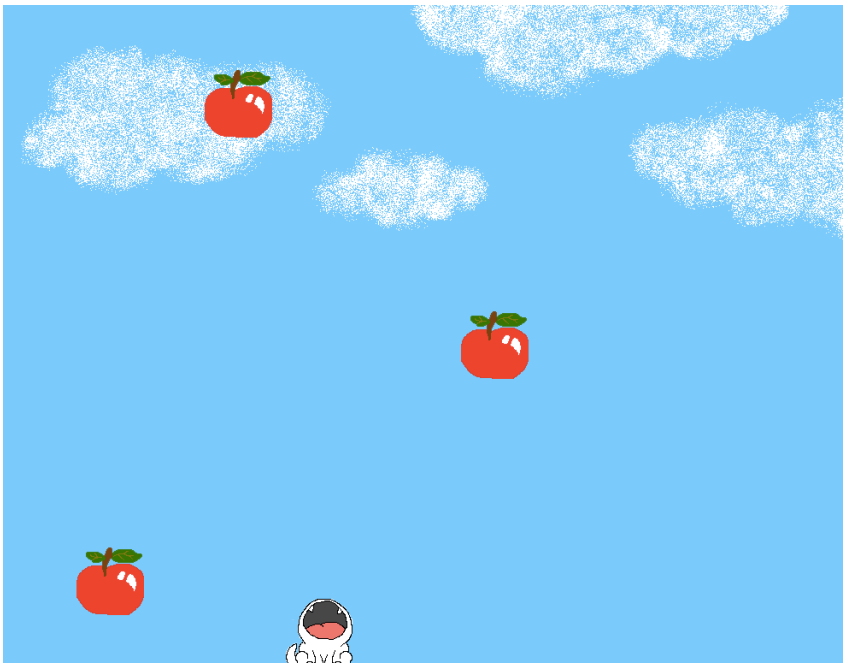
Customizing the Feed Me Game

This activity focuses on a more interesting program – a greedy apple-eating monster dog running around and eating apples that fall out of the sky!



Demo – you are the apple-eating monster dog! Catch as many apples as you can!

There are six coding files in this program, along with many image and sound files used in the game. Here's a summary of the files used in this game:

- `Game.java` – initializes a new game and runs it
- `SoundEffect.java` – manages the sound files used in the game
- `Board.java` – coordinates all the game pieces from start to end
- `Sprite.java` – manages the monster dog, how fast it moves, which direction it moves to when a specific arrow key is pressed
- `Drop.java` – manages a collection of falling apples and determines where and how many apples are dropped from the sky
- `Apple.java` – manages the falling object

- `Laser-01.wav, Move Forward86.wav` – audio files for sound effects and background music

- `Apple.png, doghappy.png, dogopen.png, sky.png` – image files for the objects and background in the game

Ideas to Personalize Your Game

1. **Different sound effects!**
   - In `SoundEffect.java`, you will see the following code:

     ```
     EAT( "laser-01.wav" );
     SONG( "Move Forward86.wav" );
     ```

   - You can change the file names to new `.wav` files so different sounds are played. You could also use a program (such as Audacity) to modify sound files.

   - Caution: use small files because Java can't handle large files. For example, the background music comes from the song `"Move Forward86.wav"` and it is about 1.5MB and the eating noise comes from `"laser-01.wav"` which is about 12KB.

2. **Different images!**
   - There are several images used in the program that you may want to change. If you want to modify an image, you could use a program (such as Paint or Gimp).

   - In `Board.java`, you can change the background image from `"sky.png"` to a new image of your choice. This is found inside the constructor method where "setup background" is done. Note that in `Game.java`, the size of the entire game is set to 1000 pixels (wide) x 800 pixels (tall). So depending on how big your new image is, you may want to adjust the size of the overall game too.

   - In `Apple.java`, you can change the apple image from `"apple.png"` to a new image of your choice (even if it's an orange or a cat or a meteoroid). This is found inside the constructor method where "initialize private variables" is done.

   - In `Sprite.java`, you can change the monster dog images from `"dogopen.png"` and `"doghappy.png"` to new images of your choice. This is found inside the constructor method where "initialize private variables" is done.

3. **Make apples fall faster!**
   - In `Apple.java`, near the beginning of the class, you will see the comment `"constants"`. Here, `FALL_SPEED` is defined as follows:

     ```
     private final int FALL_SPEED = 2;
     ```

   - You can change the number to a larger value so the apple falls faster!

4. **Drop more apples!**
   - In `Drop.java`, near the beginning of the class, you will see the comment `"constants"`. Here, `MAX` is defined as follows:

     ```
     private final static int MAX = 5;
     ```

- You can change the number to a larger value so there are more apples to catch!

5. **Change how quickly a new apple is dropped!**
   - In `Board.java`, inside the constructor method, you will see the line where the `dropper` is created like so:

     ```
     dropper = new Drop( 3 );
     ```

   - This indicates that at every 3 seconds, a new apple will drop from the sky, until all the apples have been dropped. You can change this time interval to a different value to make new apples drop sooner or later.

6. **Change where the monster dog starts!**
   - In `Sprite.java`, using the (x,y) coordinate system, the monster dog starts off at location (460,700) as defined in the constructor method. You can change this starting point to a new location.

7. **Make the monster dog move faster/slower!**
   - Inside the constructor method in `Sprite.java`, we see the variable `incr` which defines how many pixels the monster dog moves when we press the left and right arrow keys:

     ```
     incr = 5;
     ```

   - You can increase this value and make the monster dog move faster across the screen. However, note that if you make this value too big, the animation won't be smooth anymore, and the monster dog will look like it's hopping across the screen!

8. **Different ending credits!**
   - Now that you have personalized your game, you may want to say that you contributed to it. In `Board.java`, the credits are written inside the `paint` method. In this method, you can search for the part that checks to see if the game is over (search for the structure "`if ( gameOver ) { ... }`").

   - You will see that the two few lines set the font attributes (colour, font type and size). Your computer will have the commonly used fonts available. You can try changing these to different values and see how it looks!

   - Add your name to the line "`Game made by Bowen Hui and YOUR NAME`", or change it so it reads "`Game made by YOUR NAME (based on template from Bowen Hui)`".

   - If you use the same background music file, you will need to leave the music credits as is. If you end up changing the sound file, be sure to check what kind of copyrights is needed and whether you have to credit the artist. If you do, you will need to change the music credits accordingly.

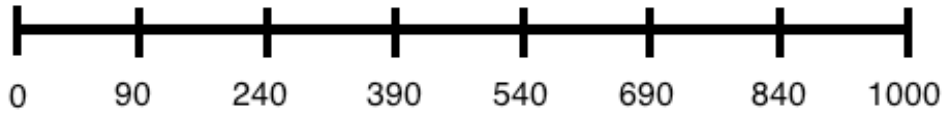
9. Change where apples are dropped! (ADVANCED)

- In `Drop.java`, near the beginning of the class, you will see the comment `"constants"`. Here, several variables are defined as follows:

```
private final static int NUMPOS = 6;
private final static int OFFSET = 90;
private final static int INCR   = 150;
```

- These variables are currently used in combination to create a series of 6 possible spots that the apples can de dropped. Once these spots have been created, the program then randomly picks one of them to drop an apple.

- Recall that the width of the entire board is 1000 pixels wide. The 6 possible spots created are 90, 240, …, 840, as shown by the ruler below:



- You can modify their values so the apples drop at different places!

10. Enable the monster dog to fly! (ADVANCED)

- In order to make the dog fly (to enable it to move up and down), we need to first understand how it currently moves left and right. In `Sprite.java`, the movements of the monster dog is determined by the arrow key presses. In particular, at the end of this file, there is a `keyPressed` method with the general structure as follows:

```
if( key == KeyEvent.VK_LEFT )    // left arrow key is pressed
{
  ...
}

if( key == KeyEvent.VK_RIGHT )   // right arrow key is pressed
{
  ...
}
```

- How much the monster dog should move is defined by `dx`. In the case when the left arrow key is pressed, the x position decrements to a lower value – unless it is already at the left edge of the screen, in which case, `dx = 0` so the monster dog won't move. Similar, when the right arrow key is pressed, the x position increments to a higher value – unless it is already at the right edge of the screen, in which case, `dx = 0`.

- In the keyReleased method, some house cleaning is done so to ensure that when we stopped pressing the arrow keys, dx is reset back to 0. Also, to ensure that the monster dog stays within the board, we use the following statements:

```
if( x < 0 )                  x = 0;
if( x > Board.BOARD_WIDTH ) x = Board.BOARD_WIDTH;
```

- Now, to enable the monster dog to move up and down, we need to do the following:
    - Add a `dy` variable to keep track of the change in `y` movements
    - Do this in the same way that `dx` is defined at the beginning of the class
    - Initialize `dy = 0;`
    - Do this in the same way that `dx` is initialized in the constructor method
    - In the `keyPressed` method, add two conditional statements to handle the up and down arrow keys respectively:

```
if( key == KeyEvent.VK_UP )
{
  ...
}

if( key == KeyEvent.VK_DOWN )
{
  ...
}
```

    - Replace the "`...`" with conditional statements similar to those for the left and right arrows.
    - In the `keyReleased` method, do the same kind of house cleaning by adding 2 conditional statements to reset `dy` and 2 conditional statements to ensure `y` is within the screen size.