```java
import danogl.*;
import danogl.collisions.Layer;
import danogl.components.*;
import danogl.gui.*;
import danogl.gui.rendering.*;
import danogl.util.Vector2;

import java.awt.*;
import java.awt.event.KeyEvent;

/**
 * A simple platformer demo with a circle as an avatar and a few platforms.
 * Move with left and right keys, jump with space, down+space to drop down a platform.
 * @author Dan Nirel
 */
public class Platformer extends GameManager {
    private static final Color BACKGROUND_COLOR = Color.decode("#80C6E5");
    private static final Color PLATFORM_COLOR = new Color(212, 123, 74);

    @Override
    public void initializeGame(ImageReader imageReader, SoundReader soundReader, UserInputListener inputListener, WindowController windowController) {
        super.initializeGame(imageReader, soundReader, inputListener, windowController);

        //background
        var background =
            new GameObject(
                    Vector2.ZERO,
                    windowController.getWindowDimensions(),
                    new RectangleRenderable(BACKGROUND_COLOR)
            );
        background.setCoordinateSpace(CoordinateSpace
```

```
31 .CAMERA_COORDINATES);
32         gameObjects().addGameObject(background, Layer
   .BACKGROUND);
33
34         placePlatform(Vector2.of(-1024, 1000),
   Vector2.ONES.mult(2048));
35         placePlatform(Vector2.of(-512, 700), Vector2.
   of(1024, 50));
36         placePlatform(Vector2.of(-256, 400), Vector2.
   of(512, 50));
37         placePlatform(Vector2.of(-128, 100), Vector2.
   of(256, 50));
38
39         var avatar = new Avatar(Vector2.of(0, 900),
   inputListener);
40         setCamera(new Camera(avatar, Vector2.ZERO,
41                 windowController.getWindowDimensions
   (), windowController.getWindowDimensions()));
42         gameObjects().addGameObject(avatar);
43     }
44
45     private void placePlatform(Vector2 pos, Vector2
   size) {
46         var platform = new GameObject(pos, size, new
   RectangleRenderable(PLATFORM_COLOR));
47         platform.physics().
   preventIntersectionsFromDirection(Vector2.UP);
48         platform.physics().setMass(GameObjectPhysics.
   IMMOVABLE_MASS);
49         gameObjects().addGameObject(platform, Layer.
   STATIC_OBJECTS);
50     }
51
52     public static void main(String[] args) {
53         new Platformer().run();
54     }
55 }
56
57 class Avatar extends GameObject {
58     private static final float VELOCITY_X = 400;
59     private static final float VELOCITY_Y = -650;
```

```
60      private static final float GRAVITY = 600;
61      private static final Color AVATAR_COLOR = Color.
    DARK_GRAY;
62
63      private UserInputListener inputListener;
64
65      public Avatar(Vector2 pos, UserInputListener
    inputListener) {
66          super(pos, Vector2.ONES.mult(50), new
    OvalRenderable(AVATAR_COLOR));
67          physics().preventIntersectionsFromDirection(
    Vector2.ZERO);
68          transform().setAccelerationY(GRAVITY);
69          this.inputListener = inputListener;
70      }
71
72      @Override
73      public void update(float deltaTime) {
74          super.update(deltaTime);
75          float xVel = 0;
76          if(inputListener.isKeyPressed(KeyEvent.
    VK_LEFT))
77              xVel -= VELOCITY_X;
78          if(inputListener.isKeyPressed(KeyEvent.
    VK_RIGHT))
79              xVel += VELOCITY_X;
80          transform().setVelocityX(xVel);
81          if(inputListener.isKeyPressed(KeyEvent.
    VK_SPACE) && getVelocity().y() == 0)
82              transform().setVelocityY(VELOCITY_Y);
83      }
84  }
```

```java
package pepse;

import java.awt.*;
/**
 * A class containing constants used throughout the game.
 * @author Eilam Soroka, Maayan Felig
 */
public class Constants {
    //Game
    public static final float CYCLE_LENGTH = 30.0f;

    //sky
    public static final Color BASIC_SKY_COLOR = Color.decode("#80C6E5");
    public static  final String SKY_TAG = "sky";

    //Block
    public static final int BLOCK_SIZE = 30;

    //Terrain
    public static final int START_x0 = 0;
    public static final double FACTOR_TERRAIN = 7;
    public static final double NOISE_FACTOR_TERRAIN = BLOCK_SIZE * FACTOR_TERRAIN;
    public static final Color BASE_GROUND_COLOR = new Color(212, 123, 74);
    public static final int TERRAIN_DEPTH = 20;
    public static final String TOP_LAYER_GROUND_BLOCK_TAG = "topGroundBlock";
    public static final String LOWER_LAYER_GROUND_BLOCK_TAG = "lowerGroundBlock";
    public static final float GROUND_HEIGHT_FACTOR = (2/3.0f);
    public static final int MAX_ENERGY = 100;
//    public static final String FRUIT_TAG = "fruit";

    //Fruit
    public static final String FRUIT_TAG = "fruit";

    //Avatar
```

```
35      public static final String AVATAR_TAG = "avatar";
36
37      //Tree
38      public static final String TRUNK_BLOCK_TAG = "
    trunk_block";
39      public static final float WIND_EFFECT_DIFFERENCES
     = 2.0f;
40
41
42      public static final String LEAF_TAG = "leaf";
43      public static final int FRUIT_ENERGY = 10;
44  }
45
```

```java
package pepse;
import danogl.GameObject;
import danogl.collisions.Layer;
import danogl.components.ScheduledTask;
import danogl.gui.rendering.Camera;
import danogl.util.Vector2;
import pepse.world.Block;
import pepse.world.EnergyDisplay;
import pepse.world.Sky;

import danogl.GameManager;
import danogl.gui.ImageReader;
import danogl.gui.SoundReader;
import danogl.gui.UserInputListener;
import danogl.gui.WindowController;
import pepse.world.Terrain;
import pepse.world.avatar.Avatar;
import pepse.world.daynight.Moon;
import pepse.world.daynight.Night;
import pepse.world.daynight.Sun;
import pepse.world.daynight.SunHalo;
import pepse.world.trees.Flora;
import pepse.world.trees.Fruit;
import pepse.world.trees.Leaf;
import pepse.world.trees.Tree;

import java.util.List;

import static pepse.Constants.BLOCK_SIZE;

/**
 * The main class of the Pepse game.
 * incharge of initializing and running the game.
 * @author Eilam Soroka, Maayan Felig
 */
public class PepseGameManager extends GameManager {


    private final java.util.Map<Integer, java.util.
List<ObjInLayer>> loadedZones =
            new java.util.HashMap<>();
```

```
41
42
43      private static final int FRUIT_LAYER = Layer.
    FOREGROUND + 1;
44      private static final int SKY_LAYER = -250;
45      private static final int SEED = 73;
46      private static final int CYCLE_OF_DAY_LENGTH = 10
    ;
47      private static final String TOP_LAYER_TAG = "
    topGroundBlock";
48      private static final int SUN_LAYER = -225;
49      private static final float FIRST_X_POSITION = 0f;
50      private float groundHeightAtX0;
51      private Avatar avatar; // todo check if it is ok
    that i hold the avatar object here
52      private Terrain terrain;
53      private Flora flora;
54      private float zoneWidth;
55      private int currentZoneIdx;
56
57      /**
58       * Initializes the game by setting up the sky and
     terrain.
59       *
60       * @param imageReader     Provides functionality
    for reading images.
61       * @param soundReader     Provides functionality
    for playing sounds.
62       * @param inputListener   Listens for user input.
63       * @param windowController Controls the game
    window.
64       */
65      @Override
66      public void initializeGame(ImageReader
    imageReader, SoundReader soundReader,
    UserInputListener inputListener,
67                                 WindowController
    windowController) {
68          super.initializeGame(imageReader, soundReader
    , inputListener, windowController);
69          gameObjects().layers().shouldLayersCollide(
```

```
69 Layer.FOREGROUND,Layer.STATIC_OBJECTS,true);
70         gameObjects().layers().shouldLayersCollide(
   Layer.FOREGROUND,FRUIT_LAYER,true);
71         initializeSky(windowController);
72 
73         terrain = new Terrain(windowController.
   getWindowDimensions(), SEED);
74         flora = new Flora(SEED, terrain::
   groundHeightAt);
75         this.groundHeightAtX0 = Terrain.
   groundHeightAtX0(windowController.
   getWindowDimensions());
76         initializeNight(windowController);
77         initializeSun(windowController);
78         initializeMoon(windowController);
79 
80 
81 
82         initializeAvatar(new Vector2(
   FIRST_X_POSITION,groundHeightAtX0),inputListener,
   imageReader,
83                 windowController);
84         initializeEnergyDisplay(windowController);
85 
86         zoneWidth = windowController.
   getWindowDimensions().x();
87         currentZoneIdx = worldXToZone(avatar.
   getCenter().x());
88         loadZone(currentZoneIdx - 1);
89         loadZone(currentZoneIdx);
90         loadZone(currentZoneIdx + 1);
91     }
92 
93     /**
94      * Updates the game state, loading and unloading
    zones as the avatar moves.
95      *
96      * @param deltaTime Time elapsed since the last
   update.
97      */
98     @Override
```

```
 99     public void update(float deltaTime) {
100         super.update(deltaTime);
101 
102         int newZone = worldXToZone(avatar.getCenter
    ().x());
103         if (newZone == currentZoneIdx) return;
104 
105         if (newZone > currentZoneIdx) {
106             // moved right: drop leftmost, add new
    rightmost
107             unloadZone(currentZoneIdx - 1);
108             loadZone(newZone + 1);
109         } else {
110             // moved left: drop rightmost, add new
    leftmost
111             unloadZone(currentZoneIdx + 1);
112             loadZone(newZone - 1);
113         }
114 
115         currentZoneIdx = newZone;
116         System.out.println("loadedZones=" +
    loadedZones.keySet());
117 
118     }
119 
120 
121     private void loadZone(int zoneIdx) {
122         if (loadedZones.containsKey(zoneIdx)) return
    ;
123 
124         int start = zoneStartX(zoneIdx);
125         int end = zoneEndX(zoneIdx);
126 
127         java.util.List<ObjInLayer> created = new
    java.util.ArrayList<>();
128 
129         for (Block block : terrain.createInRange(
    start, end)) {
130             int layer = TOP_LAYER_TAG.equals(block.
    getTag())
131                     ? Layer.STATIC_OBJECTS
```

```
132                     : Layer.BACKGROUND;
133             gameObjects().addGameObject(block, layer
    );
134             created.add(new ObjInLayer(block, layer
    ));
135         }
136
137         List<Tree> trees = flora.createInRange(start
    , end);
138         for (Tree tree : trees) {
139             for (Block trunk : tree.getTrunkBlocks
    ()) {
140                 int layer = Layer.FOREGROUND + 1;
    // you used this for trunk previously
141                 gameObjects().addGameObject(trunk,
    layer);
142                 created.add(new ObjInLayer(trunk,
    layer));
143             }
144             for (Leaf leaf : tree.getAllLeaves()) {
145                 int layer = Layer.FOREGROUND;
146                 gameObjects().addGameObject(leaf,
    layer);
147                 created.add(new ObjInLayer(leaf,
    layer));
148             }
149             for (Fruit fruit : tree.getAllFruit()) {
150                 int layer = FRUIT_LAYER;
151                 gameObjects().addGameObject(fruit,
    layer);
152                 created.add(new ObjInLayer(fruit,
    layer));
153             }
154
155         }
156         loadedZones.put(zoneIdx, created);
157     }
158
159     private void unloadZone(int zoneIdx) {
160         java.util.List<ObjInLayer> objs =
    loadedZones.get(zoneIdx);
```

```
161         if (objs == null) return;
162
163         for (ObjInLayer ref : objs) {
164             gameObjects().removeGameObject(ref.obj,
    ref.layer);
165         }
166         loadedZones.remove(zoneIdx);
167     }
168
169
170     private void initializeTrees(WindowController
    windowController) {
171         List<Tree> trees = flora.createInRange(0, (
    int) windowController.getWindowDimensions().x()); //
    todo change range
172         for (Tree tree : trees) {
173             List<Block> trunkBlocks = tree.
    getTrunkBlocks();
174             for (Block block : trunkBlocks) {
175                 gameObjects().addGameObject(block,
    Layer.FOREGROUND + 1);
176             }
177             List<Leaf> leaves = tree.getAllLeaves();
178             for (Leaf l : leaves) {
179                 gameObjects().addGameObject(l, Layer
    .FOREGROUND);
180             }
181             List<Fruit> fruits = tree.getAllFruit();
182             for (Fruit f : fruits) {
183                 gameObjects().addGameObject(f,
    FRUIT_LAYER);
184             }
185
186         }
187     }
188
189
190     private void initializeEnergyDisplay(
    WindowController windowController) {
191         new EnergyDisplay(
192                 avatar::getEnergyMeter,
```

```
193                 (gameObject, layer) -> gameObjects
    ().addGameObject(gameObject, layer)
194         );
195     }
196 
197     private void initializeAvatar(Vector2
    topBlockAtX0, UserInputListener inputListener,
198                                   ImageReader
    imageReader,
199                                   WindowController
    windowController) {
200         this.avatar = new Avatar(topBlockAtX0,
    inputListener, imageReader);
201         gameObjects().addGameObject(avatar, Layer.
    FOREGROUND);
202         setCamera(new Camera(avatar, Vector2.ZERO,
203                 windowController.getWindowDimensions
    (), windowController.getWindowDimensions()));
204     }
205 
206     private void initializeMoon(WindowController
    windowController) {
207         GameObject moon = Moon.create(
    windowController.getWindowDimensions(),
208                 CYCLE_OF_DAY_LENGTH);
209         gameObjects().addGameObject(moon, SUN_LAYER
    );
210     }
211 
212     private void initializeSun(WindowController
    windowController) {
213         GameObject sun = Sun.create(windowController
    .getWindowDimensions(),
214                 CYCLE_OF_DAY_LENGTH);
215         gameObjects().addGameObject(sun, SUN_LAYER);
216         gameObjects().addGameObject(SunHalo.create(
    sun), SUN_LAYER);
217 
218     }
219 
220     private void initializeNight(WindowController
```

```
220 windowController) {
221         gameObjects().addGameObject(Night.create(
    windowController.getWindowDimensions(),
222                 CYCLE_OF_DAY_LENGTH), Layer.UI);
223     }
224 
225     private void initializeTerrain(WindowController
    windowController) {
226         for (Block block : terrain.createInRange(0
    , (int) windowController.getWindowDimensions().x
    ())) { //todo change range
227             if (block.getTag().equals(TOP_LAYER_TAG
    ) ) {
228                 gameObjects().addGameObject(block,
    Layer.STATIC_OBJECTS);
229                 continue;
230             }
231             gameObjects().addGameObject(block, Layer
    .BACKGROUND);
232         }
233     }
234 
235     /**
236      * The main method to run the Pepse game.
237      *
238      * @param args Command-line arguments (not used
    ).
239      */
240     public static void main(String[] args) {
241         new PepseGameManager().run();
242     }
243 
244     private void initializeSky(WindowController
    windowController) {
245         gameObjects().addGameObject(Sky.create(
    windowController.getWindowDimensions()), SKY_LAYER);
246     }
247 
248     private int worldXToZone(float worldX) {
249         return Math.floorDiv((int)Math.floor(worldX
    ), (int)zoneWidth);
```

```
250     }
251
252     private int zoneStartX(int zoneIdx) {
253         return (int) (zoneIdx * zoneWidth);
254     }
255
256     private int zoneEndX(int zoneIdx) {
257         return (int) ((zoneIdx + 1) * zoneWidth);
258     }
259
260     /** A helper class to track a GameObject along
    with its layer.
261      * Used for managing loaded zones in the game.
262      * @author Eilam Soroka, Maayan Felig
263      */
264     private static class ObjInLayer {
265         final GameObject obj;
266         final int layer;
267         ObjInLayer(GameObject obj, int layer) {
268             this.obj = obj;
269             this.layer = layer;
270         }
271     }
272 }
273
```

```java
package pepse.utils;

import java.awt.*;
import java.util.Random;

/**
 * Provides procedurally-generated colors around a pivot.
 * @author Dan Nirel
 */
public final class ColorSupplier {
    private static final int DEFAULT_COLOR_DELTA = 10;
    private final static Random random = new Random();

    /**
     * Returns a color similar to baseColor, with a default delta.
     *
     * @param baseColor A color that we wish to approximate.
     * @return A color similar to baseColor.
     */
    public static Color approximateColor(Color baseColor) {
        return approximateColor(baseColor, DEFAULT_COLOR_DELTA);
    }


    /**
     * Returns a color similar to baseColor, with a difference of at most colorDelta.
     * Where the difference is equal along all channels
     *
     * @param baseColor A color that we wish to approximate.
     * @param colorDelta The maximal difference (per channel) between the sampled color and the base color
```

```
30 .
31      * @return A color similar to baseColor.
32      */
33     public static Color approximateMonoColor(Color
   baseColor, int colorDelta){
34         int channel = randomChannelInRange(baseColor.
   getRed()-colorDelta, baseColor.getRed()+colorDelta);
35         return new Color(channel, channel, channel);
36     }
37
38
39
40     /**
41      * Returns a color similar to baseColor, with a
   default delta.
42      * Where the difference is equal along all
   channels
43      *
44      * @param baseColor A color that we wish to
   approximate.
45      * @return A color similar to baseColor.
46      */
47     public static Color approximateMonoColor(Color
   baseColor) {
48         return approximateMonoColor(baseColor,
   DEFAULT_COLOR_DELTA);
49     }
50
51
52     /**
53      * Returns a color similar to baseColor, with a
   difference of at most colorDelta.
54      *
55      * @param baseColor A color that we wish to
   approximate.
56      * @param colorDelta The maximal difference (per
   channel) between the sampled color and the base color
   .
57      * @return A color similar to baseColor.
58      */
59     public static Color approximateColor(Color
```

```
59 baseColor, int colorDelta) {
60
61         return new Color(
62                 randomChannelInRange(baseColor.getRed
   ()-colorDelta, baseColor.getRed()+colorDelta),
63                 randomChannelInRange(baseColor.
   getGreen()-colorDelta, baseColor.getGreen()+
   colorDelta),
64                 randomChannelInRange(baseColor.
   getBlue()-colorDelta, baseColor.getBlue()+colorDelta
   ));
65     }
66
67     /**
68      * This method generates a random value for a
   color channel within the given range [min, max].
69      *
70      * @param min The lower bound of the given range.
71      * @param max The upper bound of the given range.
72      * @return A random number in the range [min, max
   ], clipped to [0,255].
73      */
74     private static int randomChannelInRange(int min,
   int max) {
75         int channel = random.nextInt(max-min+1) + min
   ;
76         return Math.min(255, Math.max(channel, 0));
77     }
78 }
79
```

```java
package pepse.utils;

import java.util.Random;

public class NoiseGenerator {
    private double seed;
    private long default_size;
    private int[] p;
    private int[] permutation;
    private double startPoint;

    /**
     * The constructor of the NoiseGenerator class.
     *
     * @param seed can be anything you want (even 1234 or new Random().nextGaussian()).
     *             This seed is the basis of the random generator, which
     *             will draw upon it to generate pseudo-random noise.
     *
     * @param startPoint is a relative point that the noise will be generated from.
     *                   In our case it should be your ground height at X0 (specified in
     *                   ex4 when we talk about the terrain: 2.2.1).
     *
     */
    public NoiseGenerator(double seed, int startPoint) {
        this.seed = seed;
        this.startPoint = startPoint;
        init();
    }

    private void init() {
        // Initialize the permutation array.
        this.p = new int[512];
        this.permutation = new int[]{151, 160, 137, 91, 90, 15, 131, 13, 201,
```

```
34                  95, 96, 53, 194, 233, 7, 225, 140, 36
   , 103, 30, 69, 142, 8, 99,
35                  37, 240, 21, 10, 23, 190, 6, 148, 247
   , 120, 234, 75, 0, 26,
36                  197, 62, 94, 252, 219, 203, 117, 35,
   11, 32, 57, 177, 33, 88,
37                  237, 149, 56, 87, 174, 20, 125, 136,
   171, 168, 68, 175, 74,
38                  165, 71, 134, 139, 48, 27, 166, 77,
   146, 158, 231, 83, 111,
39                  229, 122, 60, 211, 133, 230, 220, 105
   , 92, 41, 55, 46, 245, 40,
40                  244, 102, 143, 54, 65, 25, 63, 161, 1
   , 216, 80, 73, 209, 76,
41                  132, 187, 208, 89, 18, 169, 200, 196
   , 135, 130, 116, 188, 159,
42                  86, 164, 100, 109, 198, 173, 186, 3,
   64, 52, 217, 226, 250,
43                  124, 123, 5, 202, 38, 147, 118, 126,
   255, 82, 85, 212, 207,
44                  206, 59, 227, 47, 16, 58, 17, 182,
   189, 28, 42, 223, 183, 170,
45                  213, 119, 248, 152, 2, 44, 154, 163,
   70, 221, 153, 101, 155,
46                  167, 43, 172, 9, 129, 22, 39, 253, 19
   , 98, 108, 110, 79, 113,
47                  224, 232, 178, 185, 112, 104, 218,
   246, 97, 228, 251, 34, 242,
48                  193, 238, 210, 144, 12, 191, 179, 162
   , 241, 81, 51, 145, 235,
49                  249, 14, 239, 107, 49, 192, 214, 31,
   181, 199, 106, 157, 184,
50                  84, 204, 176, 115, 121, 50, 45, 127,
   4, 150, 254, 138, 236,
51                  205, 93, 222, 114, 67, 29, 24, 72,
   243, 141, 128, 195, 78, 66,
52                  215, 61, 156, 180};
53          this.default_size = 35;
54
55          // Populate it
56          for (int i = 0; i < 256; i++) {
```

```
57              p[256 + i] = p[i] = permutation[i];
58          }
59
60      }
61
62      /**
63       * Noise is responsible to generate pseudo random noise according to the seed given upon constructing the object.
64       *
65       * @param x the wanted x to receive noise for (in our case, the x coordinate of the terrain you'd want to create).
66       * @param factor describes how large the noise should be (play with it, but BLOCK_SIZE *7 should be enough).
67       * @return returns a noise you should *add* to the groundHeightAtX0 you have.
68       *
69       * example:
70       * public float groundHeightAt(float x) {
71       *           float noise = (float) noiseGenerator.noise(x, BLOCK_SIZE *7);
72       *           return groundHeightAtX0 + noise;
73       *       }
74       *
75       */
76      public double noise(double x, double factor) {
77          double value = 0.0;
78          double currentPoint = startPoint;
79
80          while (currentPoint >= 1) {
81              value += smoothNoise((x / currentPoint), 0, 0) * currentPoint;
82              currentPoint /= 2.0;
83          }
84
85          return value * factor / startPoint;
86      }
87
88
```

```
 89     private double smoothNoise(double x, double y,
    double z) {
 90         // Offset each coordinate by the seed value
 91         x += this.seed;
 92         y += this.seed;
 93         x += this.seed;
 94 
 95         int X = (int) Math.floor(x) & 255; // FIND
    UNIT CUBE THAT
 96         int Y = (int) Math.floor(y) & 255; //
    CONTAINS POINT.
 97         int Z = (int) Math.floor(z) & 255;
 98 
 99         x -= Math.floor(x); // FIND RELATIVE X,Y,Z
100         y -= Math.floor(y); // OF POINT IN CUBE.
101         z -= Math.floor(z);
102 
103         double u = fade(x); // COMPUTE FADE CURVES
104         double v = fade(y); // FOR EACH OF X,Y,Z.
105         double w = fade(z);
106 
107         int A = p[X] + Y;
108         int AA = p[A] + Z;
109         int AB = p[A + 1] + Z; // HASH COORDINATES
    OF
110         int B = p[X + 1] + Y;
111         int BA = p[B] + Z;
112         int BB = p[B + 1] + Z; // THE 8 CUBE CORNERS
    ,
113 
114         return lerp(w, lerp(v, lerp(u, grad(p[AA], x
    , y, z),    // AND ADD
115                                 grad(p[BA], x - 1, y
    , z)), // BLENDED
116                         lerp(u, grad(p[AB], x, y - 1
    , z),    // RESULTS
117                                 grad(p[BB], x - 1, y
     - 1, z))),// FROM 8
118                 lerp(v, lerp(u, grad(p[AA + 1], x, y
    , z - 1),    // CORNERS
119                                 grad(p[BA + 1], x -
```

```
119 1, y, z - 1)), // OF CUBE
120                         lerp(u, grad(p[AB + 1], x, y
     - 1, z - 1),
121                                 grad(p[BB + 1], x -
    1, y - 1, z - 1))));
122     }
123
124     private double fade(double t) {
125         return t * t * t * (t * (t * 6 - 15) + 10);
126     }
127
128     private double lerp(double t, double a, double b
    ) {
129         return a + t * (b - a);
130     }
131
132     private double grad(int hash, double x, double y
    , double z) {
133         int h = hash & 15; // CONVERT LO 4 BITS OF
    HASH CODE
134         double u = h < 8 ? x : y, // INTO 12
    GRADIENT DIRECTIONS.
135                 v = h < 4 ? y : h == 12 || h == 14
     ? x : z;
136         return ((h & 1) == 0 ? u : -u) + ((h & 2
    ) == 0 ? v : -v);
137     }
138 }
139
```

```java
package pepse.world;

import danogl.GameObject;
import danogl.components.CoordinateSpace;
import danogl.gui.rendering.RectangleRenderable;
import danogl.util.Vector2;
import pepse.Constants;

import java.awt.*;

/** * A class responsible for creating the sky background.
 * @author Eilam Soroka, Maayan Felig
 */
public class Sky {

    /**
     * Creates a sky GameObject.
     * @param windowDimensions The dimensions of the window.
     * @return The sky GameObject.
     */
    public static GameObject create(Vector2 windowDimensions){
        GameObject sky =new GameObject(
                Vector2.ZERO,
                windowDimensions,
                new RectangleRenderable(Constants.BASIC_SKY_COLOR)
        );
        sky.setCoordinateSpace(CoordinateSpace.CAMERA_COORDINATES);
        sky.setTag(Constants.SKY_TAG);
        return sky;
    }
}

```

```java
package pepse.world;

import danogl.GameObject;
import danogl.collisions.Collision;
import danogl.components.GameObjectPhysics;
import danogl.gui.rendering.Renderable;
import danogl.util.Vector2;
import pepse.Constants;

import static pepse.Constants.BLOCK_SIZE;
/**
 * A block GameObject.
 * Blocks are basic GameObjects that other GameObjects can be constructed from.
 * @author Eilam Soroka, Maayan Felig
 */
public class Block extends GameObject {


    public Block(Vector2 topLeftCorner, Renderable renderable) {
        super(topLeftCorner, Vector2.ONES.mult(BLOCK_SIZE), renderable);
        physics().preventIntersectionsFromDirection(Vector2.ZERO);
        physics().setMass(GameObjectPhysics.IMMOVABLE_MASS);
    }

    public static int getSize() {
        return BLOCK_SIZE;
    }

}

```

```java
package pepse.world;

import danogl.util.Vector2;
import pepse.Constants;
import pepse.utils.NoiseGenerator;
import danogl.gui.rendering.RectangleRenderable;
import pepse.utils.ColorSupplier;
import java.awt.*;
import java.util.ArrayList;
import java.util.List;

import static pepse.Constants.*;

/**
 * Represents the terrain of the game world.
 * The terrain's height is determined using a noise generator to create a natural, uneven surface.
 * @author Eilam Soroka, Maayan Felig
 */
public class Terrain {
    private final Vector2 windowDimensions;
    private final int seed;
    private final NoiseGenerator noiseGenerator;

    /**
     * Constructs a Terrain object.
     * @param windowDimensions The dimensions of the game window.
     * @param seed The seed for the noise generator to ensure consistent terrain generation.
     */
    public Terrain(Vector2 windowDimensions, int seed){
        this.windowDimensions = windowDimensions;
        this.seed = seed;
        //Random rand = new Random(Objects.hash(START_x0, seed));
        //this.groundHeightAtX0 = rand.nextInt((int)(windowDimensions.y()*(2/3.0)), (int)(windowDimensions.y()*(5/6.0)));
//        this.groundHeightAtX0 = (2/3.0f) *
```

```
34 windowDimensions.y();
35         this.noiseGenerator = new NoiseGenerator(seed
   , (int)(GROUND_HEIGHT_FACTOR * windowDimensions.y
   ()));
36     }
37     public float groundHeightAt(float x){
38         float noise = (float) noiseGenerator.noise(x,
   NOISE_FACTOR_TERRAIN);
39         return (GROUND_HEIGHT_FACTOR *
   windowDimensions.y() )+ noise;
40     }
41
42     /**
43      * Creates terrain blocks within the specified x-
   coordinate range.
44      * @param minX The minimum x-coordinate of the
   range.
45      * @param maxX The maximum x-coordinate of the
   range.
46      * @return A list of terrain blocks within the
   specified range.
47      */
48     public List<Block> createInRange(int minX, int
   maxX) {
49
50         ArrayList<Block> blocks = new ArrayList<>();
51         for (int x = minX; x <= maxX; x += BLOCK_SIZE
   ) {
52             float groundHeight = groundHeightAt(x);
53             Block topBlock = new Block(new Vector2(x
   , (int) groundHeight),
54                     new RectangleRenderable(
   ColorSupplier.approximateColor(BASE_GROUND_COLOR)));
55             topBlock.setTag(
   TOP_LAYER_GROUND_BLOCK_TAG);
56             blocks.add(topBlock);
57             for (int y = (int) (groundHeight +
   BLOCK_SIZE);
58                  y < (int) groundHeight + (
   TERRAIN_DEPTH * BLOCK_SIZE); y += BLOCK_SIZE) {
59                 Block newBlock = new Block(new
```

```
59 Vector2(x, y),
60                         new RectangleRenderable(
   ColorSupplier.approximateColor(BASE_GROUND_COLOR)));
61                 newBlock.setTag(
   LOWER_LAYER_GROUND_BLOCK_TAG);
62                 blocks.add(newBlock);
63             }
64         }
65         return blocks;
66     }
67 
68     /**
69      * Static method to get the ground height at x=0
   based on window dimensions.
70      * @param windowDimensions The dimensions of the
   game window.
71      * @return The ground height at x=0.
72      */
73     public static float groundHeightAtX0(Vector2
   windowDimensions) {
74         return GROUND_HEIGHT_FACTOR *
   windowDimensions.y();
75     }
76 
77     /** Getter for GROUND_HEIGHT_FACTOR
78      * @return GROUND_HEIGHT_FACTOR
79      * */
80     public static float getGroundHeightFactor() {
81         return GROUND_HEIGHT_FACTOR;
82     }
83     /**
84      * Getter for TERRAIN_DEPTH
85      * @return TERRAIN_DEPTH
86      * */
87     public static float getDepth() {
88         return TERRAIN_DEPTH;
89     }
90 
91 }
92 
```

```java
package pepse.world;

import danogl.GameObject;
import danogl.collisions.Layer;
import danogl.components.CoordinateSpace;
import danogl.gui.rendering.TextRenderable;
import danogl.util.Vector2;

import java.util.function.BiConsumer;
import java.util.function.Supplier;

/**
 * The energy display of the game.
 * Displays the player's remaining energy as text on the screen.
 * @author Eilam Soroka, Maayan Felig
 */
public class EnergyDisplay {

    private static final Vector2 ENERGY_DISPLAY_POSITION = new Vector2(20, 20);
    private static final Vector2 ENERGY_DISPLAY_SIZE = new Vector2(150, 50);
    private static final String ENERGY_TEXT_SUFFIX = "%";

    /**
     * Creates an energy text display GameObject.
     *
     * @param energySupplier A supplier of the player's current energy.
     * @param addGameObject  A consumer that adds a GameObject to the game.
     */
    public EnergyDisplay(Supplier<Integer> energySupplier,
                         BiConsumer<GameObject, Integer> addGameObject) {

        int initialEnergy = energySupplier.get();
        TextRenderable textRenderable =
```

```
34                  new TextRenderable(initialEnergy +
   ENERGY_TEXT_SUFFIX);
35          GameObject energyText = new GameObject(
36                  ENERGY_DISPLAY_POSITION,
37                  ENERGY_DISPLAY_SIZE,
38                  textRenderable
39          );
40          energyText.setCoordinateSpace(CoordinateSpace
   .CAMERA_COORDINATES);
41          final int[] lastEnergy = {initialEnergy};
42          energyText.addComponent(deltaTime -> {
43              int currentEnergy = energySupplier.get();
44              if (currentEnergy != lastEnergy[0]) {
45                  textRenderable.setString(
   currentEnergy + ENERGY_TEXT_SUFFIX);
46                  lastEnergy[0] = currentEnergy;
47              }
48          });
49          addGameObject.accept(energyText, Layer.UI);
50      }
51  }
52
```

```java
package pepse.world.trees;

import danogl.components.ScheduledTask;
import danogl.components.Transition;
import danogl.gui.rendering.RectangleRenderable;
import danogl.util.Vector2;
import pepse.Constants;
import pepse.utils.ColorSupplier;
import pepse.world.Block;

import java.awt.*;

/**
 * A class that represents a leaf block in the game world.
 * Extends the Block class to inherit properties and behaviors of a block.
 * @author Eilam Soroka and Maayan Felig
 */
public class Leaf extends Block {
    private static final Color BASE_LEAF_COLOR = new Color(50, 200, 30);;
    private static final float LEAF_CHANGE_TIME = 2.0f;
    private static final float LEAF_ANGLE_DIFF = 20.0f;
    private static final float LEAF_SIZE_FACTOR = 2.0f;

    /**
     * Constructor for a Leaf object that moves with the wind.
     * @param TopLeftCorner top left corner where the leaf should appear on screen
     */
    public Leaf(Vector2 TopLeftCorner) {
        super(TopLeftCorner, new RectangleRenderable(ColorSupplier.approximateColor(BASE_LEAF_COLOR)));
        this.setTag(Constants.LEAF_TAG);
        physics().preventIntersectionsFromDirection(Vector2.ZERO);
```

```
32      }
33
34      /**
35       * schedules the movement of a leaf in both its
   angle and dimensions
36       * @param time the amount of time to wait before
   starting movement
37       */
38      public void createWindEffect(float time) {
39          new ScheduledTask(this, time,true, () -> {
40              changeLeafAngle();
41              changeLeafDimensions();
42          });
43      }
44
45      private void changeLeafAngle() {
46          float startAngle = this.renderer().
   getRenderableAngle();
47          float endAngle = startAngle + LEAF_ANGLE_DIFF
   ;
48          new Transition<>(this,
49                  (Float angle) -> this.renderer().
   setRenderableAngle(angle),
50                  startAngle,
51                  endAngle,
52                  Transition.LINEAR_INTERPOLATOR_FLOAT,
53                  LEAF_CHANGE_TIME,
54                  Transition.TransitionType.
   TRANSITION_BACK_AND_FORTH,
55                  null);
56      }
57
58      private void changeLeafDimensions() {
59          Vector2 startSize = new Vector2(Block.getSize
   (), Block.getSize());
60          Vector2 endSize = startSize.subtract(
61                  new Vector2((float) Block.getSize()/
   LEAF_SIZE_FACTOR,
62                  0));
63          new Transition<>(this,
64                  this::setDimensions,
```

```
65                  startSize,
66                  endSize,
67                  Transition.
   LINEAR_INTERPOLATOR_VECTOR,
68                  LEAF_CHANGE_TIME,
69                  Transition.TransitionType.
   TRANSITION_BACK_AND_FORTH, null
70                  );
71      }
72
73  }
74
```

```java
package pepse.world.trees;

import danogl.gui.rendering.RectangleRenderable;
import danogl.gui.rendering.Renderable;
import danogl.util.Vector2;
import pepse.Constants;
import pepse.utils.ColorSupplier;
import pepse.world.Block;

import java.awt.*;
import java.util.ArrayList;
import java.util.List;
import java.util.Objects;
import java.util.Random;

public class Tree {

    private static final int MIN_TREE_HEIGHT = 4;
    private static final int MAX_TREE_HEIGHT_ADDITION
 = 4;
    private static final int TREE_WIDTH = 1;
    public static final Color BASE_TRUNK_COLOR = new
Color(139, 69, 19);
    private static final Renderable TRUNK_RENDERABLE
 =
            new RectangleRenderable(ColorSupplier.
approximateColor(BASE_TRUNK_COLOR));
    private static final int TREE_TOP_RADIUS = 3;
    private static final double LEAF_CHANCE = 0.4;
    private static final double FRUIT_CHANCE = 0.08;
    private final Random random;
    private final int trunkHeight;
    private final List<Block> TrunkBlocks = new
ArrayList<>();
    private final List<Leaf> allLeaves = new
ArrayList<>();
    private final List<Fruit> allFruit = new
ArrayList<>();


    /**
```

```
35      * Constructor for a Tree object that creates the
    trunk, leaves, and fruit.
36      * @param bottomLeftCorner the bottom left corner
    where the tree trunk should start
37      * @param seed a seed for random number
   generation to ensure reproducibility
38      */
39     public Tree(Vector2 bottomLeftCorner, int seed) {
40         this.random = new Random(Objects.hash(
   bottomLeftCorner.x(), seed));
41         this.trunkHeight = MIN_TREE_HEIGHT + (random.
   nextInt(MAX_TREE_HEIGHT_ADDITION));
42         createTreeTrunk(bottomLeftCorner);
43         createTreeTop();
44         createWindEffectForLeaves();
45     }
46 
47     private void createWindEffectForLeaves() {
48         for (Leaf leaf : allLeaves) {
49             float timeToStart = random.nextFloat() *
   Constants.WIND_EFFECT_DIFFERENCES;
50             leaf.createWindEffect(timeToStart);
51         }
52     }
53 
54     private void createTreeTrunk(Vector2
   bottomLeftCorner) {
55         for (int i = 0; i < this.trunkHeight; i++) {
   //check whether i should start at 0 or 1
56             for (int j = 0; j < TREE_WIDTH; j++) {
57                 Vector2 blockTopLeftCorner = new
   Vector2(bottomLeftCorner.x() + j * Block.getSize(),
58                         bottomLeftCorner.y() - (i + 1
   ) * Block.getSize());
59                 Block trunkBlock = new Block(
   blockTopLeftCorner, TRUNK_RENDERABLE);
60                 trunkBlock.setTag(Constants.
   TRUNK_BLOCK_TAG);
61                 TrunkBlocks.add(trunkBlock);
62             }
63         }
```

```
64      }
65
66      /**
67       * getter for the trunk blocks of the tree
68       * @return list of trunk blocks
69       */
70      public List<Block> getTrunkBlocks() {
71          return TrunkBlocks;
72      }
73
74      private void createTreeTop(){
75          Vector2 treeTopCenter = new Vector2(
76                  TrunkBlocks.getFirst().
    getTopLeftCorner().x() + Block.getSize(),
77                  TrunkBlocks.getLast().
    getTopLeftCorner().y() - (float) this.trunkHeight *
    Block.getSize()/2
78          );
79          for (int i = -TREE_TOP_RADIUS; i <=
    TREE_TOP_RADIUS; i++) {
80              for (int j = -TREE_TOP_RADIUS; j <=
    TREE_TOP_RADIUS; j++) {
81                  double distance = Math.sqrt(i * i +
    j * j);
82                  if (distance > TREE_TOP_RADIUS) {
83                      continue;
84                  }
85                  Vector2 particleTopLeftCorner =
    treeTopCenter.add(
86                          new Vector2(i * Block.
    getSize(), j * Block.getSize())//todo check whether
    it's ok
87                          // to have Block mentioned
    here or whether we should have a constant instead
88                  );
89                  double curRandom = random.nextDouble
    ();
90
91                  if (curRandom < LEAF_CHANCE) {
92                      Leaf leaf = new Leaf(
    particleTopLeftCorner);
```

```
 93                      leaf.setTag(Constants.LEAF_TAG);
 94                      allLeaves.add(leaf);
 95                  }
 96                  else if (curRandom < LEAF_CHANCE +
    FRUIT_CHANCE) {
 97                      Fruit fruit = new Fruit(
    particleTopLeftCorner);
 98                      fruit.setTag(Constants.FRUIT_TAG
    );
 99                      allFruit.add(fruit);
100                  }
101              }
102          }
103
104      }
105
106      /**
107       * getter for all leaves on the tree
108       * @return list of all leaves on the tree
109       */
110      public List<Leaf> getAllLeaves() {
111          return allLeaves;
112      }
113
114      /**
115       * getter for all fruit on the tree
116       * @return list of all fruit on the tree
117       */
118      public List<Fruit> getAllFruit() {
119          return allFruit;
120      }
121  }
122
```

```java
package pepse.world.trees;

import danogl.util.Vector2;

import java.util.ArrayList;
import java.util.List;
import java.util.Objects;
import java.util.Random;
import java.util.function.UnaryOperator;

public class Flora {

    private static final int MIN_SPACE_BETWEEN_TREES = 150;
    private static final int FIRST_POSSIBLE_TREE_X = 10;
    private static final double TREE_CHANCE = 0.8;
    private final int seed;
    private final UnaryOperator<Float> groundHeightAt;

    public Flora(int seed, UnaryOperator<Float> groundHeightAt) {
        this.seed = seed;
        this.groundHeightAt = groundHeightAt;
    }

    public List<Tree> createInRange(int minX, int maxX) {
        ArrayList<Tree> trees = new ArrayList<>();
        for (int i = minX; i <= maxX; i += MIN_SPACE_BETWEEN_TREES) {
            if(i == 0)
                continue;
            Random random = new Random(Objects.hash(seed, i));
            if (random.nextDouble() < TREE_CHANCE) {
                Tree tree = new Tree(new Vector2((float) i, this.groundHeightAt.apply((float) i)), this.seed);
                trees.add(tree);
```

```
33              }
34          }
35          return trees;
36      }
37 }
38
```

```java
package pepse.world.trees;

import danogl.GameObject;
import danogl.collisions.Collision;
import danogl.components.ScheduledTask;
import danogl.gui.rendering.OvalRenderable;
import danogl.gui.rendering.Renderable;
import danogl.util.Vector2;
import pepse.Constants;
import pepse.utils.ColorSupplier;
import pepse.world.Block;
import pepse.world.avatar.Avatar;

import java.awt.*;

public class Fruit extends Block {

    private static final Renderable FRUIT_RENDERABLE
     =
            new OvalRenderable(ColorSupplier.
    approximateColor(Color.ORANGE));
    private boolean isCollected = false;

    public Fruit(Vector2 TopLeftCorner)
    {
        super(TopLeftCorner, FRUIT_RENDERABLE);
        this.setTag(Constants.FRUIT_TAG);
    }

    @Override
    public boolean shouldCollideWith(GameObject other
    ) {
        return other.getTag().equals(Constants.
    AVATAR_TAG);
    }

    @Override
    public void onCollisionEnter(GameObject other,
    Collision collision) {
        if (other.getTag().equals(Constants.
    AVATAR_TAG) && !isCollected) {
```

```
36              isCollected = true;
37              renderer().setRenderable(null);
38              this.setDimensions(Vector2.ZERO);
39
40              new ScheduledTask(this,
41                      Constants.CYCLE_LENGTH,
42                      false,
43                      this::reappear);
44              //todo add 10 points of energy to avatar
45              ((Avatar) other).addEnergy(Constants.
   FRUIT_ENERGY);
46          }
47      }
48
49      private void reappear() {
50          this.isCollected = false;
51          renderer().setRenderable(FRUIT_RENDERABLE);
52          this.setDimensions(Vector2.ONES.mult(Block.
   getSize()));
53      }
54  }
55
```

```java
package pepse.world.avatar;

import danogl.GameObject;
import danogl.collisions.Collision;
import danogl.gui.ImageReader;
import danogl.gui.UserInputListener;
import danogl.util.Vector2;
import danogl.gui.rendering.AnimationRenderable;
import pepse.Constants;

import java.awt.event.KeyEvent;

import static pepse.Constants.*;

/**
 * Represents the Avatar character in the game. The avatar can move left, right, and jump,
 * while consuming energy. The avatar's energy can also be replenished by picking up fruits.
 * @author Eilam Soroka, Maayan Felig
 */
public class Avatar extends GameObject {

    private static final Vector2 AVATAR_SIZE = new Vector2(50, 50);
    private static final float OFFSET_GROUND_START = 20f;
    private static final float GRAVITY = 1000f;
    private static final float WALK_SPEED = 300f;
    private static final float JUMP_SPEED = 600f;
    private static final int LEFT_RIGHT_MOVEMENT_COST = 2;
    private static final int JUMP_COST = 20;
    private static final int DOUBLE_JUMP_COST = 50;
    private static final float ENERGY_RETURN_RATE = 1f; //todo write on readme that i change movement cost
    private static final int ENERGY_RETURN_AMOUNT = 1;
    private static final int FRUIT_ENERGY_VALUE = 10;
    private static final String[] IDLE_IMAGES = new
```

```
33 String[]
34     {"assets/idle_0.png","assets/idle_1.png","assets/
   idle_2.png","assets/idle_3.png"};
35     private static final String[] WALKING_IMAGES =
   new String[]
36             {"assets/run_0.png","assets/run_1.png","
   assets/run_2.png","assets/run_3.png"};
37     private static final String[] JUMPING_IMAGES =
   new String[]
38             {"assets/jump_0.png","assets/jump_1.png",
   "assets/jump_2.png","assets/jump_3.png"};
39     private static final double
   TIME_BETWEEN_ANIMATION_FRAMES = 0.2f;
40
41
42     private static enum state {
43         IDLE,
44         WALKING,
45         JUMPING
46     }
47
48     private boolean onGround;
49     private boolean doubleJumpUsed;
50     private float energyReturn = 0f;
51     private final Vector2 topleftCorner;
52     private final UserInputListener inputListener;
53     private final ImageReader imageReader;
54     private state currentState;
55     private AnimationRenderable idleAnimation;
56     private AnimationRenderable walkAnimation;
57     private AnimationRenderable jumpAnimation;
58
59     private int energyMeter;
60     private boolean spaceWasPressed = false;
61
62
63     /**
64      * Constructs an Avatar object with the specified
    starting position, input listener,
65      * and image reader.
66      *
```

```
67      * @param topLeftCorner The top-left corner
   position of the avatar.
68      * @param inputListener The input listener used
   to handle user inputs.
69      * @param imageReader   The image reader used to
    read image files for the avatar's animations.
70      */
71     public Avatar(Vector2 topLeftCorner,
72                   UserInputListener inputListener,
73                   ImageReader imageReader){
74         super(new Vector2(topLeftCorner.x(),
75                 topLeftCorner.y() - AVATAR_SIZE.y
   () - OFFSET_GROUND_START) , AVATAR_SIZE, imageReader
   .readImage("assets/idle_0.png", true));
76         this.topleftCorner = topLeftCorner;
77         this.inputListener = inputListener;
78         this.imageReader = imageReader;
79         physics().preventIntersectionsFromDirection(
   Vector2.ZERO);
80         transform().setAccelerationY(GRAVITY);
81         this.currentState = state.IDLE;
82         this.energyMeter = MAX_ENERGY;
83         this.doubleJumpUsed = false;
84
85         this.idleAnimation = new AnimationRenderable
   (IDLE_IMAGES
86                 , imageReader, true,
   TIME_BETWEEN_ANIMATION_FRAMES);
87         this.walkAnimation = new AnimationRenderable
   (WALKING_IMAGES
88                 , imageReader, true,
   TIME_BETWEEN_ANIMATION_FRAMES);
89         this.jumpAnimation = new AnimationRenderable
   (JUMPING_IMAGES
90                 , imageReader, true,
   TIME_BETWEEN_ANIMATION_FRAMES);
91
92         renderer().setRenderable(idleAnimation);
93
94         setTag(Constants.AVATAR_TAG);
95     }
```

```
 96
 97     /**
 98      * Updates the avatar state based on the delta
    time, including movement and energy consumption.
 99      *
100      * @param deltaTime The time delta since the
    last update.
101      */
102     @Override
103     public void update(float deltaTime) {
104         super.update(deltaTime);
105         handleMovement();
106         updateState();
107     }
108
109     private void handleMovement() {
110         boolean did_I_moved =
    left_right_movement_handler();
111         jump_movement_handler();
112         if (onGround && !did_I_moved && energyMeter
     < MAX_ENERGY) {
113             energyReturn += ENERGY_RETURN_RATE;
114             if (energyReturn >= 1f) {
115                 energyReturn = 0f;
116                 energyMeter += ENERGY_RETURN_AMOUNT;
117             }
118         }
119     }
120
121     private void updateState() {
122         if (!onGround) {
123             setState(state.JUMPING);
124         }
125         else if (transform().getVelocity().x() != 0
    ) {
126             setState(state.WALKING);
127         }
128         else {
129             setState(state.IDLE);
130         }
131     }
```

```
132
133     private void setState(state newState) {
134         if (newState == currentState) {
135             return;
136         }
137
138         currentState = newState;
139
140         switch (currentState) {
141             case IDLE:
142                 renderer().setRenderable(
    idleAnimation);
143                 break;
144             case WALKING:
145                 renderer().setRenderable(
    walkAnimation);
146                 break;
147             case JUMPING:
148                 renderer().setRenderable(
    jumpAnimation);
149                 break;
150         }
151     }
152
153     private void jump_movement_handler() {
154         boolean spacePressed = inputListener.
    isKeyPressed(KeyEvent.VK_SPACE);
155
156         if (spacePressed && !spaceWasPressed) {
157
158             if (onGround && energyMeter >= JUMP_COST
    ) {
159                 energyMeter -= JUMP_COST;
160                 transform().setVelocityY(-JUMP_SPEED
    );
161                 onGround = false;
162             }
163
164             else if (!onGround && !doubleJumpUsed
     && energyMeter >= DOUBLE_JUMP_COST) {
165                 energyMeter -= DOUBLE_JUMP_COST;
```

```
166                 transform().setVelocityY(-JUMP_SPEED
    );
167                 doubleJumpUsed = true;
168             }
169         }
170 
171         spaceWasPressed = spacePressed;
172     }
173 
174     private boolean left_right_movement_handler() {
175         float velocityX = 0;
176 
177         if (inputListener.isKeyPressed(KeyEvent.
    VK_LEFT)) {
178             velocityX -= WALK_SPEED;
179         }
180         if (inputListener.isKeyPressed(KeyEvent.
    VK_RIGHT)) {
181             velocityX += WALK_SPEED;
182         }
183 
184         boolean wantsToMove = (velocityX != 0);
185 
186         if (wantsToMove && onGround) {
187             if (energyMeter >=
    LEFT_RIGHT_MOVEMENT_COST) {
188                 energyMeter -=
    LEFT_RIGHT_MOVEMENT_COST;
189             } else {
190                 velocityX = 0;
191                 wantsToMove = false;
192             }
193         }
194         if (velocityX > 0) {
195             renderer().setIsFlippedHorizontally(
    false);
196         }
197         else if (velocityX < 0) {
198             renderer().setIsFlippedHorizontally(true
    );
199         }
```

```
200         transform().setVelocityX(velocityX);
201         return wantsToMove;
202     }
203 
204 
205     /**
206      *  * Handles collision events when the avatar
    collides with another game object.
207      *
208      * @param other     The other game object
    involved in the collision.
209      * @param collision The collision information.
210      */
211     @Override
212     public void onCollisionEnter(GameObject other,
    Collision collision) {
213         super.onCollisionEnter(other, collision);
214         if (collision.getNormal().y() < 0 &&
215                 other.getTag().equals(
    TOP_LAYER_GROUND_BLOCK_TAG)) {
216 
217             onGround = true;
218             doubleJumpUsed = false;
219             transform().setVelocityY(0);
220         }
221 
222     }
223 
224     /**
225      * Adds energy to the avatar's energy meter, up
    to the maximum limit.
226      *
227      * @param energyToAdd The amount of energy to
    add.
228      */
229     public void addEnergy(int energyToAdd) {
230         this.energyMeter = Math.min(this.energyMeter
     + energyToAdd, MAX_ENERGY);
231     }
232 
233     /**
```

```
234      * Returns the current energy level of the
    avatar.
235      *
236      * @return The current energy level.
237      */
238     public int getEnergyMeter() {
239         return energyMeter;
240     }
241
242 }
243
```

```java
package pepse.world.daynight;

import danogl.GameObject;
import danogl.components.CoordinateSpace;
import danogl.components.Transition;
import danogl.gui.rendering.OvalRenderable;
import danogl.gui.rendering.RectangleRenderable;
import danogl.util.Vector2;
import pepse.world.Terrain;

import java.awt.*;
/**
 * Handles the creation and movement of the sun in the game world.
 * @author Eilam Soroka, Maayan Felig
 */
public class Sun {
    private static final int MID_SCREEN_FACTOR = 2;
    private static final Vector2 SUN_SIZE = new Vector2(64, 64);
    private static final String SUN_TAG = "sun";
    private static final float START_CYCLE = 90f;
    private static final float END_CYCLE = 450f;

    /**
     * Creates a sun GameObject that moves in a circular path to simulate day and night.
     *
     * @param windowDimensions The dimensions of the game window
     * @param cycleLength      The duration of a full sun cycle (day-night cycle)
     * @return A GameObject representing the sun
     */
    public static GameObject create(Vector2 windowDimensions, float cycleLength){
        Vector2 sunPosition = new Vector2( windowDimensions.x() / MID_SCREEN_FACTOR,
                (windowDimensions.y() * Terrain.getGroundHeightFactor()) / MID_SCREEN_FACTOR);
        GameObject sun = new GameObject(sunPosition,
```

```
33 SUN_SIZE,
34                 new OvalRenderable(Color.YELLOW));
35         sun.setCoordinateSpace(CoordinateSpace.
   CAMERA_COORDINATES);
36         sun.setTag(SUN_TAG);
37         Vector2 initialSunCenter = sun.getCenter();
38         new Transition<Float>(
39                 sun, // the game object being changed
40                 (Float angle) -> {
41                     Vector2 cycleCenter = new Vector2
   (
42                             windowDimensions.x() / 2f
   ,
43                             windowDimensions.y() *
   Terrain.getGroundHeightFactor()
44                     );
45
46                     float radius = Math.min(
47                             windowDimensions.x() / 2f
   ,
48                             windowDimensions.y() *
   Terrain.getGroundHeightFactor()
49                     ) - SUN_SIZE.y();
50
51                     Vector2 offset = new Vector2(
   radius, 0).rotated(180f - angle);
52
53                     sun.setCenter(cycleCenter.add(
   offset));
54                 },
55                 START_CYCLE,
56                 END_CYCLE,
57                 Transition.LINEAR_INTERPOLATOR_FLOAT,
58                 cycleLength,
59                 Transition.TransitionType.
   TRANSITION_LOOP,
60                 null
61         );
62
63         return sun;
64     }
```

```
65 }
66
```

```java
package pepse.world.daynight;

import danogl.GameObject;
import danogl.components.CoordinateSpace;
import danogl.components.Transition;
import danogl.gui.rendering.OvalRenderable;
import danogl.util.Vector2;
import pepse.world.Terrain;

import java.awt.*;
/**
 * Handles the creation and movement of the moon in the game world.
 * @author Eilam Soroka, Maayan Felig
 */
public class Moon {
    private static final int MID_SCREEN_FACTOR = 2;
    private static final Vector2 MOON_SIZE = new Vector2(64, 64);
    private static final String MOON_TAG = "moon";
    private static final float START_CYCLE = -90f;
    private static final float END_CYCLE = 270f;

    /**
     * Creates a moon GameObject that moves in a circular path to simulate the night cycle.
     *
     * @param windowDimensions The dimensions of the game window
     * @param cycleLength      The duration of a full moon cycle (day-night cycle)
     * @return A GameObject representing the moon
     */
    public static GameObject create(Vector2 windowDimensions, float cycleLength){
        Vector2 moonPosition = new Vector2(windowDimensions.x() / MID_SCREEN_FACTOR,
                (windowDimensions.y() * Terrain.getGroundHeightFactor()) / MID_SCREEN_FACTOR);
        GameObject moon = new GameObject(moonPosition, MOON_SIZE,
```

```
33                  new OvalRenderable(Color.GRAY));
34          moon.setCoordinateSpace(CoordinateSpace.
    CAMERA_COORDINATES);
35          moon.setTag(MOON_TAG);
36          Vector2 initialSunCenter = moon.getCenter();
37          new Transition<Float>(
38                  moon, // the game object being
    changed
39                  (Float angle) -> {
40                      Vector2 cycleCenter = new Vector2
    (
41                              windowDimensions.x() / 2f
    ,
42                              windowDimensions.y() *
    Terrain.getGroundHeightFactor()
43                      );
44
45                      float radius = Math.min(
46                              windowDimensions.x() / 2f
    ,
47                              windowDimensions.y() *
    Terrain.getGroundHeightFactor()
48                      ) - MOON_SIZE.y();
49
50                      Vector2 offset = new Vector2(
    radius, 0).rotated(180f - angle);
51
52                      moon.setCenter(cycleCenter.add(
    offset));
53                  },
54                  START_CYCLE,
55                  END_CYCLE,
56                  Transition.LINEAR_INTERPOLATOR_FLOAT,
57                  cycleLength,
58                  Transition.TransitionType.
    TRANSITION_LOOP,
59                  null
60          );
61
62          return moon;
63      }
```

```
64 }
65
66
```

```java
package pepse.world.daynight;

import danogl.GameObject;
import danogl.components.Transition;
import danogl.gui.rendering.RectangleRenderable;
import danogl.util.Vector2;

import java.awt.*;
/**
 * Simulates the night effect by gradually increasing and decreasing a dark overlay's opacity.
 * @author Eilam Soroka, Maayan Felig
 */
public class Night {

    private static final float NOON_OPACITY = 0.0f;
    private static final float MIDNIGHT_OPACITY = 0.5f;
    private static final String NIGHT_TAG = "night";

    /**
     * Creates a GameObject to simulate night by overlaying a dark semi-transparent rectangle.
     *
     * @param windowDimensions The dimensions of the game window
     * @param cycleLength      The duration of a full day-night cycle
     * @return A GameObject representing the night effect
     */
    public static GameObject create(Vector2 windowDimensions, float cycleLength){
        GameObject night = new GameObject(Vector2.ZERO, windowDimensions,
                new RectangleRenderable(Color.BLACK));
        night.setCoordinateSpace(danogl.components.CoordinateSpace.CAMERA_COORDINATES);
        night.setTag(NIGHT_TAG);
        new Transition<Float>(night,
```

```
32                  night.renderer()::setOpaqueness,
33                  NOON_OPACITY,
34                  MIDNIGHT_OPACITY,
35                  danogl.components.Transition.
   CUBIC_INTERPOLATOR_FLOAT,
36                  cycleLength / 2f,
37                  Transition.TransitionType.
   TRANSITION_BACK_AND_FORTH, null);
38          return night;
39      }
40 }
41
```

```java
package pepse.world.daynight;

import danogl.GameObject;
import danogl.components.CoordinateSpace;
import danogl.gui.rendering.OvalRenderable;
import danogl.util.Vector2;

import java.awt.*;
/**
 * Creates a halo around the sun to simulate its glowing effect.
 * @author Eilam Soroka, Maayan Felig
 */
public class SunHalo {
    private static final Vector2 SUN_HALO_SIZE = new Vector2(128, 128);
    private static final String SUN_HALO_TAG = "sunHalo";
    private static final Color SUN_HALO_COLOR = new Color(255, 255, 0, 20);

    /**
     * Creates the sun's halo object that follows the sun's position.
     *
     * @param sun The sun GameObject that the halo should follow
     * @return A GameObject representing the sun's halo
     */
    public static GameObject create(GameObject sun){
        GameObject sunHalo = new GameObject(sun.getTopLeftCorner(), SUN_HALO_SIZE,
                new OvalRenderable(SUN_HALO_COLOR));
        sunHalo.setCoordinateSpace(CoordinateSpace.CAMERA_COORDINATES);
        sunHalo.setTag(SUN_HALO_TAG);
        sunHalo.addComponent(deltaTime -> sunHalo.setCenter(sun.getCenter()));
        return sunHalo;
    }
```

```
32 }
33
```