Subject: Python Tic Tac Toe: Learn Game Development by Building a Powerful Python Game [PT-2]
Posted by admin on Thu, 30 Jul 2026 08:14:42 GMT
View Forum Message <> Reply to Message

Continuation of the Tic-Tac-Toe Article - Part 2: Implementing the Game Logic and AI Move In the previous section, we set up the user interface for the Tic-Tac-Toe game using Kivy. Now, let's dive into the implementation of the game logic and the AI player's move.
Step 4: Handling Button Clicks and Game Logic

```
def on_button_click(self, button):
    row, col = self.get_button_position(button)

    if self.board[row][col] == '':
        self.board[row][col] = self.current_player
        button.text = self.current_player

        if self.check_win(self.current_player):
            self.display_winner(self.current_player)
            self.reset_game()
        elif self.check_draw():
            self.display_draw()
            self.reset_game()
        else:
            self.switch_player()
            Clock.schedule_once(lambda dt: self.ai_move(), 1)  # Delay before AI
move

def get_button_position(self, button):
    for row, row_buttons in enumerate(self.buttons):
        for col, btn in enumerate(row_buttons):
            if btn == button:
                return row, col

def switch_player(self):
    self.current_player = 'O' if self.current_player == 'X' else 'X'

def check_win(self, player):
    # Check rows
    for row in self.board:
        if row[0] == row[1] == row[2] == player:
            return True

    # Check columns
    for col in range(3):
        if self.board[0][col] == self.board[1][col] == self.board[2][col] == player:
            return True
```

```
    # Check diagonals
    if self.board[0][0] == self.board[1][1] == self.board[2][2] == player:
        return True
    if self.board[0][2] == self.board[1][1] == self.board[2][0] == player:
        return True

    return False

def check_draw(self):
    return all(all(cell != '' for cell in row) for row in self.board)

def display_winner(self, player):
    self.popup = Popup(title="Game Over", size_hint=(0.6, 0.4))
    content = Label(text=f"Player {player} wins!")
    self.popup.content = content
    self.popup.open()

def display_draw(self):
    self.popup = Popup(title="Game Over", size_hint=(0.6, 0.4))
    content = Label(text="It's a draw!")
    self.popup.content = content
    self.popup.open()

def reset_game(self):
    Clock.schedule_once(self.dismiss_popup, 2)
    self.current_player = 'X'
    self.board = [['', '', ''] for _ in range(3)]
    for row in self.buttons:
        for button in row:
            button.text = ''

def dismiss_popup(self, dt):
    if self.popup:
        self.popup.dismiss()
        self.popup = None
```

Now we define several methods in the TicTacToeGame class to handle button clicks and implement the game logic. Let's go through each method:

on_button_click(self, button): This method is called when a button is clicked. It retrieves the row and column position of the clicked button using the get_button_position method. If the clicked cell is empty, the current player's symbol is placed on the board, and the button's text is updated accordingly. Then, it checks if the current player has won the game using the check_win method. If a win is detected, the display_winner method is called, and the game is reset. If the game is a draw, the display_draw method is called, and the game is reset. Otherwise, it switches the player's turn and schedules the AI's move using the ai_move method after a delay of 1 second.

get_button_position(self, button):* This method is used to get the row and column position of a

button in the grid layout. It iterates over the self.buttons list and returns the row and column indices when a matching button is found.

switch_player(self): This method switches the current player between 'X' and 'O'.

check_win(self, player): This method checks if the specified player has won the game. It checks for winning combinations in rows, columns, and diagonals. If a winning combination is found, it returns True; otherwise, it returns False.

check_draw(self): This method checks if the game is a draw by ensuring that all cells on the board are non-empty. If all cells are occupied, it returns True; otherwise, it returns False.

display_winner(self, player): This method displays a popup message indicating the winner of the game. It creates a Popup widget with a title and a Label as its content, showing the winning player's symbol.

display_draw(self): This method displays a popup message indicating a draw in the game. It creates a Popup widget with a title and a Label as its content, showing the draw message.

reset_game(self): This method resets the game after it ends. It schedules the dismissal of the popup message after a delay of 2 seconds, resets the current player to 'X', clears the game board, and resets the button texts.

dismiss_popup(self, dt): This method dismisses the current popup message if it exists.
Step 5: Implementing the AI Move

```
def ai_move(self):
    if self.check_draw() or self.check_win(self.current_player):
        return

    empty_cells = []
    for row in range(3):
        for col in range(3):
            if self.board[row][col] == '':
                empty_cells.append((row, col))

    row, col = choice(empty_cells)
    self.board[row][col] = self.ai_player
    button = self.buttons[row][col]
    button.text = self.ai_player

    if self.check_win(self.ai_player):
        self.display_winner(self.ai_player)
        self.reset_game()
    elif self.check_draw():
        self.display_draw()
        self.reset_game()
    else:
```

```
        self.switch_player()
```

In this step, we implement the ai_move method in the TicTacToeGame class to make the AI player's move. Let's go through the code:

ai_move(self): This method is responsible for the AI player's move. It first checks if the game is already a draw or if the current player has won. If either condition is true, it returns without making a move.

The AI player generates a list of empty cells on the game board. It iterates over each row and column and adds the coordinates of empty cells to the empty_cells list.

Using the choice function from the random module, the AI player randomly selects a cell from the empty_cells list.

The selected cell's row and column indices are used to update the game board with the AI player's symbol (`self.ai_player`). The corresponding button's text is also updated.

After making the move, it checks if the AI player has won the game or if it's a draw. If the AI player wins, the display_winner method is called, and the game is reset. If it's a draw, the display_draw method is called, and the game is reset. Otherwise, it switches the player's turn.
Step 6: Building the App and Running the Game

```
class TicTacToeApp(App):
    def build(self):
        return TicTacToeGame()

if __name__ == '__main__':
    TicTacToeApp().run()
```

In this final step, we define the TicTacToeApp class that represents the Kivy application. Let's understand the code:

TicTacToeApp(App): This class inherits from the App class provided by Kivy and represents the main application.

build(self): This method is overridden to define the root widget of the application. Here, we create an instance of the TicTacToeGame class and return it as the root widget.

if __name__ == '__main__':: This condition checks if the script is being run directly and not imported as a module.

TicTacToeApp().run(): This line creates an instance of the TicTacToeApp class and runs the Kivy application. The run() method starts the application's main loop, which handles events, updates the UI, and responds to user interactions.

When you execute this script, the Kivy application is launched, and you can play the Tic-Tac-Toe game by interacting with the UI. The game board is displayed, and you can click on the buttons to

make moves. The AI opponent automatically makes its moves after a delay of 1 second.
Full Code:

```
import kivy
from kivy.app import App
from kivy.uix.gridlayout import GridLayout
from kivy.uix.button import Button
from kivy.uix.popup import Popup
from kivy.uix.label import Label
from kivy.clock import Clock
from random import choice

kivy.require('2.0.0')

class TicTacToeGame(GridLayout):
    def __init__(self, **kwargs):
        super(TicTacToeGame, self).__init__(**kwargs)
        self.cols = 3
        self.buttons = []
        self.current_player = 'X'
        self.board = [['', '', ''] for _ in range(3)]
        self.popup = None

        for row in range(3):
            row_buttons = []
            for col in range(3):
                button = Button(font_size=80)
                button.bind(on_press=self.on_button_click)
                self.add_widget(button)
                row_buttons.append(button)
            self.buttons.append(row_buttons)

        # AI opponent
        self.ai_player = 'O'

    def on_button_click(self, button):
        row, col = self.get_button_position(button)

        if self.board[row][col] == '':
            self.board[row][col] = self.current_player
            button.text = self.current_player

            if self.check_win(self.current_player):
                self.display_winner(self.current_player)
                self.reset_game()
            elif self.check_draw():
                self.display_draw()
                self.reset_game()
```

```
            else:
                self.switch_player()
                Clock.schedule_once(lambda dt: self.ai_move(), 1)  # Delay
before AI move

    def get_button_position(self, button):
        for row, row_buttons in enumerate(self.buttons):
            for col, btn in enumerate(row_buttons):
                if btn == button:
                    return row, col

    def switch_player(self):
        self.current_player = 'O' if self.current_player == 'X' else 'X'

    def check_win(self, player):
        for row in self.board:
            if row[0] == row[1] == row[2] == player:
                return True

        for col in range(3):
            if self.board[0][col] == self.board[1][col] == self.board[2][col] == player:
                return True

        if self.board[0][0] == self.board[1][1] == self.board[2][2] == player:
            return True

        if self.board[0][2] == self.board[1][1] == self.board[2][0] == player:
            return True

        return False

    def check_draw(self):
        return all(all(cell != '' for cell in row) for row in self.board)

    def display_winner(self, player):
        self.popup = Popup(title="Game Over", size_hint=(0.6, 0.4))
        content = Label(text=f"Player {player} wins!")
        self.popup.content = content
        self.popup.open()

    def display_draw(self):
        self.popup = Popup(title="Game Over", size_hint=(0.6, 0.4))
        content = Label(text="It's a draw!")
        self.popup.content = content
        self.popup.open()

    def reset_game(self):
        Clock.schedule_once(self.dismiss_popup, 2)
```

```
        self.current_player = 'X'
        self.board = [['', '', ''] for _ in range(3)]
        for row in self.buttons:
            for button in row:
                button.text = ''

    def dismiss_popup(self, dt):
        if self.popup:
            self.popup.dismiss()
            self.popup = None

    def ai_move(self):
        if self.check_draw() or self.check_win(self.current_player):
            return

        empty_cells = []
        for row in range(3):
            for col in range(3):
                if self.board[row][col] == '':
                    empty_cells.append((row, col))

        row, col = choice(empty_cells)
        self.board[row][col] = self.ai_player
        button = self.buttons[row][col]
        button.text = self.ai_player

        if self.check_win(self.ai_player):
            self.display_winner(self.ai_player)
            self.reset_game()
        elif self.check_draw():
            self.display_draw()
            self.reset_game()
        else:
            self.switch_player()

class TicTacToeApp(App):
    def build(self):
        return TicTacToeGame()

if __name__ == '__main__':
    TicTacToeApp().run()
```