SWEN20003
Object Oriented Software Development
Workshop 4 (Solutions)

Eleanor McMurtry

Semester 2, 2020

Workshop

Questions

1. Study the `BagelTest` class, using the Bagel documentation available at https://people.eng.unimelb.edu.au/mcmurtrye/bagel-doc to understand how it works.

   (a) Look at the documentation for the `Image` class to understand how images are loaded and rendered.

   (b) Similarly, look at the documentation for the `Input` class to understand how keyboard and mouse input is handled.

2. Create a simple game following the below instructions.

   (a) When the game starts, our player should be rendered to the screen at the **point**: `(50, 350)`, and our house should be rendered at **point**: `(850, 180)`.

   - The player image is located at `res/player.png`, and the house image is located at `res/house.png`.

   (b) The player should be able to move left, right, up, and down, using the respective arrow keys at a constant speed (in pixels per frame). Try different values for the speed (a **constant**), starting at 1.

   (c) If the player comes within 55 pixels of the center of the house door, print to the console "Welcome home!". The center of the door is located at point: `(854, 268)`. (Remember, the distance between points (x_1, y_1) and (x_2, y_2) is given by $d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.)

   (d) Instead of printing the greeting to the console, draw the text to the screen using the **font** provided (`res/conformable.otf`). Draw it in size 24 font at coordinate (32, 32). (See the `Font` class in Bagel documentation.)

   (e) The game should exit when the Escape key is pressed.

   Solution:

```
import bagel.*;
import bagel.util.Point;

public class SimpleGame extends AbstractGame {
    private final Image playerImage = new Image("res/player.png");
    private final Image houseImage = new Image("res/house.png");
    private final Font font = new Font("res/conformable.otf", 24);

    private static final double SPEED = 0.5;
    private static final Point HOUSE_POINT = new Point(850, 180);
    private static final Point DOOR_POINT = new Point(854, 268);
    private static final Point TEXT_POINT = new Point(32, 32);

    private static final double GREET_DISTANCE = 55;

    private double playerX = 50;
    private double playerY = 350;
```

```
    @Override
    protected void update(Input input) {
        if (input.isDown(Keys.LEFT)) {
            playerX -= SPEED;
        }
        if (input.isDown(Keys.RIGHT)) {
            playerX += SPEED;
        }
        if (input.isDown(Keys.UP)) {
            playerY -= SPEED;
        }
        if (input.isDown(Keys.DOWN)) {
            playerY += SPEED;
        }

        if (new Point(playerX, playerY).distanceTo(DOOR_POINT) <= GREET_DISTANCE) {
            font.drawString("Welcome home!", TEXT_POINT.x, TEXT_POINT.y);
        }
        houseImage.draw(HOUSE_POINT.x, HOUSE_POINT.y);
        playerImage.draw(playerX, playerY);
    }

    public static void main(String[] args) {
        new SimpleGame().run();
    }
}
```

3. Here's a more complex game: **Catch the Ball**.

   (a) First, **clone** your workshop repository. Head to `https://gitlab.eng.unimelb.edu.au/swen20003-s2-2020/<us` and follow the steps in last week's lecture slides to clone the repository.

   (b) Copy the contents of `bagel-starter-pack` into the cloned repository. Run `git add .` to add the files, and `git commit -m "copied template"` to make your first commit. Then run `git push` to push your changes; you should commit and push for each of the following steps.

   (c) The player (using the same image as in Question 2) must move around the window catching a ball (`res/ball.png`). When the game starts, the ball should choose a random coordinate on the window to be drawn at.

   (d) When the player is within 24 pixels of the centre of the ball, the ball should move to another random coordinate.

   (e) Keep track of the player's **score**, which starts at 0 and increases by 1 every time the player catches the ball. Draw this score (using `Font`) at the top-left of the window.

   (f) Extend the code to make the ball choose a random diagonal direction, and move in that direction at a speed of 0.2 pixels per frame. When the ball reaches the left and right edges of the window, it should reverse horizontal direction. Similarly, when the ball reaches the top and bottom edges of the window, it should reverse vertical direction.

Solution:

```
// Ball.java
import bagel.Image;
import bagel.Window;
import bagel.util.Point;

public class Ball {
    private final Image ballImage = new Image("res/ball.png");

    private double x;
    private double y;

    private static final double SPEED = 0.2;
```

```
    private double xSpeed;
    private double ySpeed;

    public Ball() {
        resetPosition();
        // Chose a random direction
        xSpeed = (Math.random() > 0.5 ? 1 : -1) * SPEED;
        ySpeed = (Math.random() > 0.5 ? 1 : -1) * SPEED;
    }

    public Point getPoint() {
        return new Point(x, y);
    }

    public void resetPosition() {
        x = Math.random() * Window.getWidth();
        y = Math.random() * Window.getHeight();
    }

    public void update() {
        if (x < 0 || x > Window.getWidth()) {
            xSpeed *= -1;
        }
        if (y < 0 || y > Window.getHeight()) {
            ySpeed *= -1;
        }
        x += xSpeed;
        y += ySpeed;
        ballImage.draw(x, y);
    }
}
// CatchTheBall.java
import bagel.*;
import bagel.util.Point;

public class CatchTheBall extends AbstractGame {
    private final Image playerImage = new Image("res/player.png");
    private final Ball ball = new Ball();
    private final Font font = new Font("res/conformable.otf", 48);

    private static final double PLAYER_SPEED = 0.25;
    private static final double BALL_DISTANCE = 24;
    private double playerX = 512;
    private double playerY = 384;

    private int score = 0;

    @Override
    protected void update(Input input) {
        if (input.isDown(Keys.LEFT)) {
            playerX -= PLAYER_SPEED;
        }
        if (input.isDown(Keys.RIGHT)) {
            playerX += PLAYER_SPEED;
        }
        if (input.isDown(Keys.UP)) {
            playerY -= PLAYER_SPEED;
        }
        if (input.isDown(Keys.DOWN)) {
            playerY += PLAYER_SPEED;
```

```
        }

        if (new Point(playerX, playerY).distanceTo(ball.getPoint()) < BALL_DISTANCE) {
            ++score;
            ball.resetPosition();
        }

        ball.update();
        playerImage.draw(playerX, playerY);

        font.drawString("Score: " + score, 32, 32);
    }

    public static void main(String[] args) {
        new CatchTheBall().run();
    }
}
```

4. The below code (also on Canvas) contains bugs that stops it from working.

```
import java.util.Arrays;

public class VideoGame {
    public final String name;
    public final String platform;

    private static final VideoGame[] games = new VideoGame[100];
    private static int numGames = 0;

    public VideoGame(String name, String platform) {
        this.name = name;
        this.platform = platform;

        games[numGames++] = this;
    }

    public int compareTo(VideoGame other) {
        int platformCompare = platform.compareTo(other.platform);
        if (platformCompare != 0) {
            return platformCompare;
        } else {
            return name.compareTo(other.name);
        }
    }

    public static VideoGame[] getSortedVideoGames() {
        // Create a copy of the array to sort
        VideoGame[] sorted = Arrays.copyOf(games, games.length);

        // Recursively sort the array
        quickSort(sorted, 0, numGames - 1);
        return sorted;
    }

    private static void quickSort(VideoGame[] array, int low, int high) {
        if (low < high) {
            // Choose a pivot
            VideoGame pivot = array[high];

            // Partition the array
            int i = low;
```

```
            for (int j = low; j < high; ++j) {
                if (array[j].compareTo(pivot) < 0) {
                    // Swap the elements
                    VideoGame temp = array[i];
                    array[j] = array[i];
                    array[j] = temp;
                    ++i;
                }
            }

            // Swap the elements
            VideoGame temp = array[i];
            array[high] = array[i];
            array[i] = temp;

            quickSort(array, low, i - 1);
            quickSort(array, i + 1, high - 1);
        }
    }

    public String toString() {
        return String.format("%s (%s)", name, platform);
    }

    public static void main(String[] args) {
        new VideoGame("Halo", "Xbox");
        new VideoGame("Ocarina of Time", "Nintendo 64");
        new VideoGame("Red Dead Redemption 2", "Xbox One");
        new VideoGame("Majora's Mask", "Nintendo 64");

        System.out.println(Arrays.toString(VideoGame.getSortedVideoGames()));
    }
}
```

Use the **IntelliJ debugger** to help you find the bugs. Click to the right of the line number to set a **breakpoint**, and click the bug icon (next to the play icon) in the top-right; the code will **break** (pause) at the line you selected. Experiment with the features of the debugger view. Particularly helpful will be the **Variables** view and the **Step Over** button.

Solution: this is the corrected `quickSort` method.

```
private static void quickSort(VideoGame[] array, int low, int high) {
    if (low < high) {
        // Choose a pivot
        VideoGame pivot = array[high];

        // Partition the array
        int i = low;
        for (int j = low; j < high; ++j) {
            if (array[j].compareTo(pivot) < 0) {
                // Swap the elements
                VideoGame temp = array[i];
                array[i] = array[j];
                array[j] = temp;
                ++i;
            }
        }

        // Swap the elements
        VideoGame temp = array[i];
        array[i] = array[high];
        array[i] = temp;
```

```
        quickSort(array, low, i - 1);
        quickSort(array, i + 1, high);
    }
}
```