SWEN20003
Object Oriented Software Development
Workshop 4

Workshop

This week the focus is on learning to use the Bagel library. To get started, download the starter pack on Canvas and unzip it. Open the directory as an IntelliJ project, and run the main method in `BagelTest`. If you get errors, try right-clicking the file `pom.xml` and clicking `Add as Maven project`; this will force IntelliJ to download and set up the appropriate dependencies for Bagel. You can also try changing the JDK version to 17 or lower.

- In Bagel, the window is divided into **pixels**; by default, the window is 1024 pixels wide and 768 pixels high.
- Coordinates in Bagel are specified from the **top-left** of the window (unlike in usual mathematics[1], where the origin is the bottom-left). Therefore, by default
  - `(0, 0)` is the top-left
  - `(512, 384)` is the centre
  - (1024, 768) is the bottom-right
- The graphics in Bagel are handled by clearing the screen to blank many times per second, and re-rendering all of the graphics. One **clear-render** step is called a **frame**. The `update` method inherited from `AbstractGame` must be used to render the graphics, otherwise they will be erased at the beginning of the frame. (We will learn more about inheritance this week.)

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. All resources, e.g., the image and font files, are contained in the `res/` folder.
   (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. Now, you are ready to commit and push the `bagel-starter-pack` - refer to Week 3 Lectures or Workshop 3 for `GitLab` commands.
   (a) Create a new folder in the cloned repository, call it "Week 5" or "Workshop 4".
   (b) Copy all the contents of `bagel-starter-pack` into the new folder you just created.
   (c) Run `git add` to add the files, and `git commit -m "copied template"` to make your first commit.
   (d) Then, run `git push` to push to the remote repository; you should commit and push at the end of each question below.

[1] You can blame the Microsoft programmers Craig Eisler, Alex St. John, and Eric Engstrom who created DirectX (one of the first widely-used graphics libraries) in 1995 for this. They simply preferred "left-handed" coordinates.

3. Create the **catch the ball** game following the below instructions.

   (a) When the game starts, the player should be rendered to the screen at the **position**: `(200, 350)`, and the ball should be rendered at **position**: `(650, 180)`. You can use the `Point` class to define the position.

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

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

   (c) If the player comes within 20 pixels of the ball, we say the player catches the ball and you should print to the console "Great job!". Remember, the Euclidean distance between points (x_1, y_1) and (x_2, y_2) is given by:

$$d_{12} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{1}$$

   (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.

   (f) Now add, commit and push the code to the repository - remember to add a meaningful comment when you commit the code to describe your changes to the code.

4. Now, we are revising the catch the Ball game so that the player moves toward the ball step by step. In this question, you will learn how to set the player's moving direction and how to let the player **move towards this direction by one step**. Let the movement of the player controlled by the Enter key, not the arrow keys: If the Enter key is pressed (use `Input.wasPressed` method), do the following:

   (a) Calculate the direction (x_d, y_d) pointing from the player to the ball: for (x_1, y_1) and (x_2, y_2) being the positions of the player and the ball respectively, set[2]:

$$\begin{aligned} x_d &= \frac{x_2 - x_1}{\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}}, \\ y_d &= \frac{y_2 - y_1}{\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}} \end{aligned} \tag{2}$$

   (b) Moving in direction (x_d, y_d) by one step:

$$\begin{aligned} &\text{add } \texttt{STEP\_SIZE} * x_d \text{ to } x_1, \\ &\text{add } \texttt{STEP\_SIZE} * y_d \text{ to } y_1 \end{aligned} \tag{3}$$

   (c) After each movement, print to the console the position of the player in the form of

```
x-coordinate, y-coordinate
```

   Keep 2 decimal digits for both x-coordinate and y-coordinate. To do so, you can use the `format` method in `java.text.DecimalFormat` class, e.g., define a static attribute

```
private static DecimalFormat df = new DecimalFormat("0.00");
```

   and use it as

```
System.out.println(df.format(playerX) +"," + df.format(playerY));
```

   Check if the one-step movement is correct.

   (d) Now add, commit and push the code to the repository - remember to add a meaningful comment when you commit the code to describe your changes to the code.

5. Now, you will develop a more complex game of **Catch the Ball** game as follows.

   (a) The player (in Question 3) needs to catch a moving ball (`res/ball.png`). When the game starts, the ball should choose a random position on the window to be drawn at and a random direction. Set the step size of the ball to 0.5.

[2] It should be noted that, the same as position, a direction has both x- and y-coordinates. The denominators in Equation (2) equal to the Euclidean distance between (x_1, y_1) and (x_2, y_2) in Equation (1). By dividing this distance, the direction (x_d, y_d) in Equation (2) is normalized so that the movement in Equation (3) has a length equal `STEP_SIZE`.

(b) When the player is within 20 pixels of the ball, the ball should move to another random position.

(c) 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.

(d) Extend the code to make the ball choose a random diagonal direction, and move in that direction. 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. You can change the step size of player and ball when simulating.

(e) Now add, commit and push the code to the repository - remember to add a meaningful comment when you commit the code to describe your changes to the code.