APCSP - Part01

Logical Selection, Mouse Interaction and Buttons with Game State

Kai kai@42.us.org

Summary: Start creating interaction and variable program states by responding to mouse clicks.

Contents

Chapter I

Goals for Today

When building a computer game, it is super nice to have a menu at the start where the player can click buttons to select their difficulty level, number of players, or a button that triggers a help screen to appear.

Today you'll learn everything you need to set up some clickable buttons that can modify the state of your canvas or trigger some text to print to the screen.

I.1 Turning in Your Work

You should turn your work into Vogsphere like so:

1. After closing your team for the project and opening your grading slots for the end of the day, clone your repository from Vogsphere.

2. Create a text document inside your project folder, and paste the links to your Processing sketches for this week into the text document.

3. Push your project folder to Vogsphere before setting the project as finished.

4. Sign up for corrections on intra.

Additional detail about how to use Vogsphere is on the 'Hello 42! Hello Terminal!' PDF.

Chapter II

The Building Blocks of Algorithms

An essential concept to remember for the APCSP curriculum is that `algorithms` are made up of `sequencing`, `selection` and `iteration`.

II.1 Sequencing

refers to the fact that when we write a program, each line of code will execute its action in the order that we write them. As you write code, think about the order in which things happen. In Processing, the setup() function runs once, and then the draw() function repeats many times over and over again. Inside the draw() function, everything happens in the order in which you write it.

II.1.1 Sequencing Example

Draw a rectangle on your canvas, and then draw a triangle overlapping with it, in a different color. How is that different than drawing the triangle first and then the rectangle?

II.2 Selection

is what today's lesson is all about. We can use logical statements in our programs to make the computer choose different options depending on where you click, what you type, or what input is randomly generated. The AP test calls this `selection` and most programmers call it `"if/else statements"` or `"conditionals"`.

II.3 Iteration

refers to repeating something over and over again, like the draw() function. We'll talk about how to use that in algorithms next week.

Chapter III

Selection

The best place to learn is the official documentation on if..else in Javascript.

III.1 A single if

The `If` statement is used in practically every computer science language to create a decision point in the code.

In plain english, when you see an "if" you can read it as: "If the following condition is true, Then do the following code."

```
var num_players = 2; // Number of players has been set to 2.
if (num_players === 1) { // If the numbers of players is 1,
    text("Single player mode", 10, 20); // Then tell the player that we are in single player mode.
}

var difficulty = 1; // Difficulty has been set to 1.
if (difficulty === 1) { // If the difficulty setting is 1,
    text("Difficulty: Easy", 10, 40); // Then tell the player that we are in easy mode.
}
```

III.2 Comparison Operators

List of the Conditional Operators in Javascript

Inside every "if" statement is a condition:

```
// This is not working code, just a pattern!
if (condition) {
    code;
}
```

The condition statement is a logical expression that always evaluates as `true` or `false`. We can use some familiar symbols from math class to generate the logical expressions. aterrizadoaterrizado Equals ("===") or not equals ("!==") is useful for any data type, including both strings and numbers. Greater than (">" and ">=") or less than ("<" and

"<=") operators should only be used with numbers.

The following snippets are examples. You can write them into your code to test them, but no need to include them with the homework submission.

```
var role = "teacher";
if (role === "teacher") {
    print("Here is the answer key");
}
```

```
var gameOver = true;
if (gameOver !== true) {
    print("Play another turn:");
}
```

```
var turn_number = 11;
if (turn_number >= 10) {
    text("Game Over!", 10, 20);
}
```

```
var lives = 0;
if (lives === 0) {
    text("Game Over!", 10, 20);
}
```

III.3 Booleans

`true` and `false` are special keywords: they are `Booleans` named after the infamous George Boole.

A variable can be set to true or false just as easily as it can be set to a number or a string. You don't need to put true or false in quotes.

If you put a boolean into the condition of an if statement, you don't need an equals sign. The if statment will run only if the boolean is true.

```
var gameOver = true;
if (gameOver) {
    text("Game Over!", 10, 20);
}
```

III.4 Logical Operators

There are three logical operators which create 'logical equations' for booleans. They are `and (&&)`, `or (||)`, and `not (!)`.

AND evaluates as true if either of the booleans being compared are true.

true && false = false
false && true = false
true & true = true
false & false = false

```
var gameOver = true;
var win = true;
if (gameOver && win) {
    text("Congratulations! You win!", 10, 20);
}
```

OR evaluates as true only if both of the booleans being compared are true.

true || false = true
false || true = true
true || true = true
false || false = false

```
var quit = true;
var gameOver = false;
if (gameOver || quit) {
    print("Exiting game");
}
```

NOT flips the boolean following it to the opposite value.

!true = false
!false = true

```
var available = false;
if (!available) {
    print("Area is available");
}
```

III.5 Combining logical operators

With parentheses you can combine logical operators just like you combine mathematical operators.

```
var available = true;
var gameOver = false;
var infiniteplay = true;
if (available && (!gameOver || infiniteplay)) {
    print("Area is available");
}
```

You can also combine these expressions together with comparison operators.

```
var numturns = 2;
var points = 100;
var easy_mode = false;
var single_player = true;
if ((single_player && easy_mode) || (numturns < 2 && points < 99)) {
    print("Take another turn");
}
```

III.6 Multiple ifs vs if/else

Multiple ifs: If the code has one if statement after another, the result of one if statement has no effect over the result of the next one. They could both be true, or only one of them, or neither.

```
var score = 9000;
var player_highscore = 420;
var all_time_highscore = 4200;
if (score > player_highscore){
    print("Awesome! New personal best.");
}
if (score > all_time_highscore) {
    print("Amazing! Add your name to the Hall of Fame.");
}
```

If/Else: The if/else construction lets you create two mutually exclusive options. Instead of this:

```
var number = 7;
if (number < 0) {
    print("Your number is negative.")
}
if (number >= 0){
    print("Your number is positive.")
}
```

...write this:

```
var number = 7;
if (number < 0) {
    print("Your number is negative.")
} else {
    print("Your number is positive.")
}
```

And, if you want to add even more mutually exclusive options, you can add an "else if".

```
var number = 7;
if (number < 0) {
    print("Your number is negative.")
} else if (number === 0) {
    print("Your number is zero.")
} else {
    print("Your number is positive.")
}
```

Chapter IV

MouseClick and other Functions

IV.1 MouseClicked

Take a look at the p5js documentation on events, specifically mouse events. We'll focus on the mouseClicked() function to start although you may find other ones useful later on.

When you click the mouse on a Processing project, the web browser (Safari or Firefox or Chrome etc) will send a "mouseclick" event to the code. Processing responds to the mouseclick by running whatever code, if any, you have provided inside of the mouseClicked() function.

In other words, mouseClicked() is a function that you can fill out alongside setup() and draw() which will run whenever the mouse is clicked anywhere inside your canvas.

IV.2 MouseX and MouseY

No matter whether the mouse has been clicked or not, you always have access to the mouseX and mouseY variables which will describe where the mouse currently is inside of your canvas.

Try this program to demonstrate:

```
function setup() {
    createCanvas(400, 400);
}

function mouseClicked() {
    print(`The mouse was clicked at ${mouseX}, ${mouseY}`);
}

function draw() {
    background("#b042f4");
    print(`X is ${mouseX}, Y is ${mouseY}!`);
}
```

Chapter V

Variable Scope and Game State

If the following code, where a variable is created inside of one function and used inside a different function, you will see an error in the console called "Uncaught ReferenceError."

```
function setup() {
    createCanvas(400, 400);
    var points = 0;
}

function mouseClicked() {
    points = points + 1;
}

function draw() {
    background(220);
}
```

The reason you see an error is because of `variable scope.` whenever you declare a new variable (var x = ...) inside of a function or a block of curly braces, the variable will only exist inside that section of the code.

If you want certain variables to stay relevant for the entire duration of your program, you can make them `global variables` by declaring them outside of the functions.

```
var points;

function setup() {
    createCanvas(400, 400);
    points = 0;
}

function mouseClicked() {
    points = points + 1;
}

function draw() {
    background(220);
}
```

You will find many uses for these type global variables when you start building games. One example is the concept of a "game state" variable which keeps track of the player's progress in a game. Game state could be a Boolean (var finished = false), or a number or string with many possible values (such as a var called gamestage which could be equal to "menu", "playing", "victory", or "lost").

Chapter VI

Assignments

VI.1 Detect a Mouse Click

Write a Processing sketch where, once you click anywhere on the canvas, the background changes color.

VI.2 Detect a Click on a Square Button

Write a Processing sketch which shows a square button of a different color than the background. If the player clicks the button, the background changes color. If they click outside the button, the button changes color. Neither the button nor the background need to change back when you click again.

VI.3 Use a Boolean state variable

With the help of a true/false state variable, write a Processing sketch where repeatedly clicking on a square button causes the background color to toggle back and forth between two different colors. Think of it this way: the square button is a lamp, and if it's "on", the background is yellow; if it is "off", the background is black.

VI.4 Create a Help Text Button

Create a Processing sketch which has a rectangular button labelled "Help!". When the player clicks the Help! button, some more text with pro-tips for life on the canvas. The help text should have a border color and/or a background color. If they player clicks "Help!" again, the helptext goes away.

VI.5 Use a numeric state variable

Create a Processing sketch which displays two buttons, one red and one blue. There should be two global variables called score_blue and score_red. The score variables keep track of how many times the player has clicked the corresponding buttons.

VI.6 Draw a Continuous Line

Go beyond the explanations in this PDF and examine the p5js documentation for other mouse controls which could allow you to draw a line wherever the mouse goes on your canvas. Bonus: Only draw while the while the mouse button is held down, and stop drawing when the mouse is released.

VI.7 Bonus: Mouse Click on a Circular Button

Redo the red vs blue project, but make each button circular. Points can only be awarded if the player clicks inside the colored circle.

Chapter VII

SuperBonus!

Done all of this before?

Do all the assignments above, but do them by creating your own `Button` class which has color, coordinates, shape, and an onClick() function (plus any other attributes that you think of which would make it more usable.)