

Introduction to Programming

Credit Task 1.2: Food Hunter



Overview

With arrays and records now in your toolkit, you can start to create some larger programs. In this task you will put these tools together to create a small game.

- Purpose:** Learn how to put the key programming tools together to create a program.
- Task:** Create the Food Hunter game.
- Time:** This task should be completed before the start of week 6, but may be completed later (see following note).
- Resources:**
- Chapter 6 of the Programming Arcana
 - Swinburne CodeCasts ([YouTube Channel](#), [iTunesU](#))
 - [Using arrays to work with multiple values](#)
 - [Dynamically changing the size of an array](#)
 - Syntax Videos
 - [Pass by Reference \(Var Parameters\)](#), [Pass by Reference \(Const Parameters\)](#), [Pass by Reference \(Out Parameters\)](#), [Arrays](#), [For Loop](#), [Dynamic arrays](#)

Note: If you are not currently up to date you should skip this task and return to it once you are up to date with the Pass Tasks. Do not allow Credit Tasks to delay you in keeping up with the unit's Pass Tasks.

Submission Details

You must submit the following files to Doubtfire:

- Code for the program, and a screenshot of it working at the terminal.

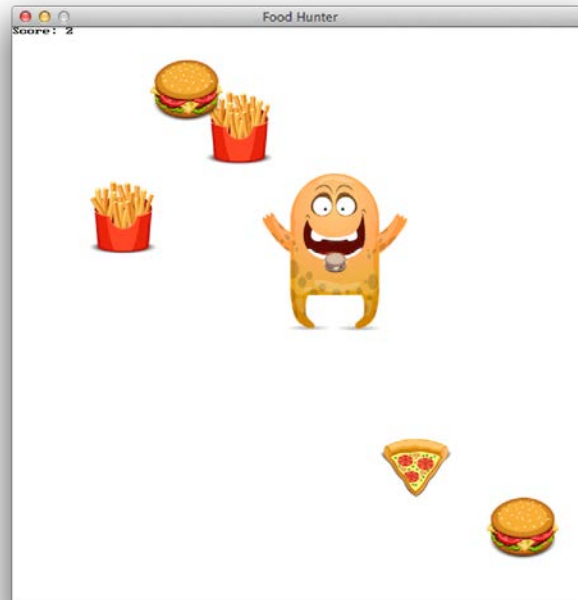
Make sure that your task has the following in your submission:

- Code must follow the Pascal coding convention used in the unit (layout, and use of case).
- You are storing and working with multiple values in an array.
- You are using a record and enumeration to store the values.
- The code must compile and the screenshot show it working, and the validations in action.

Instructions

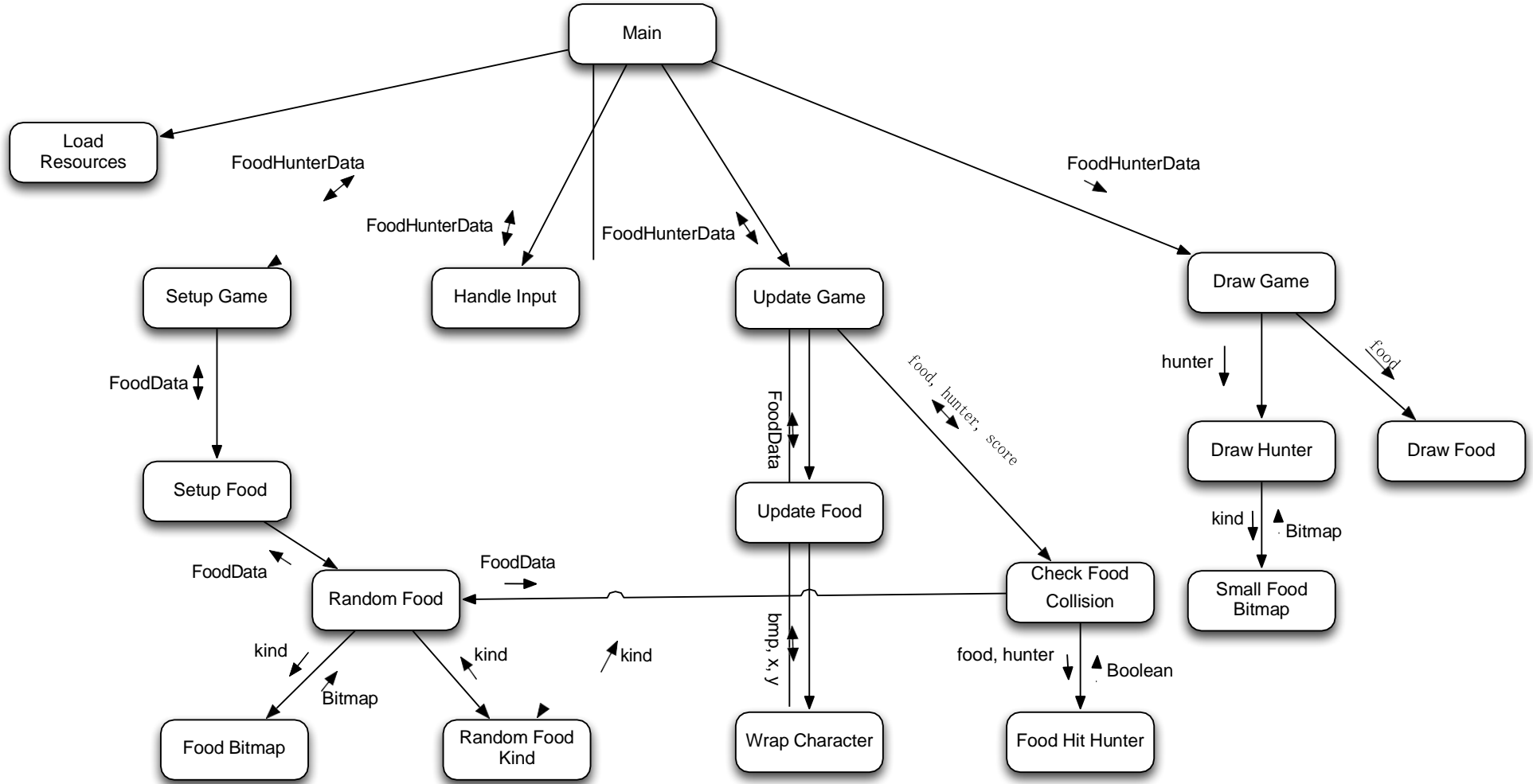
In this task we are going to create a small computer game "Food Hunter".

The game will have the player control a character and chase food around the screen. The player will aim to eat one kind of food and dodge the others. The kinds of food include Burger, Chips, Ice-cream, and Pizza.



At this point another developer has started the project, and it is progressing well. The developer has left some notes in the text and a structure chart, but no other documentation. Currently the main issue is that there is only one item of food in the game for the player to chase, the full list of issues is presented below.

1. The player should be able to wrap around the screen like the food does. For example, when the player goes off the right of the screen they should reappear on the left.
2. The food is only moving left/right. It should be moving up/down, left/right or on diagonals.
3. There is only a single item of food.



1. Download the starter code for the **Food Hunter** program, and extract from the zip file.
2. Read through the code and check the functionality of each function and procedure.

Tip: If you are not sure how something is working ask a question on Blackboard.

3. Open a Terminal window and compile and run the program. Verify the three issues are present.

Once you have noted the issues we can start working on the first bug. The player needs to be able to wrap around the screen.

4. Read **UpdateFood** and determine how the food is currently being updated so that it wraps around the screen.
5. Examining the code in the **HandleInput** procedure, and alter it to enable the game's hunter to wrap around the screen in a similar way to the food items.

Hint: The fix for this should involve a single procedure call - code reuse is great!

6. Open a Terminal window and compile and run the program.
7. Make sure you can move the player off one side of the screen and have it appear on the other side. Verify this works for all four directions.
 - 7.1. Identify the issue with the current code.
 - 7.2. Correct the issue you identify.
 - 7.3. Compile and run the program and test it again to make sure it works.

Note: Make sure that you can wrap the character around the screen in all directions.

1. When going off the screen to the left, the player should reappear on the right.
2. When going off the screen to the right, the player should reappear on the left.
3. When going off the screen to the top, the player should reappear at the bottom.
4. When going off the screen to the bottom, the player should reappear at the top.

The second issue relates to the food only moving left/right.

8. This issue is related to updating of the food so start by reading the **Update Game** procedure, and the procedures it calls. Locate where FoodData is updated and ensure that it is changing the food's x and y values.

The food's y value needs to be updated by the food's dy value. The dy (delta y) represents the distance that the food item moves in the y direction each time the food is updated. Storing this as a value allows each item of food to move at a different pace.

9. Once you have verified that y is being updated correctly, the other potential issue is that dy is not set correctly. This value is assigned when the FoodData is initialised. Review **RandomFood** and check that dy is being assigned a random value.
10. When you have identified the cause of the problem, and corrected it, switch back to the Terminal and compile and run your program.
11. Test that the food items are now moving in all directions (left/right, up/down, or diagonal).

Now for the more important issue, there should be multiple food items on the screen at a time. This will require changes in a number of places, but most of the code should remain as is.

12. Locate the definition of the **FoodHunterData** record.
13. Change the food field to be an **array of FoodData**, rather than a single FoodData item.
14. Locate **DrawGame** change it to draw each element of the game's food array:
 - Add an i variable to track the current index of the array.
 - Add a for loop that loops from 0 to the last element of the array, i.e. High(game.food). This should just repeat the drawing of the food item, nothing else.
 - Change the code to access the ith element of the game's food array.
15. Locate **UpdateGame** and change it to update each element of the game's food array:
 - Add an i variable to track the current index of the array.
 - Add a for loop that loops from 0 to the last element of the array. This should just repeat the updating of the food item and checking food collision, nothing else.
 - Change the code to access the ith element of the game's food array for both the call to **UpdateFood** and **CheckFoodCollision**.

16. Locate **SetupFood** and change it to have the following logic:

```
procedure: Setup Food
-----
Parameters:
- game : a variable reference to a FoodHunterData variable
- numFood : an Integer that represents the number of food
            items to have in the game.
Local Variables:
- i : an Integer to track the current index of the array
-----
Steps:
1: Set the Length of game's food, to numFood
2: For each element of game's food
3:   Assign game's  $i^{\text{th}}$  food element, the result of calling
    Random Food
```

17. Switch to the terminal and compile and run the program. You should be able to see multiple food items.

Your program should now be complete.