MVC

Model-View-Controller Architecture

Lecture Question

Complete the MVC structures chat app in the examples repository

- In the CSE116-Examples repo, find the package named mvc with classes Model, View, and Controller
- This is the start of a chat app with no functionality. Your task is to complete it
- Model API (You must implement these methods)
    - sendMessage(String): Unit
        - Called when a/the user sends a message
    - allMessages(): List[String]
        - Returns all sent messages (order doesn't matter for testing)
- Controller (You must implement the handle method)
    - Action event handler called when the user clicks the send button
    - Contains references to the text typed by the user and the model
- View (Do not edit the view)
    - Displays GUI elements to the user and calls allMessages() to display latest data

MVC

- Software **architecture pattern**
  - A way to organize the code of an entire project
  - As opposed to **design patterns** which solve a specific problems within a project
- Separate code into a Model, View, and Controller
  - Decouple the project into 3 pieces
- All three parts work independently and communicate with each other through APIs

API

- In CSE115 you've seen **web** APIs which have various endpoints
- An API is a set of functions/methods that can be called
- In the state pattern, define an API for the state
  - These are the methods that can be called and are deferred to the current state for functionality
  - Other classes only look at the API and call those methods
    - List of ways of interacting with the object
- The methods of any class/object define an API
- Can change functionality, just don't change the API
  - Ex. We can add dynamic collisions to the physics engine. If we don't change the way updateWorld is called, all games using the engine now have dynamic collisions
- Changing the API (ex. changing a return type from Double to Int) will break all code using that API

API

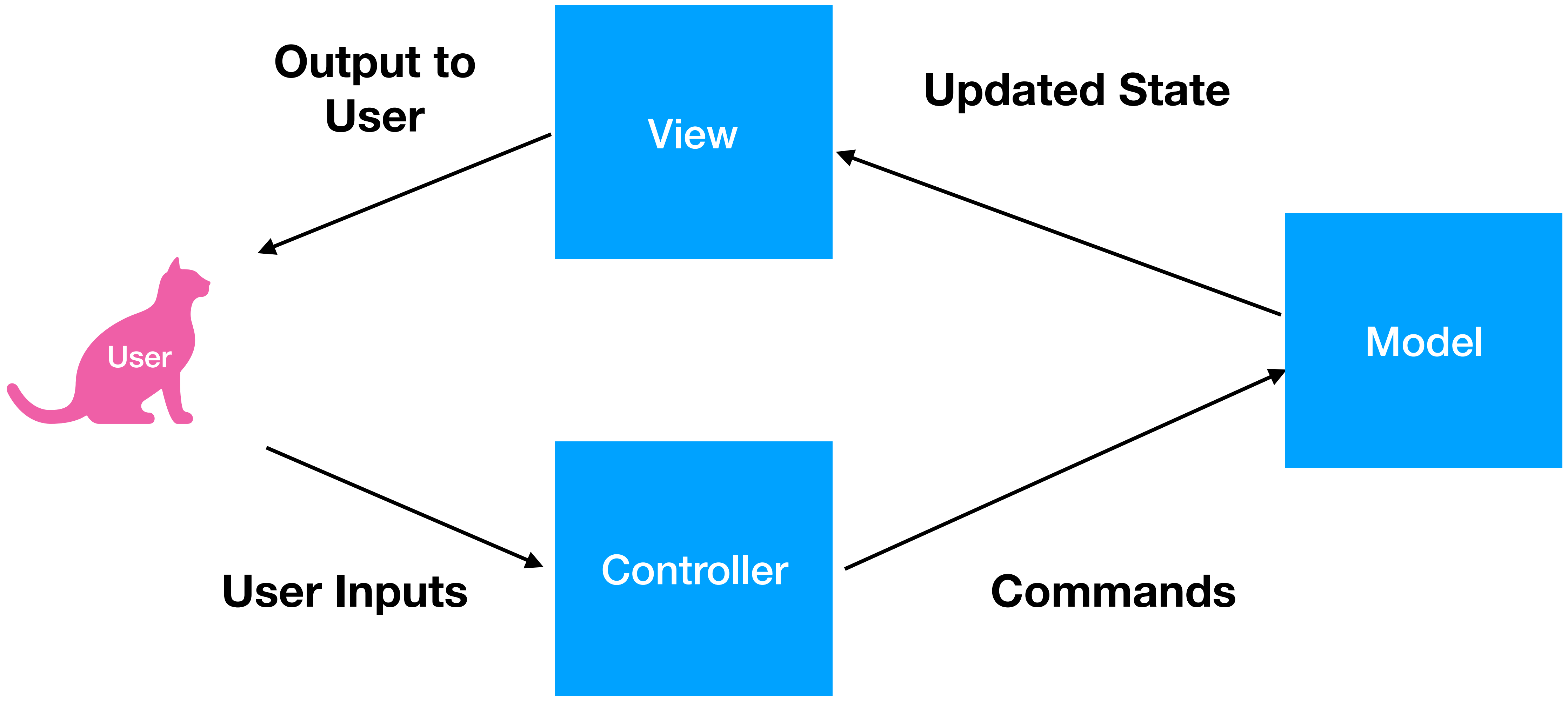
- Can change functionality if the API remains the same
- Example:
    - We can add dynamic collisions to the physics engine
    - If we don't change the way updateWorld is called, all games using the engine now have dynamic collisions

- Changing the API will break all code using that API
- Example: Changing a return type from Double to Int will cause type mismatch errors in any code calling that API method

MVC

- Model (Data and Logic)
  - Controls the app and its data
  - The core of the app
- View (Display)
  - Visualizes the app
  - No logic
- Control (User Inputs)
  - Handles user inputs
  - Calls model API methods based on inputs

MVC


Output to User
View
Updated State
User
Model
Controller
User Inputs
Commands

MVC - Model

- The core of the app
- Most of the code you've written so far in CSE115/116 is part of a model
    - Physics Engine handling how objects move
    - Calculator logic when buttons are pressed
- Controls the logic and functionality of the app
- Maintains the data
    - Controls any data structure, databases, and files related to how the program behaves
- **Has no knowledge of the user of the app**
- Functionality accessed through an API

MVC - View

- Displays the state of the app to the user
- **Output only**
- No logic
    - The view cannot change the state of the app
- Since the view is output only and does not alter the app, it can change and be replaced without affecting the app itself
- Can test the logic of an app without using the view
- Can have the same app with a CLI (command line interface) and a GUI (graphical user interface)
- Can have the same app with a web front-end and a desktop front-end!

MVC - Controller

- Handles user inputs
- In ScalaFX, defined by EventHandlers
- Processes user inputs and converts them into calls of the model API
- Can validate and block invalid inputs
- Acts as a barrier between the GUI and the model
  - If the GUI changes, replace view and controller and model remains unchanged

MVC - Advantages

- Focus on 1 part of a project at a time
    - Reduce spaghetti code
    - Divide work among team members
        - Just agree on the APIs
- Views can be easily replaced
- Keeps code organized
- Easier to add new features
    - Model can add features as long as API remains unchanged

MVC on the Web

- Model runs on the server
- View runs in the browser (HTML/CSS)
- Controller can run on both
  - JavaScript in the browser converts user inputs into AJAX requests
  - Server validates the data and sends the commands to the Model

MVC - Jumper

- Model API
    - Left, right, jump pressed for each player
    - Allows view to access all data
- Controller
    - Convert W, A, D, ←, ↑, → key presses into model API calls
- View
    - Displays all game objects to the player
    - Receives absolute locations of all objects from model
        - Computes vertical scroll and translates objects accordingly

MVC - Calculator

- Model API
  - Up to you to decide which methods are called by the controller (Can correlate with the button presses directly)
  - displayNumber(): Double is called by the view to determine what should be displayed to the user
- Controller
  - Each button on the calculator has an event handle for you to implement
  - Testing is done through these handlers to allow you to design your own model API
- View
  - Uses a grid pane for more control over element placement
  - Separate CalculatorButton class to easily change the appearance of all buttons
  - Calls displayNumber to update the display whenever the mouse is clicked on the GUI

MVC - Calculator

- Model is not aware of ScalaFX
- If we want to build a GUI using a different library
  - No need to change the model at all
  - Build a new view and controller to call the same model API methods

MVC - Project

- LA2, demo 1, LA4 are all part of your model
- Demo 2 is all about your view [and part of your controller]
- Your view in demo 2 will access from the model JSON strings containing
    - The overworld map
    - All party locations and state on the map
    - Status of all characters in each battle
- Your controller for demo 2 will send to the model
    - User inputs on the overworld map
    - Actions chosen by the player in battle

MVC - Project

- The rest of the controller and model will be developed for demo 3
- The model will be networked and be able to handle inputs from the desktop and web versions
  - Model won't even know if an input is from the web or desktop version
- Two different GUIs are completely compatible using MVC as long as they both use the model API
- Communication abstracted through JSON strings
  - GUIs are written in 2 different languages so we can't use language specific types

LQ - Chat App

- For the lecture question you'll finish a chat app using MVC
    - The chat app allows a user to send a message by clicking a send button
    - The model API contains a method for sending a message (Model is unaware of the send button)
    - Controller listens for the button press and calls the appropriate model API method
- If we want to allow the user to also press enter to send a message:
    - Only change the controller to listen to the enter key press and call the model API method
    - Model only cares that a message was sent. It does not care how it was sent

Lecture Question

Complete the MVC structures chat app in the examples repository

- In the CSE116-Examples repo, find the package named mvc with classes Model, View, and Controller
- This is the start of a chat app with no functionality. Your task is to complete it
- Model API (You must implement these methods)
    - sendMessage(String): Unit
        - Called when a/the user sends a message
    - allMessages(): List[String]
        - Returns all sent messages (order doesn't matter for testing)
- Controller (You must implement the handle method)
    - Action event handler called when the user clicks the send button
    - Contains references to the text typed by the user and the model
- View (Do not edit the view)
    - Displays GUI elements to the user and calls allMessages() to display latest data