

In-Class Activity #5: Digital Pet App



Due: 02/19/2026 – 7:59 pm



Submit: l-college Drop-box



Topic: Introduction to State Management

Demonstrate the key concepts of state management in Flutter

using **StatefulWidget** to bring a digital pet to life!



Assignment Overview

The goal of this Flutter application is to create a digital pet simulation where users can interact with a virtual pet and manage its state. This project demonstrates the key concept of state management in Flutter using **StatefulWidget** and the **State** class. Users can perform actions like playing with or feeding the pet, which updates the pet's happiness and hunger levels dynamically.

State Management 101: In Flutter, state management is the process of handling the mutable state of a widget. **StatefulWidget** is used to create widgets that can dynamically change their appearance or behavior over time. The **State**

class is responsible for maintaining the mutable state of the widget, and the `setState` method is used to notify the framework when the state has changed.

Learning Objectives

By completing this assignment, you will:

- ✨ Understand the difference between `StatefulWidget` and `StatelessWidget`
- 🔄 Master the use of `setState()` to trigger UI updates when data changes
- 🎨 Learn to dynamically modify widget properties (colors, text, images) based on app state
- ⌚ Implement time-based events using `Timer` from `dart:async`
- 🖼️ Efficiently manage image assets using `ColorFiltered` widget
- 🎮 Create interactive user experiences with conditional logic and game mechanics
- 📁 Properly configure and use assets in Flutter projects via `pubspec.yaml`
- 🏗️ Structure state management logic within a Flutter application

Step-by-Step Implementation Guide

Follow these steps to successfully complete your Digital Pet App:

Step 1: Set Up Your Flutter Project

1. Open your terminal/command prompt
2. Run: `flutter create digital_pet_app`
3. Navigate into the project: `cd digital_pet_app`
4. Open the project in your preferred IDE (VS Code, Android Studio, etc.)

Step 2: Prepare Your Assets

1. Create a folder named `assets` in your project root directory
2. Find a transparent PNG image of a pet (dog, cat, bird, etc.) online or create your own
3. Save the image in the `assets` folder as `pet_image.png`
4. Open `pubspec.yaml` and add under the `flutter:` section:

```
flutter:  
  assets:  
    - assets/pet_image.png
```

5. Run `flutter pub get` in the terminal



Step 3: Copy the Starter Code

Navigate to `lib/main.dart` and replace all the existing code with the starter code provided below.



Step 4: Implement Part 1 Features

Add the required features one at a time:

1. **Dynamic Color Change:** Use the `ColorFiltered` widget (see hint below)
2. **Mood Indicator:** Create a helper method to return mood text and emoji
3. **Name Customization:** Add a `TextField` and button above the pet display
4. **Auto-Hunger Timer:** Import `dart:async` and use `Timer.periodic`
5. **Win/Loss Logic:** Add conditional checks and display dialogs or text



Step 5: Add Part 2 Advanced Features

Choose 2 features (undergrad) or 3 features (grad) and implement them carefully.

Step 6: Test Your App

1. Run your app: `flutter run`
2. Test all buttons and interactions
3. Verify the timer is working (hunger increases automatically)
4. Check color changes based on happiness levels
5. Test win/loss conditions

Step 7: Build Your APK

1. Run: `flutter build apk`
2. Find your APK at: `build/app/outputs/flutter-apk/app-release.apk`

Step 8: Push to GitHub

1. Initialize git: `git init`
2. Add your files: `git add .`
3. Commit: `git commit -m "Initial commit - Digital Pet App"`
4. Create a new repository on GitHub
5. Connect and push: Follow GitHub's instructions to push your code
6. Take a screenshot of your commits



The Starter Code

Copy this code into `lib/main.dart` . Click the "Copy Code" button below for easy copying!

```
import 'package:flutter/material.dart';

void main() {
  runApp(MaterialApp(
    home: DigitalPetApp(),
  ));
}

class DigitalPetApp extends StatefulWidget {
  @override
  _DigitalPetAppState createState() => _DigitalPetAppState();
}

class _DigitalPetAppState extends State<DigitalPetApp> {
  String petName = "Your Pet";
  int happinessLevel = 50;
  int hungerLevel = 50;

  void _playWithPet() {
    setState(() {
      happinessLevel += 10;
      _updateHunger();
    });
  }

  void _feedPet() {
    setState(() {
      hungerLevel -= 10;
      _updateHappiness();
    });
  }

  void _updateHappiness() {
    if (hungerLevel < 30) {
      happinessLevel -= 20;
    } else {
      happinessLevel += 10;
    }
  }
}
```



```

    }

    void _updateHunger() {
      setState(() {
        hungerLevel += 5;
        if (hungerLevel > 100) {
          hungerLevel = 100;
          happinessLevel -= 20;
        }
      });
    }

    @override
    Widget build(BuildContext context) {
      return Scaffold(
        appBar: AppBar(
          title: Text('Digital Pet'),
        ),
        body: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: <Widget>[
              Text('Name: $petName', style: TextStyle(fontSize: 20.0)),
              SizedBox(height: 16.0),
              Text('Happiness Level: $happinessLevel', style:
TextStyle(fontSize: 20.0)),
              SizedBox(height: 16.0),
              Text('Hunger Level: $hungerLevel', style:
TextStyle(fontSize: 20.0)),
              SizedBox(height: 32.0),
              ElevatedButton(
                onPressed: _playWithPet,
                child: Text('Play with Your Pet'),
              ),
              SizedBox(height: 16.0),
              ElevatedButton(
                onPressed: _feedPet,
                child: Text('Feed Your Pet'),
              ),
            ],
          ),
        ),
      ),
    ),
  ),

```

```
);  
}  
}
```



Part 1: Required Features (All Students)

Implement the following logic to make the app interactive and dynamic.



Dynamic Pet Color Change: Change the pet image color based on happiness.

Happy (>70): Green | Neutral (30-70): Yellow | Unhappy (<30): Red



Pet Mood Indicator: Display a text indicator (e.g., "Happy", "Neutral") and a corresponding emoji next to the pet.



Pet Name Customization: Add a text input field and button to allow users to set a custom name for their pet.



Auto-Increasing Hunger: Use `Timer` (from `dart:async`) to increase hunger every 30 seconds automatically.



Win/Loss Conditions:

Win: Happiness > 80 for 3 minutes.

Loss: Hunger reaches 100 AND Happiness drops to 10 (Show "Game Over").



How to use the `ColorFiltered` Widget:

Instead of swapping out multiple images, use `ColorFiltered` to tint a single asset. This is more efficient!

```
Color _moodColor(double happinessLevel) {  
  if (happinessLevel > 70) {  
    return Colors.green;  
  } else if (happinessLevel >= 30) {
```

```
        return Colors.yellow;
    } else {
        return Colors.red;
    }
}

// In your build method:
ColorFiltered(
  colorFilter: ColorFilter.mode(
    _moodColor(happinessLevel),
    BlendMode.modulate,
  ),
  child: Image.asset('assets/pet_image.png'),
)
```

⚡ Part 2: Advanced Features

Undergraduates: Choose any 2 features. **Graduates:** Choose any 3 features.

1. **Add Energy Bar Widget:** Introduce a `LinearProgressIndicator` or custom widget to visually represent energy.
2. **Implement Energy Level Logic:** Define `_energyLevel` state. Logic should decrease energy during activities.
3. **Activity Selection:** Add a dropdown menu to select specific activities (e.g., "Run", "Sleep").
4. **Update Pet State:** Logic to update happiness/energy based on the specific activity selected (e.g., Sleeping increases energy but might increase hunger).

🔧 Helpful Hints & Guides

⚠ Important: Asset Management

For the `Image.asset` to work, you must:

1. Create an `assets` folder in your project root.

2. Add a transparent PNG of a pet to that folder.
3. Update your `pubspec.yaml` file to include the asset:

```
flutter:  
  assets:  
    - assets/pet_image.png
```

Remember to run `flutter pub get` after changing the yaml file!

Understanding `setState()`

When you call `setState()`, you are telling Flutter that "something has changed" in the internal state of this object. This causes the framework to schedule a build for this State object. If you just change `happinessLevel = 90` without wrapping it in `setState()`, the variable changes, but the UI on the screen will not update to reflect it.

Critical Thinking Questions

Answer the following questions in a separate Word document to deepen your understanding.

1. Why do we need to extend `StatefulWidget` instead of `StatelessWidget` for this application? What specific feature makes the difference?
2. In the `_updateHunger` method, why is `setState` called? What would happen visually if the logic ran without it?
3. Explain how `BlendMode.modulate` works mathematically regarding the RGB channels of the image and the color filter.
4. We used a `Timer` for the auto-hunger feature. If the user navigates away from this screen (pops the route), what must we do with the Timer to avoid memory leaks?
5. Using `ColorFiltered` saves us from having multiple image assets (`red_pet.png`, `green_pet.png`). How does this impact the app size and

memory usage?

6. If we wanted to persist the pet's happiness level even after the app is closed and reopened, what technologies or packages would we need to add?

7. Look at the `build` method. Every time `setState` is called, the entire widget tree inside `build` is reconstructed. Is this efficient? Why does Flutter operate this way?

8. In a real-world app, would you keep all the logic (feeding, playing, timer) inside the UI file (`main.dart`)? If not, how would you structure the code?

9. What happens if the `_moodColor` function returns `null`? How does Dart's null safety help here?

10. If we added a feature where the pet "evolves" (changes image entirely) at happiness level 100, would `ColorFiltered` still be the best approach? Why or why not?



Submission Instructions

 **Due Date: 02/18/2026 @ 11:59 pm**

Late submissions will NOT be accepted beyond this deadline.



Submission Requirements

1 GitHub Repository URL (via Text File)

Upload your complete project to GitHub and submit the direct link to your repository.

- Create a text file named: `github_link.txt`
- Inside the file, paste your GitHub repository URL

- Example: https://github.com/yourusername/digital_pet_app

2 APK File

Attach your compiled Android APK file.

- Build your APK: `flutter build apk`
- Locate: `build/app/outputs/flutter-apk/app-release.apk`
- Rename it to: `YourName_DigitalPetApp.apk`

3 Critical Thinking Answers (Word Document)

Answer all critical thinking questions in a Word document.

- File name: `YourName_CriticalThinking.docx`
- Include your name and student ID at the top
- Answer all 10 questions thoroughly

4 Upload to iCollege

Submit all three items to the correct assignment folder:

- Navigate to **In Class ACT 05** folder in iCollege
- Upload: `github_link.txt`
- Upload: `YourName_DigitalPetApp.apk`
- Upload: `YourName_CriticalThinking.docx`
- Verify all files are properly uploaded before the deadline

✓ Pre-Submission Checklist

- ✓ GitHub repository is public and accessible



`main.dart` has comments with your name (and partner's name if consulting)



Screenshot of Git commits is included in your repository README



APK file has been tested on an Android device or emulator



All Part 1 features are implemented and working



Correct number of Part 2 features completed (2 for undergrad, 3 for grad)



Critical Thinking questions answered completely



All files follow the naming conventions specified above



Submission made to iCollege before **02/18/2026 @ 2:30 pm**



Good luck! Let's make this pixel pet happy! 🐶 🐱

Remember: Your digital pet may not be real, but its hunger for attention definitely is!