import time

import logging

import sys

import os

import traceback

import numpy as np

import pandas as pd

from datetime import datetime, timedelta

import threading

import tkinter as tk

from tkinter import ttk, messagebox, filedialog

import matplotlib.pyplot as plt

from matplotlib.figure import Figure

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

from iqoptionapi.stable\_api import IQ\_Option

# Configurar logging

logging.basicConfig(level=logging.INFO,

format='%(asctime)s - %(levelname)s - %(message)s',

handlers=[

logging.FileHandler("bot\_log.txt", mode='a'),

logging.StreamHandler(sys.stdout)

])

logger = logging.getLogger()

logger.setLevel(logging.DEBUG)

# Credenciais (serão solicitadas ao usuário)

EMAIL = "diogorodrigobelo@hotmail.com"

PASSWORD = "Maritaca36"

class TradingBot:

def \_\_init\_\_(self, master=None):

self.master = master

self.api = None

self.connected = False

self.is\_trading = False

self.trading\_thread = None

self.update\_gui\_after\_id = None

# Controle de operações

self.active\_trades = {}

self.trade\_history = []

# Variáveis de controle

self.account\_type = tk.StringVar(value="PRACTICE") # Demo ou Real

self.operation\_mode = tk.StringVar(value="ULTRA RÁPIDO")

self.risk\_mode = tk.StringVar(value="AGRESSIVO")

# Configuração do modo de operação

self.operation\_settings = {

"ULTRA RÁPIDO": {

"expiration": 1, # Expiração em minutos

"timeframe": 60, # Timeframe em segundos

"signal\_threshold": 75, # Limiar para entrada

"indicators": ["RSI", "MACD", "BOLLINGER", "STOCHASTIC"],

"retracement\_level": 0.25 # Nível de retração

},

"NORMAL": {

"expiration": 5, # Expiração em minutos

"timeframe": 300, # Timeframe em segundos

"signal\_threshold": 65, # Limiar para entrada

"indicators": ["RSI", "MACD", "BOLLINGER"],

"retracement\_level": 0.5 # Nível de retração

},

"SEGURO": {

"expiration": 15, # Expiração em minutos

"timeframe": 900, # Timeframe em segundos

"signal\_threshold": 80, # Limiar para entrada

"indicators": ["RSI", "MACD", "BOLLINGER", "ADX"],

"retracement\_level": 0.75 # Nível de retração

}

}

# Configuração do gerenciamento de risco

self.risk\_settings = {

"AGRESSIVO": {

"base\_amount": 10, # Valor base

"martingale\_factor": 2.5, # Fator de martingale

"max\_martingale": 2, # Máximo de martingales

"max\_daily\_loss": 50, # Perda máxima diária em %

"daily\_goal": 30, # Meta diária em %

"max\_open\_trades": 4 # Máximo de operações simultâneas

},

"MODERADO": {

"base\_amount": 5,

"martingale\_factor": 2.0,

"max\_martingale": 1,

"max\_daily\_loss": 25,

"daily\_goal": 15,

"max\_open\_trades": 2

},

"CONSERVADOR": {

"base\_amount": 2,

"martingale\_factor": 0, # Sem martingale

"max\_martingale": 0,

"max\_daily\_loss": 10,

"daily\_goal": 5,

"max\_open\_trades": 1

}

}

# Pares disponíveis e selecionados (focando em pares OTC que estão disponíveis 24h)

self.available\_pairs = ["EURUSD-OTC", "EURGBP-OTC", "GBPUSD-OTC", "USDCHF-OTC", "EURJPY-OTC", "NZDUSD-OTC", "AUDCAD-OTC", "USDJPY-OTC"]

self.selected\_pairs = ["EURUSD-OTC", "GBPUSD-OTC", "USDJPY-OTC", "EURGBP-OTC"] # Padrão

# Estatísticas

self.stats = {

"total\_trades": 0,

"wins": 0,

"losses": 0,

"win\_rate": 0,

"profit": 0,

"start\_balance": 0,

"current\_balance": 0,

"session\_start": datetime.now()

}

if master:

self.create\_gui()

def create\_gui(self):

"""Criar interface gráfica"""

self.master.title("IQ Option Bot - Trading em Tempo Real")

self.master.geometry("1200x800")

self.master.minsize(1000, 700)

# Estilo

self.colors = {

"bg": "#121212",

"fg": "#f0f0f0",

"accent": "#4CAF50",

"accent\_dark": "#388E3C",

"warning": "#FF9800",

"danger": "#F44336",

"info": "#2196F3",

"card": "#1E1E1E",

"border": "#333333"

}

style = ttk.Style()

style.theme\_use("clam") # Base theme

# Configure styles

style.configure("TFrame", background=self.colors["bg"])

style.configure("Card.TFrame", background=self.colors["card"],

borderwidth=1, relief="solid")

style.configure("TLabel", background=self.colors["bg"],

foreground=self.colors["fg"], font=("Segoe UI", 10))

style.configure("Header.TLabel", font=("Segoe UI", 12, "bold"))

style.configure("Title.TLabel", font=("Segoe UI", 14, "bold"))

style.configure("TButton", background=self.colors["accent"],

foreground=self.colors["fg"], font=("Segoe UI", 10))

style.map("TButton", background=[("active", self.colors["accent\_dark"])])

style.configure("Danger.TButton", background=self.colors["danger"])

style.map("Danger.TButton", background=[("active", "#D32F2F")])

style.configure("Secondary.TButton", background="#424242")

style.map("Secondary.TButton", background=[("active", "#616161")])

# Criar notebook principal

self.notebook = ttk.Notebook(self.master)

self.notebook.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

# Abas

self.tab\_trading = ttk.Frame(self.notebook, style="TFrame")

self.tab\_charts = ttk.Frame(self.notebook, style="TFrame")

self.tab\_history = ttk.Frame(self.notebook, style="TFrame")

self.tab\_settings = ttk.Frame(self.notebook, style="TFrame")

self.notebook.add(self.tab\_trading, text="Trading")

self.notebook.add(self.tab\_charts, text="Gráficos")

self.notebook.add(self.tab\_history, text="Histórico")

self.notebook.add(self.tab\_settings, text="Configurações")

# Configurar abas

self.setup\_trading\_tab()

self.setup\_charts\_tab()

self.setup\_history\_tab()

self.setup\_settings\_tab()

# Barra de status

self.status\_bar = tk.Frame(self.master, bg=self.colors["bg"], height=30)

self.status\_bar.pack(side=tk.BOTTOM, fill=tk.X)

self.status\_label = tk.Label(self.status\_bar, text="Desconectado",

bg=self.colors["bg"], fg=self.colors["warning"])

self.status\_label.pack(side=tk.LEFT, padx=10)

self.connection\_label = tk.Label(self.status\_bar, text="●",

bg=self.colors["bg"], fg=self.colors["danger"])

self.connection\_label.pack(side=tk.LEFT, padx=5)

self.balance\_label = tk.Label(self.status\_bar, text="Saldo: $0.00",

bg=self.colors["bg"], fg=self.colors["info"])

self.balance\_label.pack(side=tk.RIGHT, padx=10)

# Evento de fechamento

self.master.protocol("WM\_DELETE\_WINDOW", self.on\_close)

def setup\_trading\_tab(self):

"""Configurar aba de trading"""

# Frame superior para controles

control\_frame = ttk.Frame(self.tab\_trading, style="TFrame")

control\_frame.pack(fill=tk.X, padx=10, pady=10)

# Frame de conexão

connect\_frame = ttk.Frame(control\_frame, style="Card.TFrame")

connect\_frame.pack(side=tk.LEFT, padx=5, pady=5, fill=tk.Y)

ttk.Label(connect\_frame, text="Login IQ Option", style="Header.TLabel").pack(padx=10, pady=5)

# Credenciais

cred\_frame = ttk.Frame(connect\_frame)

cred\_frame.pack(padx=10, pady=5, fill=tk.X)

ttk.Label(cred\_frame, text="Email:").grid(row=0, column=0, padx=5, pady=5, sticky=tk.W)

self.email\_var = tk.StringVar(value=EMAIL)

email\_entry = ttk.Entry(cred\_frame, textvariable=self.email\_var, width=30)

email\_entry.grid(row=0, column=1, padx=5, pady=5, sticky=tk.W)

ttk.Label(cred\_frame, text="Senha:").grid(row=1, column=0, padx=5, pady=5, sticky=tk.W)

self.password\_var = tk.StringVar(value=PASSWORD)

password\_entry = ttk.Entry(cred\_frame, textvariable=self.password\_var, show="\*", width=30)

password\_entry.grid(row=1, column=1, padx=5, pady=5, sticky=tk.W)

# Tipo de conta

account\_frame = ttk.Frame(connect\_frame)

account\_frame.pack(padx=10, pady=5, fill=tk.X)

ttk.Label(account\_frame, text="Tipo de Conta:").grid(row=0, column=0, padx=5, pady=5, sticky=tk.W)

ttk.Radiobutton(account\_frame, text="Demo", variable=self.account\_type, value="PRACTICE").grid(row=0, column=1, padx=5, pady=5)

ttk.Radiobutton(account\_frame, text="Real", variable=self.account\_type, value="REAL").grid(row=0, column=2, padx=5, pady=5)

# Botões de conexão

button\_frame = ttk.Frame(connect\_frame)

button\_frame.pack(padx=10, pady=10, fill=tk.X)

self.connect\_button = ttk.Button(button\_frame, text="Conectar", command=self.connect)

self.connect\_button.pack(side=tk.LEFT, padx=5, fill=tk.X, expand=True)

self.disconnect\_button = ttk.Button(button\_frame, text="Desconectar", command=self.disconnect, state=tk.DISABLED)

self.disconnect\_button.pack(side=tk.LEFT, padx=5, fill=tk.X, expand=True)

# Frame de ações de trading

actions\_frame = ttk.Frame(control\_frame, style="Card.TFrame")

actions\_frame.pack(side=tk.RIGHT, padx=5, pady=5, fill=tk.Y)

ttk.Label(actions\_frame, text="Controle de Trading", style="Header.TLabel").pack(padx=10, pady=5)

# Modo de operação

mode\_frame = ttk.Frame(actions\_frame)

mode\_frame.pack(padx=10, pady=5, fill=tk.X)

ttk.Label(mode\_frame, text="Modo:").grid(row=0, column=0, padx=5, pady=5, sticky=tk.W)

mode\_menu = ttk.OptionMenu(mode\_frame, self.operation\_mode, "ULTRA RÁPIDO", "ULTRA RÁPIDO", "NORMAL", "SEGURO")

mode\_menu.grid(row=0, column=1, padx=5, pady=5, sticky=tk.W)

ttk.Label(mode\_frame, text="Risco:").grid(row=1, column=0, padx=5, pady=5, sticky=tk.W)

risk\_menu = ttk.OptionMenu(mode\_frame, self.risk\_mode, "AGRESSIVO", "AGRESSIVO", "MODERADO", "CONSERVADOR")

risk\_menu.grid(row=1, column=1, padx=5, pady=5, sticky=tk.W)

# Botões de trading

trade\_button\_frame = ttk.Frame(actions\_frame)

trade\_button\_frame.pack(padx=10, pady=10, fill=tk.X)

self.start\_button = ttk.Button(trade\_button\_frame, text="Iniciar Trading",

command=self.start\_trading, state=tk.DISABLED)

self.start\_button.pack(side=tk.LEFT, padx=5, fill=tk.X, expand=True)

self.stop\_button = ttk.Button(trade\_button\_frame, text="Parar",

command=self.stop\_trading, state=tk.DISABLED, style="Danger.TButton")

self.stop\_button.pack(side=tk.LEFT, padx=5, fill=tk.X, expand=True)

# Frame central para estatísticas

stats\_frame = ttk.Frame(self.tab\_trading, style="Card.TFrame")

stats\_frame.pack(fill=tk.X, padx=10, pady=10)

ttk.Label(stats\_frame, text="Estatísticas de Trading", style="Header.TLabel").pack(anchor=tk.W, padx=10, pady=5)

# Grid de estatísticas

grid = ttk.Frame(stats\_frame)

grid.pack(padx=10, pady=5, fill=tk.X)

# Linha 1

ttk.Label(grid, text="Total Operações:").grid(row=0, column=0, padx=10, pady=5, sticky=tk.W)

self.total\_trades\_label = ttk.Label(grid, text="0")

self.total\_trades\_label.grid(row=0, column=1, padx=5, pady=5, sticky=tk.W)

ttk.Label(grid, text="Vitórias:").grid(row=0, column=2, padx=20, pady=5, sticky=tk.W)

self.wins\_label = ttk.Label(grid, text="0")

self.wins\_label.grid(row=0, column=3, padx=5, pady=5, sticky=tk.W)

ttk.Label(grid, text="Derrotas:").grid(row=0, column=4, padx=20, pady=5, sticky=tk.W)

self.losses\_label = ttk.Label(grid, text="0")

self.losses\_label.grid(row=0, column=5, padx=5, pady=5, sticky=tk.W)

ttk.Label(grid, text="Taxa de Acerto:").grid(row=0, column=6, padx=20, pady=5, sticky=tk.W)

self.win\_rate\_label = ttk.Label(grid, text="0%")

self.win\_rate\_label.grid(row=0, column=7, padx=5, pady=5, sticky=tk.W)

# Linha 2

ttk.Label(grid, text="Lucro:").grid(row=1, column=0, padx=10, pady=5, sticky=tk.W)

self.profit\_label = ttk.Label(grid, text="$0.00")

self.profit\_label.grid(row=1, column=1, padx=5, pady=5, sticky=tk.W)

ttk.Label(grid, text="Saldo Inicial:").grid(row=1, column=2, padx=20, pady=5, sticky=tk.W)

self.start\_balance\_label = ttk.Label(grid, text="$0.00")

self.start\_balance\_label.grid(row=1, column=3, padx=5, pady=5, sticky=tk.W)

ttk.Label(grid, text="Saldo Atual:").grid(row=1, column=4, padx=20, pady=5, sticky=tk.W)

self.current\_balance\_label = ttk.Label(grid, text="$0.00")

self.current\_balance\_label.grid(row=1, column=5, padx=5, pady=5, sticky=tk.W)

ttk.Label(grid, text="Operações Ativas:").grid(row=1, column=6, padx=20, pady=5, sticky=tk.W)

self.active\_trades\_label = ttk.Label(grid, text="0")

self.active\_trades\_label.grid(row=1, column=7, padx=5, pady=5, sticky=tk.W)

# Frame para operações ativas

active\_frame = ttk.Frame(self.tab\_trading, style="Card.TFrame")

active\_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

ttk.Label(active\_frame, text="Operações Ativas", style="Header.TLabel").pack(anchor=tk.W, padx=10, pady=5)

# Tabela de operações ativas

columns = ("id", "pair", "direction", "amount", "open\_time", "expiry", "martingale")

self.active\_tree = ttk.Treeview(active\_frame, columns=columns, show="headings", height=5)

# Configurar colunas

self.active\_tree.heading("id", text="ID")

self.active\_tree.heading("pair", text="Par")

self.active\_tree.heading("direction", text="Direção")

self.active\_tree.heading("amount", text="Valor")

self.active\_tree.heading("open\_time", text="Hora")

self.active\_tree.heading("expiry", text="Expiração")

self.active\_tree.heading("martingale", text="Martingale")

self.active\_tree.column("id", width=70)

self.active\_tree.column("pair", width=100)

self.active\_tree.column("direction", width=100)

self.active\_tree.column("amount", width=100)

self.active\_tree.column("open\_time", width=150)

self.active\_tree.column("expiry", width=100)

self.active\_tree.column("martingale", width=100)

# Adicionar scrollbar

scrollbar = ttk.Scrollbar(active\_frame, orient=tk.VERTICAL, command=self.active\_tree.yview)

self.active\_tree.configure(yscrollcommand=scrollbar.set)

self.active\_tree.pack(side=tk.LEFT, fill=tk.BOTH, expand=True, padx=10, pady=5)

scrollbar.pack(side=tk.RIGHT, fill=tk.Y, padx=(0, 10), pady=5)

# Frame para log

log\_frame = ttk.Frame(self.tab\_trading, style="Card.TFrame")

log\_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

ttk.Label(log\_frame, text="Log de Operações", style="Header.TLabel").pack(anchor=tk.W, padx=10, pady=5)

# Widget de texto para log

self.log\_text = tk.Text(log\_frame, height=10, width=50, bg="#212121", fg="#f0f0f0", wrap=tk.WORD)

self.log\_text.pack(side=tk.LEFT, fill=tk.BOTH, expand=True, padx=10, pady=5)

log\_scrollbar = ttk.Scrollbar(log\_frame, orient=tk.VERTICAL, command=self.log\_text.yview)

self.log\_text.configure(yscrollcommand=log\_scrollbar.set)

log\_scrollbar.pack(side=tk.RIGHT, fill=tk.Y, padx=(0, 10), pady=5)

# Configurar tags para colorir o log

self.log\_text.tag\_configure("info", foreground="#2196F3")

self.log\_text.tag\_configure("success", foreground="#4CAF50")

self.log\_text.tag\_configure("warning", foreground="#FF9800")

self.log\_text.tag\_configure("error", foreground="#F44336")

def setup\_charts\_tab(self):

"""Configurar aba de gráficos"""

control\_frame = ttk.Frame(self.tab\_charts, style="TFrame")

control\_frame.pack(fill=tk.X, padx=10, pady=10)

# Seleção de par

ttk.Label(control\_frame, text="Par:").pack(side=tk.LEFT, padx=5, pady=5)

self.chart\_pair\_var = tk.StringVar(value="EURUSD")

pair\_menu = ttk.OptionMenu(control\_frame, self.chart\_pair\_var, "EURUSD", \*self.available\_pairs)

pair\_menu.pack(side=tk.LEFT, padx=5, pady=5)

# Timeframe

ttk.Label(control\_frame, text="Timeframe:").pack(side=tk.LEFT, padx=15, pady=5)

self.chart\_timeframe\_var = tk.StringVar(value="1m")

timeframe\_menu = ttk.OptionMenu(control\_frame, self.chart\_timeframe\_var, "1m", "1m", "5m", "15m", "1h")

timeframe\_menu.pack(side=tk.LEFT, padx=5, pady=5)

# Botão atualizar

ttk.Button(control\_frame, text="Atualizar Gráfico", command=self.update\_chart).pack(side=tk.LEFT, padx=15, pady=5)

# Canvas para o gráfico

chart\_frame = ttk.Frame(self.tab\_charts, style="Card.TFrame")

chart\_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

self.figure = Figure(figsize=(5, 4), dpi=100, facecolor=self.colors["card"])

self.chart\_canvas = FigureCanvasTkAgg(self.figure, master=chart\_frame)

self.chart\_canvas.get\_tk\_widget().pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

# Adicionar subplots

self.price\_subplot = self.figure.add\_subplot(311) # Preço principal

self.indicator1\_subplot = self.figure.add\_subplot(312) # RSI

self.indicator2\_subplot = self.figure.add\_subplot(313) # MACD

def setup\_history\_tab(self):

"""Configurar aba de histórico"""

control\_frame = ttk.Frame(self.tab\_history, style="TFrame")

control\_frame.pack(fill=tk.X, padx=10, pady=10)

# Filtros

ttk.Label(control\_frame, text="Filtrar por:").pack(side=tk.LEFT, padx=5, pady=5)

self.filter\_result\_var = tk.StringVar(value="TODOS")

filter\_menu = ttk.OptionMenu(control\_frame, self.filter\_result\_var, "TODOS", "TODOS", "VITÓRIAS", "DERROTAS")

filter\_menu.pack(side=tk.LEFT, padx=5, pady=5)

# Botão exportar

ttk.Button(control\_frame, text="Exportar CSV", command=self.export\_history).pack(side=tk.RIGHT, padx=10, pady=5)

# Tabela de histórico

history\_frame = ttk.Frame(self.tab\_history, style="Card.TFrame")

history\_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

columns = ("id", "pair", "direction", "amount", "result", "profit", "open\_time", "close\_time", "martingale")

self.history\_tree = ttk.Treeview(history\_frame, columns=columns, show="headings")

# Configurar colunas

self.history\_tree.heading("id", text="ID")

self.history\_tree.heading("pair", text="Par")

self.history\_tree.heading("direction", text="Direção")

self.history\_tree.heading("amount", text="Valor")

self.history\_tree.heading("result", text="Resultado")

self.history\_tree.heading("profit", text="Lucro")

self.history\_tree.heading("open\_time", text="Abertura")

self.history\_tree.heading("close\_time", text="Fechamento")

self.history\_tree.heading("martingale", text="Martingale")

self.history\_tree.column("id", width=70)

self.history\_tree.column("pair", width=100)

self.history\_tree.column("direction", width=100)

self.history\_tree.column("amount", width=100)

self.history\_tree.column("result", width=100)

self.history\_tree.column("profit", width=100)

self.history\_tree.column("open\_time", width=150)

self.history\_tree.column("close\_time", width=150)

self.history\_tree.column("martingale", width=100)

# Adicionar scrollbar

scrollbar = ttk.Scrollbar(history\_frame, orient=tk.VERTICAL, command=self.history\_tree.yview)

self.history\_tree.configure(yscrollcommand=scrollbar.set)

self.history\_tree.pack(side=tk.LEFT, fill=tk.BOTH, expand=True, padx=10, pady=10)

scrollbar.pack(side=tk.RIGHT, fill=tk.Y, padx=(0, 10), pady=10)

# Configurar tags para colorir o histórico

self.history\_tree.tag\_configure("win", background="#1e392a")

self.history\_tree.tag\_configure("loss", background="#3c1e1e")

def setup\_settings\_tab(self):

"""Configurar aba de configurações"""

# Frame para configuração de operação

operation\_frame = ttk.Frame(self.tab\_settings, style="Card.TFrame")

operation\_frame.pack(fill=tk.X, padx=10, pady=10)

ttk.Label(operation\_frame, text="Configuração de Operação", style="Header.TLabel").pack(anchor=tk.W, padx=10, pady=5)

# Timeframe e expiração

settings\_frame = ttk.Frame(operation\_frame)

settings\_frame.pack(padx=10, pady=5, fill=tk.X)

ttk.Label(settings\_frame, text="Expiração:").grid(row=0, column=0, padx=5, pady=5, sticky=tk.W)

self.expiration\_var = tk.StringVar(value="1")

ttk.Spinbox(settings\_frame, from\_=1, to=60, textvariable=self.expiration\_var, width=5).grid(row=0, column=1, padx=5, pady=5, sticky=tk.W)

ttk.Label(settings\_frame, text="minuto(s)").grid(row=0, column=2, padx=5, pady=5, sticky=tk.W)

ttk.Label(settings\_frame, text="Limiar de Sinal:").grid(row=0, column=3, padx=20, pady=5, sticky=tk.W)

self.threshold\_var = tk.StringVar(value="75")

ttk.Spinbox(settings\_frame, from\_=50, to=95, textvariable=self.threshold\_var, width=5).grid(row=0, column=4, padx=5, pady=5, sticky=tk.W)

ttk.Label(settings\_frame, text="%").grid(row=0, column=5, padx=5, pady=5, sticky=tk.W)

# Indicadores

indicators\_frame = ttk.Frame(operation\_frame)

indicators\_frame.pack(padx=10, pady=5, fill=tk.X)

ttk.Label(indicators\_frame, text="Indicadores Ativos:").grid(row=0, column=0, padx=5, pady=5, sticky=tk.W)

self.use\_rsi = tk.BooleanVar(value=True)

self.use\_macd = tk.BooleanVar(value=True)

self.use\_bollinger = tk.BooleanVar(value=True)

self.use\_stochastic = tk.BooleanVar(value=True)

ttk.Checkbutton(indicators\_frame, text="RSI", variable=self.use\_rsi).grid(row=0, column=1, padx=10, pady=5)

ttk.Checkbutton(indicators\_frame, text="MACD", variable=self.use\_macd).grid(row=0, column=2, padx=10, pady=5)

ttk.Checkbutton(indicators\_frame, text="Bollinger", variable=self.use\_bollinger).grid(row=0, column=3, padx=10, pady=5)

ttk.Checkbutton(indicators\_frame, text="Estocástico", variable=self.use\_stochastic).grid(row=0, column=4, padx=10, pady=5)

# Frame para gerenciamento de risco

risk\_frame = ttk.Frame(self.tab\_settings, style="Card.TFrame")

risk\_frame.pack(fill=tk.X, padx=10, pady=10)

ttk.Label(risk\_frame, text="Gerenciamento de Risco", style="Header.TLabel").pack(anchor=tk.W, padx=10, pady=5)

# Configurações de valor

value\_frame = ttk.Frame(risk\_frame)

value\_frame.pack(padx=10, pady=5, fill=tk.X)

ttk.Label(value\_frame, text="Valor Base:").grid(row=0, column=0, padx=5, pady=5, sticky=tk.W)

self.base\_amount\_var = tk.StringVar(value="10")

ttk.Spinbox(value\_frame, from\_=1, to=100, textvariable=self.base\_amount\_var, width=5).grid(row=0, column=1, padx=5, pady=5, sticky=tk.W)

ttk.Label(value\_frame, text="$").grid(row=0, column=2, padx=5, pady=5, sticky=tk.W)

ttk.Label(value\_frame, text="Perda Máxima:").grid(row=0, column=3, padx=20, pady=5, sticky=tk.W)

self.max\_loss\_var = tk.StringVar(value="50")

ttk.Spinbox(value\_frame, from\_=5, to=100, textvariable=self.max\_loss\_var, width=5).grid(row=0, column=4, padx=5, pady=5, sticky=tk.W)

ttk.Label(value\_frame, text="% do saldo").grid(row=0, column=5, padx=5, pady=5, sticky=tk.W)

ttk.Label(value\_frame, text="Meta Diária:").grid(row=0, column=6, padx=20, pady=5, sticky=tk.W)

self.daily\_goal\_var = tk.StringVar(value="20")

ttk.Spinbox(value\_frame, from\_=5, to=100, textvariable=self.daily\_goal\_var, width=5).grid(row=0, column=7, padx=5, pady=5, sticky=tk.W)

ttk.Label(value\_frame, text="% do saldo").grid(row=0, column=8, padx=5, pady=5, sticky=tk.W)

# Configurações de martingale

gale\_frame = ttk.Frame(risk\_frame)

gale\_frame.pack(padx=10, pady=5, fill=tk.X)

self.use\_martingale = tk.BooleanVar(value=True)

ttk.Checkbutton(gale\_frame, text="Usar Martingale", variable=self.use\_martingale).grid(row=0, column=0, padx=5, pady=5, sticky=tk.W)

ttk.Label(gale\_frame, text="Fator:").grid(row=0, column=1, padx=20, pady=5, sticky=tk.W)

self.martingale\_factor\_var = tk.StringVar(value="2.5")

ttk.Spinbox(gale\_frame, from\_=1.0, to=3.0, increment=0.1, textvariable=self.martingale\_factor\_var, width=5).grid(row=0, column=2, padx=5, pady=5, sticky=tk.W)

ttk.Label(gale\_frame, text="Níveis Máximos:").grid(row=0, column=3, padx=20, pady=5, sticky=tk.W)

self.max\_martingale\_var = tk.StringVar(value="2")

ttk.Spinbox(gale\_frame, from\_=0, to=5, textvariable=self.max\_martingale\_var, width=5).grid(row=0, column=4, padx=5, pady=5, sticky=tk.W)

ttk.Label(gale\_frame, text="Máximo Simultâneas:").grid(row=0, column=5, padx=20, pady=5, sticky=tk.W)

self.max\_trades\_var = tk.StringVar(value="4")

ttk.Spinbox(gale\_frame, from\_=1, to=10, textvariable=self.max\_trades\_var, width=5).grid(row=0, column=6, padx=5, pady=5, sticky=tk.W)

# Frame para pares de moedas

pairs\_frame = ttk.Frame(self.tab\_settings, style="Card.TFrame")

pairs\_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

ttk.Label(pairs\_frame, text="Pares de Moedas", style="Header.TLabel").pack(anchor=tk.W, padx=10, pady=5)

# Listbox para seleção de pares

self.pairs\_listbox = tk.Listbox(pairs\_frame, selectmode=tk.MULTIPLE, height=10)

self.pairs\_listbox.pack(side=tk.LEFT, fill=tk.BOTH, expand=True, padx=10, pady=10)

# Adicionar pares à lista

for pair in self.available\_pairs:

self.pairs\_listbox.insert(tk.END, pair)

if pair in self.selected\_pairs:

idx = self.available\_pairs.index(pair)

self.pairs\_listbox.selection\_set(idx)

# Botão aplicar

button\_frame = ttk.Frame(self.tab\_settings)

button\_frame.pack(fill=tk.X, padx=10, pady=10)

ttk.Button(button\_frame, text="Aplicar Configurações", command=self.apply\_settings).pack(side=tk.RIGHT, padx=10, pady=5)

def log\_message(self, message, level="info"):

"""Adicionar mensagem ao log"""

timestamp = datetime.now().strftime("%H:%M:%S")

log\_msg = f"[{timestamp}] {message}\n"

self.log\_text.insert(tk.END, log\_msg, level)

self.log\_text.see(tk.END)

# Também registrar no logger do sistema

if level == "info":

logger.info(message)

elif level == "success":

logger.info(message)

elif level == "warning":

logger.warning(message)

elif level == "error":

logger.error(message)

def connect(self):

"""Conectar à IQ Option API"""

email = self.email\_var.get()

password = self.password\_var.get()

account\_type = self.account\_type.get()

if not email or not password:

messagebox.showerror("Erro", "Email e senha são obrigatórios")

return

self.log\_message(f"Conectando à IQ Option com {email}...")

# Atualizar interface durante a conexão

self.status\_label.config(text="Conectando...", fg=self.colors["warning"])

self.master.update()

# Inicializar API

self.api = IQ\_Option(email, password)

# Tentar conectar

connected, reason = self.api.connect()

if connected:

# Verificar se ainda está conectado (double-check)

if self.api.check\_connect():

# Mudar para o tipo de conta selecionado

self.api.change\_balance(account\_type)

account\_mode = "DEMO" if account\_type == "PRACTICE" else "REAL"

# Obter saldo

balance = self.api.get\_balance()

self.stats["start\_balance"] = balance

self.stats["current\_balance"] = balance

# Atualizar interface

self.status\_label.config(text=f"Conectado ({account\_mode})", fg=self.colors["accent"])

self.connection\_label.config(fg=self.colors["accent"])

self.balance\_label.config(text=f"Saldo: ${balance:.2f}")

# Atualizar botões

self.connect\_button.config(state=tk.DISABLED)

self.disconnect\_button.config(state=tk.NORMAL)

self.start\_button.config(state=tk.NORMAL)

self.connected = True

self.log\_message(f"Conectado com sucesso à conta {account\_mode}", "success")

# Atualizar estatísticas

self.update\_stats\_display()

return True

else:

self.log\_message("Falha na verificação de conexão", "error")

self.status\_label.config(text="Desconectado", fg=self.colors["danger"])

return False

else:

self.log\_message(f"Falha na conexão: {reason}", "error")

self.status\_label.config(text="Desconectado", fg=self.colors["danger"])

messagebox.showerror("Erro de Conexão", f"Não foi possível conectar: {reason}")

return False

def disconnect(self):

"""Desconectar da API"""

if self.is\_trading:

if not messagebox.askyesno("Confirmação", "Trading em andamento. Interromper e desconectar?"):

return

self.stop\_trading()

if self.api:

# Fechar a conexão

self.api.disconnect()

self.connected = False

# Resetar estatísticas

self.reset\_stats()

# Atualizar interface

self.status\_label.config(text="Desconectado", fg=self.colors["warning"])

self.connection\_label.config(fg=self.colors["danger"])

self.balance\_label.config(text="Saldo: $0.00")

# Atualizar botões

self.connect\_button.config(state=tk.NORMAL)

self.disconnect\_button.config(state=tk.DISABLED)

self.start\_button.config(state=tk.DISABLED)

self.stop\_button.config(state=tk.DISABLED)

self.log\_message("Desconectado com sucesso", "warning")

def apply\_settings(self):

"""Aplicar configurações definidas pelo usuário"""

try:

# Obter pares selecionados

selected\_indices = self.pairs\_listbox.curselection()

self.selected\_pairs = [self.available\_pairs[i] for i in selected\_indices]

if not self.selected\_pairs:

messagebox.showwarning("Aviso", "Selecione pelo menos um par de moedas")

return

# Atualizar configurações

operation\_mode = self.operation\_mode.get()

risk\_mode = self.risk\_mode.get()

# Obter configurações atuais

current\_op = self.operation\_settings[operation\_mode]

current\_risk = self.risk\_settings[risk\_mode]

# Atualizar com valores da interface

current\_op["expiration"] = int(self.expiration\_var.get())

current\_op["signal\_threshold"] = int(self.threshold\_var.get())

current\_risk["base\_amount"] = float(self.base\_amount\_var.get())

current\_risk["max\_daily\_loss"] = int(self.max\_loss\_var.get())

current\_risk["daily\_goal"] = int(self.daily\_goal\_var.get())

current\_risk["martingale\_enabled"] = self.use\_martingale.get()

current\_risk["martingale\_factor"] = float(self.martingale\_factor\_var.get())

current\_risk["max\_martingale"] = int(self.max\_martingale\_var.get())

current\_risk["max\_open\_trades"] = int(self.max\_trades\_var.get())

# Atualizar lista de indicadores

indicators = []

if self.use\_rsi.get():

indicators.append("RSI")

if self.use\_macd.get():

indicators.append("MACD")

if self.use\_bollinger.get():

indicators.append("BOLLINGER")

if self.use\_stochastic.get():

indicators.append("STOCHASTIC")

if not indicators:

messagebox.showwarning("Aviso", "Selecione pelo menos um indicador")

return

current\_op["indicators"] = indicators

self.log\_message(f"Configurações aplicadas: Modo {operation\_mode}, Risco {risk\_mode}", "success")

self.log\_message(f"Pares selecionados: {', '.join(self.selected\_pairs)}", "info")

except Exception as e:

self.log\_message(f"Erro ao aplicar configurações: {str(e)}", "error")

messagebox.showerror("Erro", f"Erro ao aplicar configurações: {str(e)}")

def start\_trading(self):

"""Iniciar o processo de trading"""

if not self.connected or not self.api:

self.log\_message("Não conectado. Conecte-se primeiro.", "error")

return

# Verificar se há pares selecionados

if not self.selected\_pairs:

# Usar os pares padrão

self.selected\_pairs = self.available\_pairs[:4]

self.log\_message(f"Usando pares padrão: {', '.join(self.selected\_pairs)}", "warning")

# Confirmação especial para conta real

account\_type = self.account\_type.get()

if account\_type == "REAL":

confirmed = messagebox.askyesno("ATENÇÃO - CONTA REAL",

"⚠️ Você está prestes a operar com DINHEIRO REAL!\n\n"

"Tem certeza que deseja continuar?",

icon="warning")

if not confirmed:

self.log\_message("Operação em conta real cancelada pelo usuário", "warning")

return

# Segunda confirmação

confirmed2 = messagebox.askokcancel("CONFIRMAÇÃO FINAL - CONTA REAL",

"ÚLTIMA CHANCE: Você confirma que deseja operar com dinheiro real?\n\n"

"Digite 'CONFIRMAR' na caixa abaixo.",

initialvalue="")

if not confirmed2 or confirmed2.upper() != "CONFIRMAR":

self.log\_message("Operação em conta real não confirmada", "warning")

return

self.log\_message("Uso de conta REAL confirmado!", "warning")

# Obter modo de operação e risco

operation\_mode = self.operation\_mode.get()

risk\_mode = self.risk\_mode.get()

# Atualizar controles

self.start\_button.config(state=tk.DISABLED)

self.stop\_button.config(state=tk.NORMAL)

# Registrar início da sessão

self.stats["session\_start"] = datetime.now()

# Iniciar thread

self.is\_trading = True

self.trading\_thread = threading.Thread(target=self.trading\_loop)

self.trading\_thread.daemon = True

self.trading\_thread.start()

# Configurar atualização da GUI

self.schedule\_gui\_update()

self.log\_message(f"Trading iniciado: Modo {operation\_mode}, Risco {risk\_mode}", "success")

def stop\_trading(self):

"""Parar o processo de trading"""

self.is\_trading = False

# Aguardar thread de trading finalizar

if self.trading\_thread and self.trading\_thread.is\_alive():

self.log\_message("Aguardando conclusão das operações...", "warning")

self.trading\_thread.join(timeout=2.0)

# Cancelar atualização da GUI

if self.update\_gui\_after\_id:

self.master.after\_cancel(self.update\_gui\_after\_id)

self.update\_gui\_after\_id = None

# Atualizar controles

self.start\_button.config(state=tk.NORMAL)

self.stop\_button.config(state=tk.DISABLED)

self.log\_message("Trading interrompido pelo usuário", "warning")

def reset\_stats(self):

"""Resetar estatísticas"""

self.stats = {

"total\_trades": 0,

"wins": 0,

"losses": 0,

"win\_rate": 0,

"profit": 0,

"start\_balance": 0,

"current\_balance": 0,

"session\_start": datetime.now()

}

self.active\_trades = {}

self.update\_stats\_display()

# Limpar tabelas

for item in self.active\_tree.get\_children():

self.active\_tree.delete(item)

for item in self.history\_tree.get\_children():

self.history\_tree.delete(item)

def schedule\_gui\_update(self):

"""Agendar atualização da GUI"""

if self.is\_trading:

self.update\_gui()

self.update\_gui\_after\_id = self.master.after(1000, self.schedule\_gui\_update)

def update\_gui(self):

"""Atualizar elementos da GUI"""

if not self.connected or not self.api:

return

try:

# Verificar se ainda está conectado

if not self.api.check\_connect():

self.log\_message("Conexão perdida. Reconectando...", "error")

connected, \_ = self.api.connect()

if not connected:

self.log\_message("Falha ao reconectar", "error")

self.disconnect()

return

# Atualizar saldo

balance = self.api.get\_balance()

self.stats["current\_balance"] = balance

self.balance\_label.config(text=f"Saldo: ${balance:.2f}")

# Atualizar contagem de operações ativas

self.active\_trades\_label.config(text=str(len(self.active\_trades)))

# Calcular lucro da sessão

profit = balance - self.stats["start\_balance"]

self.stats["profit"] = profit

# Atualizar estatísticas

self.update\_stats\_display()

# Atualizar tabela de operações ativas

self.update\_active\_trades\_display()

except Exception as e:

self.log\_message(f"Erro ao atualizar GUI: {str(e)}", "error")

def update\_stats\_display(self):

"""Atualizar display de estatísticas"""

# Calcular taxa de acerto

total = self.stats["wins"] + self.stats["losses"]

if total > 0:

win\_rate = (self.stats["wins"] / total) \* 100

else:

win\_rate = 0

self.stats["win\_rate"] = win\_rate

# Atualizar labels

self.total\_trades\_label.config(text=str(self.stats["total\_trades"]))

self.wins\_label.config(text=str(self.stats["wins"]))

self.losses\_label.config(text=str(self.stats["losses"]))

self.win\_rate\_label.config(text=f"{win\_rate:.1f}%")

# Formatar valores monetários com cor

profit\_color = self.colors["accent"] if self.stats["profit"] >= 0 else self.colors["danger"]

self.profit\_label.config(text=f"${self.stats['profit']:.2f}", foreground=profit\_color)

self.start\_balance\_label.config(text=f"${self.stats['start\_balance']:.2f}")

self.current\_balance\_label.config(text=f"${self.stats['current\_balance']:.2f}")

def update\_active\_trades\_display(self):

"""Atualizar tabela de operações ativas"""

# Limpar tabela

for item in self.active\_tree.get\_children():

self.active\_tree.delete(item)

# Adicionar operações ativas

for trade\_id, trade in self.active\_trades.items():

# Formatar valores

direction = "CALL" if trade["direction"] == "call" else "PUT"

amount = f"${trade['amount']:.2f}"

open\_time = trade["open\_time"].strftime("%H:%M:%S")

expiry = f"{trade['expiration']}m"

martingale = f"Nível {trade['martingale\_level']}" if trade.get('martingale\_level', 0) > 0 else "Não"

# Inserir na tabela

self.active\_tree.insert("", tk.END, values=(

trade\_id, trade["pair"], direction, amount, open\_time, expiry, martingale

))

def update\_chart(self):

"""Atualizar gráfico com dados em tempo real"""

if not self.connected or not self.api:

messagebox.showwarning("Aviso", "Conecte-se primeiro para visualizar gráficos")

return

pair = self.chart\_pair\_var.get()

timeframe\_str = self.chart\_timeframe\_var.get()

# Converter timeframe para segundos

timeframe\_map = {"1m": 60, "5m": 300, "15m": 900, "1h": 3600}

timeframe = timeframe\_map.get(timeframe\_str, 60)

try:

# Obter candles

end\_time = time.time()

candles = self.api.get\_candles(pair, timeframe, 100, end\_time)

if not candles or len(candles) == 0:

self.log\_message(f"Nenhum dado disponível para {pair} em {timeframe\_str}", "warning")

return

# Converter para DataFrame

df = pd.DataFrame(candles)

# Limpar subplots

self.price\_subplot.clear()

self.indicator1\_subplot.clear()

self.indicator2\_subplot.clear()

# Configurar cores

self.price\_subplot.set\_facecolor(self.colors["card"])

self.indicator1\_subplot.set\_facecolor(self.colors["card"])

self.indicator2\_subplot.set\_facecolor(self.colors["card"])

# Criar timestamps

timestamps = [datetime.fromtimestamp(candle['from']) for candle in candles]

# Gráfico de preço

self.price\_subplot.plot(timestamps, df['close'], color='white', linewidth=1.5)

self.price\_subplot.set\_title(f"{pair} - {timeframe\_str}", color='white')

self.price\_subplot.tick\_params(axis='x', colors='white')

self.price\_subplot.tick\_params(axis='y', colors='white')

self.price\_subplot.grid(True, alpha=0.3)

# Adicionar Bollinger Bands

window = 20

rolling\_mean = df['close'].rolling(window=window).mean()

rolling\_std = df['close'].rolling(window=window).std()

upper\_band = rolling\_mean + (rolling\_std \* 2)

lower\_band = rolling\_mean - (rolling\_std \* 2)

self.price\_subplot.plot(timestamps, rolling\_mean, color='blue', alpha=0.7, linewidth=1)

self.price\_subplot.plot(timestamps, upper\_band, color='red', alpha=0.5, linewidth=1)

self.price\_subplot.plot(timestamps, lower\_band, color='green', alpha=0.5, linewidth=1)

# Calcular e adicionar RSI

delta = df['close'].diff()

gain = delta.where(delta > 0, 0)

loss = -delta.where(delta < 0, 0)

avg\_gain = gain.rolling(window=14).mean()

avg\_loss = loss.rolling(window=14).mean()

rs = avg\_gain / avg\_loss

rsi = 100 - (100 / (1 + rs))

self.indicator1\_subplot.plot(timestamps, rsi, color='purple', linewidth=1)

self.indicator1\_subplot.axhline(y=70, color='red', alpha=0.5, linestyle='--')

self.indicator1\_subplot.axhline(y=30, color='green', alpha=0.5, linestyle='--')

self.indicator1\_subplot.set\_title("RSI", color='white')

self.indicator1\_subplot.tick\_params(axis='x', colors='white')

self.indicator1\_subplot.tick\_params(axis='y', colors='white')

self.indicator1\_subplot.set\_ylim(0, 100)

self.indicator1\_subplot.grid(True, alpha=0.3)

# Calcular e adicionar MACD

exp1 = df['close'].ewm(span=12, adjust=False).mean()

exp2 = df['close'].ewm(span=26, adjust=False).mean()

macd = exp1 - exp2

signal = macd.ewm(span=9, adjust=False).mean()

histogram = macd - signal

self.indicator2\_subplot.plot(timestamps, macd, color='blue', linewidth=1, label='MACD')

self.indicator2\_subplot.plot(timestamps, signal, color='red', linewidth=1, label='Signal')

# Adicionar histograma do MACD

for i, (date, hist) in enumerate(zip(timestamps, histogram)):

color = 'green' if hist > 0 else 'red'

self.indicator2\_subplot.bar(date, hist, color=color, alpha=0.5, width=10)

self.indicator2\_subplot.set\_title("MACD", color='white')

self.indicator2\_subplot.tick\_params(axis='x', colors='white')

self.indicator2\_subplot.tick\_params(axis='y', colors='white')

self.indicator2\_subplot.grid(True, alpha=0.3)

self.indicator2\_subplot.legend(loc='upper left')

# Configurar layout e exibir

self.figure.tight\_layout()

self.chart\_canvas.draw()

self.log\_message(f"Gráfico atualizado: {pair} - {timeframe\_str}", "info")

except Exception as e:

self.log\_message(f"Erro ao atualizar gráfico: {str(e)}", "error")

logger.error(traceback.format\_exc())

def export\_history(self):

"""Exportar histórico para CSV"""

if not self.trade\_history:

messagebox.showinfo("Informação", "Não há histórico para exportar")

return

try:

filename = filedialog.asksaveasfilename(

defaultextension=".csv",

filetypes=[("CSV files", "\*.csv"), ("All files", "\*.\*")],

title="Exportar Histórico de Operações"

)

if not filename:

return # Cancelado pelo usuário

with open(filename, 'w') as f:

# Cabeçalho

f.write("ID,Par,Direção,Valor,Resultado,Lucro,Hora\_Abertura,Hora\_Fechamento,Nível\_Martingale\n")

# Filtrar se necessário

filter\_type = self.filter\_result\_var.get()

filtered\_history = []

if filter\_type == "VITÓRIAS":

filtered\_history = [t for t in self.trade\_history if t.get("result") == "WIN"]

elif filter\_type == "DERROTAS":

filtered\_history = [t for t in self.trade\_history if t.get("result") == "LOSS"]

else:

filtered\_history = self.trade\_history

# Dados

for trade in filtered\_history:

# Formatações

direction = "CALL" if trade["direction"] == "call" else "PUT"

result = trade.get("result", "")

profit = trade.get("profit", 0)

open\_time = trade["open\_time"].strftime("%Y-%m-%d %H:%M:%S") if isinstance(trade["open\_time"], datetime) else trade["open\_time"]

close\_time = ""

if "close\_time" in trade and trade["close\_time"]:

close\_time = trade["close\_time"].strftime("%Y-%m-%d %H:%M:%S") if isinstance(trade["close\_time"], datetime) else trade["close\_time"]

martingale = trade.get("martingale\_level", 0)

f.write(f"{trade['id']},{trade['pair']},{direction},{trade['amount']},{result},"

f"{profit},{open\_time},{close\_time},{martingale}\n")

self.log\_message(f"Histórico exportado para {filename}", "success")

except Exception as e:

self.log\_message(f"Erro ao exportar histórico: {str(e)}", "error")

messagebox.showerror("Erro", f"Falha ao exportar: {str(e)}")

def analyze\_market(self, pair, timeframe=60):

"""Analisar mercado para identificar oportunidades de entrada"""

if not self.api:

return None

try:

# Obter candles

end\_time = time.time()

candles = self.api.get\_candles(pair, timeframe, 50, end\_time)

if not candles or len(candles) < 30: # Precisamos de dados suficientes

self.log\_message(f"Dados insuficientes para análise de {pair}", "warning")

return None

# Converter para DataFrame

df = pd.DataFrame(candles)

# Obter modo de operação atual

operation\_mode = self.operation\_mode.get()

operation\_settings = self.operation\_settings[operation\_mode]

# Inicializar resultados

signals = []

confidence = 0

# Verificar quais indicadores usar

indicators = operation\_settings["indicators"]

# Calcular indicadores conforme configurado

if "RSI" in indicators:

# Calcular RSI

delta = df['close'].diff()

gain = delta.where(delta > 0, 0)

loss = -delta.where(delta < 0, 0)

avg\_gain = gain.rolling(window=14).mean()

avg\_loss = loss.rolling(window=14).mean()

rs = avg\_gain / avg\_loss

rsi = 100 - (100 / (1 + rs))

latest\_rsi = rsi.iloc[-1] if not rsi.empty else 50

if latest\_rsi < 30: # Sobrevenda

signals.append("buy")

confidence += 25

elif latest\_rsi > 70: # Sobrecompra

signals.append("sell")

confidence += 25

if "BOLLINGER" in indicators:

# Calcular Bandas de Bollinger

window = 20

rolling\_mean = df['close'].rolling(window=window).mean()

rolling\_std = df['close'].rolling(window=window).std()

upper\_band = rolling\_mean + (rolling\_std \* 2)

lower\_band = rolling\_mean - (rolling\_std \* 2)

latest\_price = df['close'].iloc[-1]

latest\_upper = upper\_band.iloc[-1] if not upper\_band.empty else latest\_price \* 1.02

latest\_lower = lower\_band.iloc[-1] if not lower\_band.empty else latest\_price \* 0.98

if latest\_price < latest\_lower: # Preço abaixo da banda inferior

signals.append("buy")

confidence += 25

elif latest\_price > latest\_upper: # Preço acima da banda superior

signals.append("sell")

confidence += 25

if "MACD" in indicators:

# Calcular MACD

exp1 = df['close'].ewm(span=12, adjust=False).mean()

exp2 = df['close'].ewm(span=26, adjust=False).mean()

macd = exp1 - exp2

signal\_line = macd.ewm(span=9, adjust=False).mean()

if len(macd) > 1 and len(signal\_line) > 1:

if macd.iloc[-2] < signal\_line.iloc[-2] and macd.iloc[-1] > signal\_line.iloc[-1]:

# Cruzamento para cima (bullish)

signals.append("buy")

confidence += 30

elif macd.iloc[-2] > signal\_line.iloc[-2] and macd.iloc[-1] < signal\_line.iloc[-1]:

# Cruzamento para baixo (bearish)

signals.append("sell")

confidence += 30

if "STOCHASTIC" in indicators:

# Calcular Estocástico

k\_period = 14

d\_period = 3

# Verificar se temos as colunas corretas

if 'low' not in df.columns or 'high' not in df.columns:

# Usar min/max em vez de low/high se não estiverem disponíveis

if 'min' in df.columns and 'max' in df.columns:

df['low'] = df['min']

df['high'] = df['max']

else:

# Se não tiver nenhuma das colunas, usar open/close

df['low'] = df['close'].rolling(window=k\_period).min()

df['high'] = df['close'].rolling(window=k\_period).max()

low\_min = df['low'].rolling(window=k\_period).min()

high\_max = df['high'].rolling(window=k\_period).max()

k = 100 \* ((df['close'] - low\_min) / (high\_max - low\_min))

d = k.rolling(window=d\_period).mean()

latest\_k = k.iloc[-1] if not k.empty else 50

latest\_d = d.iloc[-1] if not d.empty else 50

if latest\_k < 20 and latest\_d < 20: # Sobrevenda

signals.append("buy")

confidence += 20

elif latest\_k > 80 and latest\_d > 80: # Sobrecompra

signals.append("sell")

confidence += 20

# Verificar predominância de sinais

if signals:

buy\_count = signals.count("buy")

sell\_count = signals.count("sell")

# Determinar direção final do sinal

if buy\_count > sell\_count:

signal\_direction = "buy"

elif sell\_count > buy\_count:

signal\_direction = "sell"

else:

# Em caso de empate, verificar tendência recente

if "RSI" in indicators and not rsi.empty:

signal\_direction = "buy" if rsi.iloc[-1] < 50 else "sell"

else:

# Sem desempate claro, não enviar sinal

signal\_direction = None

# Verificar se a confiança atinge o limite mínimo

threshold = operation\_settings["signal\_threshold"]

if signal\_direction and confidence >= threshold:

return {

"pair": pair,

"direction": signal\_direction,

"confidence": confidence,

"price": df['close'].iloc[-1]

}

# Nenhum sinal forte o suficiente

return None

except Exception as e:

self.log\_message(f"Erro na análise de mercado para {pair}: {str(e)}", "error")

logger.error(traceback.format\_exc())

return None

def place\_trade(self, pair, direction, amount, expiration):

"""Realizar uma operação"""

if not self.api:

return None

try:

# Verificar se ainda está conectado

if not self.api.check\_connect():

self.log\_message("Conexão perdida. Reconectando...", "error")

connected, \_ = self.api.connect()

if not connected:

self.log\_message("Falha ao reconectar", "error")

self.disconnect()

return None

# Converter direção para o formato da API

api\_direction = "call" if direction == "buy" else "put"

self.log\_message(f"Colocando ordem: {api\_direction.upper()} em {pair} | Valor: ${amount} | Expiração: {expiration}m", "info")

# Executar ordem

try:

status, trade\_id = self.api.buy(amount, pair, api\_direction, expiration)

if status:

self.log\_message(f"Ordem colocada com sucesso: ID {trade\_id}", "success")

# Registrar operação

trade\_info = {

"id": trade\_id,

"pair": pair,

"direction": api\_direction,

"amount": amount,

"expiration": expiration,

"open\_time": datetime.now(),

"martingale\_level": 0,

"martingale\_parent": None

}

self.active\_trades[trade\_id] = trade\_info

return trade\_id

else:

# Verificar mensagem de erro específica

if isinstance(trade\_id, str) and "active is suspended" in trade\_id:

# Par suspenso temporariamente

self.log\_message(f"Par {pair} está temporariamente suspenso para operações. Removendo da lista.", "warning")

if pair in self.selected\_pairs:

self.selected\_pairs.remove(pair)

else:

self.log\_message(f"Falha ao colocar ordem: {trade\_id}", "error")

return None

except Exception as e:

self.log\_message(f"Erro na execução da ordem: {str(e)}", "error")

return None

except Exception as e:

self.log\_message(f"Erro ao colocar ordem: {str(e)}", "error")

logger.error(traceback.format\_exc())

return None

def check\_trade\_results(self):

"""Verificar resultados das operações ativas"""

if not self.api or not self.active\_trades:

return

# Obter configurações de risco atuais

risk\_mode = self.risk\_mode.get()

risk\_settings = self.risk\_settings[risk\_mode]

try:

# Verificar cada operação ativa

for trade\_id in list(self.active\_trades.keys()):

# Verificar se a operação ainda existe

if trade\_id not in self.active\_trades:

continue

# Obter resultado da operação

result = self.api.check\_win\_v3(trade\_id)

# None significa que ainda não expirou

if result is not None:

trade = self.active\_trades[trade\_id]

# Registrar horário de fechamento

trade["close\_time"] = datetime.now()

# Processar resultado

if result > 0: # Vitória

trade["result"] = "WIN"

trade["profit"] = result

self.stats["wins"] += 1

self.stats["profit"] += result

self.log\_message(f"✅ VITÓRIA: {trade['direction'].upper()} em {trade['pair']} = +${result:.2f}", "success")

else: # Derrota

trade["result"] = "LOSS"

trade["profit"] = result

self.stats["losses"] += 1

self.stats["profit"] += result

self.log\_message(f"❌ DERROTA: {trade['direction'].upper()} em {trade['pair']} = ${result:.2f}", "error")

# Verificar se deve aplicar martingale

if risk\_settings["martingale\_enabled"] and trade.get("martingale\_level", 0) < risk\_settings["max\_martingale"]:

new\_level = trade.get("martingale\_level", 0) + 1

new\_amount = trade["amount"] \* risk\_settings["martingale\_factor"]

self.log\_message(f"🔄 Aplicando Martingale nível {new\_level} em {trade['pair']}: ${new\_amount:.2f}", "warning")

# Colocar nova operação martingale

new\_trade\_id = self.place\_trade(

trade["pair"],

"buy" if trade["direction"] == "call" else "sell",

new\_amount,

trade["expiration"]

)

if new\_trade\_id:

self.active\_trades[new\_trade\_id]["martingale\_level"] = new\_level

self.active\_trades[new\_trade\_id]["martingale\_parent"] = trade\_id

# Adicionar ao histórico

self.trade\_history.append(trade)

# Remover da lista de ativos

del self.active\_trades[trade\_id]

# Incrementar contador total

self.stats["total\_trades"] += 1

except Exception as e:

self.log\_message(f"Erro ao verificar resultados: {str(e)}", "error")

logger.error(traceback.format\_exc())

def trading\_loop(self):

"""Loop principal de trading"""

try:

# Obter configurações

operation\_mode = self.operation\_mode.get()

risk\_mode = self.risk\_mode.get()

operation\_settings = self.operation\_settings[operation\_mode]

risk\_settings = self.risk\_settings[risk\_mode]

self.log\_message(f"Iniciando loop de trading com modo {operation\_mode} e gestão {risk\_mode}", "info")

self.log\_message(f"Monitorando pares: {', '.join(self.selected\_pairs)}", "info")

# Controle de loop

consecutive\_losses = 0

while self.is\_trading:

# Verificar conexão

if not self.api.check\_connect():

self.log\_message("Conexão perdida. Reconectando...", "error")

connected, \_ = self.api.connect()

if not connected:

self.log\_message("Falha ao reconectar. Interrompendo trading.", "error")

self.is\_trading = False

break

# Atualizar saldo

balance = self.api.get\_balance()

self.stats["current\_balance"] = balance

# Verificar resultados das operações ativas

self.check\_trade\_results()

# Verificar limites de perda e ganho

profit\_percent = ((balance - self.stats["start\_balance"]) / self.stats["start\_balance"]) \* 100

if profit\_percent <= -risk\_settings["max\_daily\_loss"]:

self.log\_message(f"Atingiu perda máxima diária de {risk\_settings['max\_daily\_loss']}%. Interrompendo trading.", "error")

self.is\_trading = False

break

if profit\_percent >= risk\_settings["daily\_goal"]:

self.log\_message(f"Atingiu meta diária de {risk\_settings['daily\_goal']}%! Interrompendo trading.", "success")

self.is\_trading = False

break

# Verificar número máximo de operações simultâneas

if len(self.active\_trades) >= risk\_settings["max\_open\_trades"]:

# Aguardar antes de verificar novamente

time.sleep(1)

continue

# Analisar mercado para cada par selecionado

for pair in self.selected\_pairs:

# Verificar se ainda podemos abrir mais operações

if len(self.active\_trades) >= risk\_settings["max\_open\_trades"]:

break

# Analisar

signal = self.analyze\_market(pair, operation\_settings["timeframe"])

if signal:

# Obter valor base

amount = risk\_settings["base\_amount"]

# Colocar ordem

trade\_id = self.place\_trade(

pair,

signal["direction"],

amount,

operation\_settings["expiration"]

)

if trade\_id:

self.log\_message(f"Sinal detectado: {signal['direction'].upper()} em {pair} (Confiança: {signal['confidence']}%)", "info")

# Pequena pausa para não sobrecarregar a API

time.sleep(0.5)

# Pausa para reduzir o consumo de CPU

time.sleep(1)

except Exception as e:

self.log\_message(f"Erro no loop de trading: {str(e)}", "error")

logger.error(traceback.format\_exc())

finally:

# Garantir que o sinalizador de trading seja atualizado

self.is\_trading = False

def on\_close(self):

"""Manipular evento de fechamento da janela"""

if self.is\_trading:

confirm = messagebox.askyesno("Confirmação",

"Trading em andamento. Deseja realmente sair?",

icon="warning")

if not confirm:

return

self.stop\_trading()

if self.connected:

self.disconnect()

self.master.destroy()

def main():

"""Função principal"""

try:

# Criar janela principal

try:

root = tk.Tk()

# Configurar janela

root.title("IQ Option Bot - Trading em Tempo Real")

# Verificar se a janela foi criada com sucesso

if root:

app = TradingBot(root)

root.mainloop()

else:

print("Erro ao criar janela principal. Iniciando modo headless...")

# Se não conseguir criar a GUI, iniciar em modo headless

bot = TradingBot()

bot.email\_var = tk.StringVar(value=EMAIL)

bot.password\_var = tk.StringVar(value=PASSWORD)

bot.account\_type = tk.StringVar(value="PRACTICE")

bot.log\_message = lambda msg, level="info": print(f"[{datetime.now().strftime('%H:%M:%S')}] {msg}")

bot.connect()

if bot.connected:

bot.start\_trading()

else:

print("Falha ao conectar. Verifique suas credenciais.")

except Exception as e:

print(f"Erro ao iniciar aplicação: {str(e)}")

print(traceback.format\_exc())

except Exception as e:

logger.error(f"Erro na aplicação: {str(e)}")

logger.error(traceback.format\_exc())

if \_\_name\_\_ == "\_\_main\_\_":

main()