)w.cord1.intoJcoord()._y;
        g2d.fillRect(x, y, (int)w.getLength(),  (int)w.getHeight());
    }

    public static void putErase(Wall w, Graphics2D g2d)
    {
        int x = (int)w.cord1.intoJcoord()._x;
        int y = (int)w.cord1.intoJcoord()._y;
        g2d.drawRect(x, y, (int)w.getLength(),  (int)w.getHeight());
    }

    public static void putRoundwall(RoundWall w, Graphics2D g2d)
    {
        int x = (int)w.cord1.intoJcoord()._x;
        int y = (int)w.cord1.intoJcoord()._y;
        g2d.fillOval(x - (int)w._rad , y - (int)w._rad, (int)w._rad*2, (int)w._rad*2);
    }
}
```

שם דברים על המסך.

סיכום

אם הייתי מתחיל היום, כנראה הייתי מארגן את הקוד ביעילות גבוהה יותר, עם הידע החדש שלי בנוגע לגרפיקה בJava והספריות שאני עובד איתן. בנוסף, כנראה הייתי משתמש בספריות בשביל המתמטיקה שאני משתמש בא, כי אני מתאר לעצמי שהספרייה שלי לא יעילה במיוחד ושהתכנה תרוץ ביעילות רבה יותר עם ספרייה חיצונית.

הייתי דואג יותר ליצור כלי משתמש טובים יותר, כדי שהמנוע יוכל לשמש מנוע משחק אמיתי שאפשר לבנות עליו משחקים, במקום תכנת סימולציה של פיזיקה דו-מימדית.

הפרוייקט לימד אותי על שימוש בספריות, תכנון כלים עצמיים ותיעוד עצמי. עבודה על הפרוייקט עזרה לי ללמוד שימושים בgithub וeclipse, והעבודה על פרוייקט ארוך שכולל בו צורך בתיעוד נרחב עזר לי לקבל מבט טוב יותר על העבודה שאעשה בצבא ובתעשייה.

קשיים קיימים:

במהירויות מספיק גבוהות ובקירות מספיק דקים, כדורים יכולים לעבור דרך קירות או לצאת מגבולות המשחק. זה קורה בגלל שבמהירות גדולה כל כך, כשמוסיפים אותה לכדור הוא כבר עובר את הקיר שהוא אמור להתנגש בו. בוצע תיקון מהיר ליציאה מגבולות המשחק בכך שכשמשחק רואה שזה קרה הוא מחזיר את הכדור לגבולות המשחק במקום שהוא אמור להיות בו, אך תיקון של מעבר דרך הקירות הדקים יקח זמן רב ופונקציות חדשות, ואפילו אולי דרך חדשה לחשב את המיקום של העצמים במסך.

כדורים יכול להכנס אל תוך קיר, ולא יכולו לצאת בלי עזרה חיצונית. זה קורה כשמזמנים אותם בתוך הקיר, או כשכדור אחר מתנגש בהם כשהם כבר נוגעים בקיר, דבר שגורם לפונקציית הfixOverlap להכניס את הכדור לתוך הקיר. יש כמה פונקציות שנועדו לתקן את המצב הזה, אך הוא עוד קורא לעיתים, בעיקר כשהקירות קטנים מדי לכמות הכדורים שבהם.

כאשר המסך קפוא (מצב שבו כל הכדורים אמורים להפסיק לזוז) אך כדור נמצא בתוך קיר, לפעמים הוא עוד יזוז, כי לולאת המשחק עוד רצה ולכן פונקציות ההתנגשות עדיין נקראות כמו אם המסך לא היה קפוא, וגם פונקציות fixOverlap נקראות כדי לנסות לתקן את החפיפה.

מימוש הוקטורים גורם לתזוזה מדוייקת להיות בלתי אפשרית, כמו שתואר בתיאור הקוד, ולכן תזוזה ישרה למעלה, למטה ושמאלה בלתי אפשרית (תזוזה ישרה ימינה כן אפשרית כי ימין מיוצג ב0 והשאר במספרים אי-רציונלים). ניתן לתקן את הבאג הזה אם נייצג את הוקטורים כ(x,y), אך מימוש זה יכול לגרום לבעיות, ואין ברצוני לתקן אותו כי הוא לא משפיע מאוד על הדיוק הכולל של הסימולציה.

במהלך יצירת הפרוייקט, התהליך שלקח לי הכי הרבה זמן זה מימוש ההתנגשויות. בהתחלה מימשתי את מערכת הצירים כמו שהיא ממומשת בjpanel, כלומר כשהציר האנכי מתחיל למעלה ויורד למטה. דבר זה יצר הרבה בעיות בתזוזה, ומהירות, ומאוד פגע במימוש ההתנגשויות שלי כי ההתנגשות שחישבתי הייתה תמונת מראה של ההתנגשות שחישבתי. בסופו של דבר החלטתי לתקן את מימוש מערכת הצירים, ובכך פישטתי מאוד את העבודה על התנגשויות וייצוג עצמים.

ביבליוגרפיה

https://docs.oracle.com/javase/7/docs/api/
אתר התיעוד של Java, שם מצאתי את הפונקציות הדרושות שלי לרוב הפרוייקט.

רשימת הספריות שהשתמשתי בהן:

ספריות listener לקלט:
- java.awt.event.KeyListener
- java.awt.event.KeyEvent
- java.awt.event.MouseListener
- java.awt.event.MouseEvent
- java.awt.event.MouseMotionListene

דברים לגרפיקה, לחלון ולUI:
- java.awt.Graphics
- java.awt.Graphics2D
- java.awt.Point
- java.awt.RenderingHints
- javax.swing.JFrame
- javax.swing.JPanel
- javax.swing.JTextField
- java.awt.Color
- java.awt.Font

java.util.LinkedList	רשימה מקושרת בשביל מבנה נתונים דינאמי
java.util.Timer	כדי לשמור את זמן המערכת
java.util.TimerTask	כדי לתזמן אירועים מראש (לנעול את הfps של המנוע)

https://gamedev.stackexchange.com/questions/56017/java-best-implementation-keylistener-for-games
אתר שעזר לי להבין את העקרון של לולאת משחק ושל KeyListener.

https://en.wikipedia.org/wiki/Elastic_collision
עמוד ויקיפדיה שתיאר את הנוסחאות המתמטיות הדרושות להתנגשות דו מימדית.