```
#Region ;**** Directives created by AutoIt3Wrapper_GUI ****
#AutoIt3Wrapper_Icon=Include\RogueReader.ico
#AutoIt3Wrapper_Compression=4
#AutoIt3Wrapper_UseX64=y
#AutoIt3Wrapper_Res_Description=Trainer for ProjectRogue
#AutoIt3Wrapper_Res_Fileversion=5.0.0.37
#AutoIt3Wrapper_Res_Fileversion_AutoIncrement=y
#AutoIt3Wrapper_Res_ProductName=Rogue Reader
#AutoIt3Wrapper_Res_ProductVersion=4
#AutoIt3Wrapper_Res_CompanyName=Training Trainers.LLC
#AutoIt3Wrapper_Res_LegalCopyright=Use only for authorized security testing.
#AutoIt3Wrapper_Res_LegalTradeMarks=TrainingTrainersLLC
#AutoIt3Wrapper_Res_Language=1033
#AutoIt3Wrapper_Run_AU3Check=n
#AutoIt3Wrapper_Run_Tidy=y
#AutoIt3Wrapper_Tidy_Stop_OnError=n
#EndRegion ;**** Directives created by AutoIt3Wrapper_GUI ****
#Region ;**** Directives created by AutoIt3Wrapper_GUI ****
#AutoIt3Wrapper_Icon=Include\RogueReader.ico
#AutoIt3Wrapper_Compression=4
#AutoIt3Wrapper_UseX64=y
#AutoIt3Wrapper_Res_Description=Trainer for ProjectRogue
#AutoIt3Wrapper_Res_Fileversion=5.0.0.37
#AutoIt3Wrapper_Res_Fileversion_AutoIncrement=y
#AutoIt3Wrapper_Res_ProductName=Rogue Reader
#AutoIt3Wrapper_Res_ProductVersion=4
#AutoIt3Wrapper_Res_CompanyName=Training Trainers.LLC
#AutoIt3Wrapper_Res_LegalCopyright=Use only for authorized security testing.
#AutoIt3Wrapper_Res_LegalTradeMarks=TrainingTrainersLLC
#AutoIt3Wrapper_Res_Language=1033
#AutoIt3Wrapper_Run_AU3Check=n
#AutoIt3Wrapper_Tidy_Stop_OnError=n
#EndRegion ;**** Directives created by AutoIt3Wrapper_GUI ****

#include <GUIConstantsEx.au3>
#include <File.au3>
#include <WindowsConstants.au3>
#include <WinAPI.au3>
#include <Process.au3>
#include <Array.au3> ; For _ArraySearch

; --------------------------------------------------------------------------------
; 1) Define fallback constants for Lock/Unlock if your AutoIt version doesn't have them
; --------------------------------------------------------------------------------
If Not IsDeclared("SW_LOCKDRAW") Then
    Global Const $SW_LOCKDRAW = 133   ; numeric values introduced in v3.3.17
EndIf

If Not IsDeclared("SW_UNLOCKDRAW") Then
    Global Const $SW_UNLOCKDRAW = 134
EndIf

Opt("MouseCoordMode", 2)

Global $version = FileGetVersion(@ScriptFullPath)
Global Const $locationFile = @ScriptDir & "\Locations.ini"
Global $currentLocations = 1
Global $maxLocations = 20000
Global Const $sButtonConfigFile = @ScriptDir & "\NewButtonConfig.ini"

ConsoleWrite("Script Version: " & $version & @CRLF)

; --- Load Config Settings ---
Global $aTempBlocked[0][2]

If Not FileExists($sButtonConfigFile) Then CreateButtonDefaultConfig()
LoadButtonConfig()

Global $iCurrentIndex = 0
Global $aLocations = LoadLocations()              ; This may show error if the file is missing
Global $Debug = False
Global $LootIdleTimer = TimerInit()
Global $LootIdleWaiting = False

Global $LootQueued = False
Global $LootCount = 0
Global $LootReady = False
Global $LootTimer = TimerInit()
Global $PausedWalkerForLoot = False
Global $LastPlayerX = 0
Global $LastPlayerY = 0
Global $HadTarget = False
Global $LastTargetHeld = TimerInit()
Global $LastTargetTime = TimerInit()
Global $LootingCheckbox
Global $LootCheckX = -1
Global $LootCheckY = -1

; Define the game process and memory offsets
Global $ProcessName = "Project Rogue Client.exe"
Global $WindowName = "Project Rogue"
Global $TypeOffset = 0xBE7974        ; ; 0=Player, 1=Monster, etc
Global $AttackModeOffset = 0xB5BC00  ;
Global $PosYOffset = 0xBF9E08        ;
Global $PosXOffset = 0xBF9E10        ;
Global $HPOffset = 0x7C400           ;
Global $MaxHPOffset = 0x7C404        ;
Global $ChattOpenOffset = 0xB678D8   ;
Global $SicknessOffset = 0x7C5E4     ;
Global $BackPack = 0x731A8           ;
Global $BackPackMax = 0x731AC        ;

Global $MovmentSlider = 200 ;walk after removed from gui turned to solid state,

Global $currentTime = TimerInit()
Global $LastHealTime = TimerInit()
Global $lastX = 0
Global $lastY = 0
Global $Running = True
Global $HealerStatus = 0
Global $CureStatus = 0
Global $TargetStatus = 0
Global $MoveToLocationsStatus = 0
Global $iPrevValue = 95
Global $MPrevValue = " "
Global $hProcess = 0
Global $BaseAddress = 0
Global $TypeAddress, $AttackModeAddress, $PosXAddress, $PosYAddress
Global $HPAddress, $MaxHPAddress, $ChattOpenAddress, $SicknessAddress
Global $Type, $Chat, $Sickness, $AttackMode

Global $sicknessArray = [ _
        1, 2, 65, 66, 67, 68, 69, 72, 73, 81, 97, 98, 99, 257, 258, 513, 514, 515, 577, _
        8193, 8194, 8195, 8257, 8258, 8705, 8706, 8707, 8708, 8709, 8712, 8713, _
        8721, 8737, 8769, 8770, 16385, 16386, 16449, 16450, 16451, 16452, 16897, _
        16898, 24577, 24578, 24579, 24581, 24582, 24583, 24585, 24609, 24641, _
        24642, 24643, 24645, 24646, 24647, 24649, 25089, 25090, 25091, 25093, _
        25094, 25095, 25097, 25121, 33283, 33284, 33285, 33286, 33287, 33288, _
        33289, 33291, 33293, 33294, 33295, 33793, 41985, 41986, 41987, 41988, _
        41989, 41990, 41991, 41993, 41995]

Global $TargetDelay = 400, $HealDelay = 1700

; -------------------
; Create the GUI
; -------------------
;...;
Global $Gui = GUICreate($version, 248, 360, 15, 15)

Global $TypeLabel = GUICtrlCreateLabel("Target: N/A", 105, 21, 115, 15)
GUICtrlSetFont(-1, 8.5, 400, $GUI_FONTNORMAL, "$GUI_FONTNORMAL")
GUICtrlSetBkColor(-1, 0xBEBEBE)
Global $AttackModeLabel = GUICtrlCreateLabel("Attack: N/A", 105, 37, 115, 15)
GUICtrlSetFont(-1, 8.5, 400, $GUI_FONTNORMAL, "$GUI_FONTNORMAL")
GUICtrlSetBkColor(-1, 0xBEBEBE)
Global $PosXLabel = GUICtrlCreateLabel("X: N/A", 11, 23, 75, 15)
GUICtrlSetFont(-1, 8.5, 400, $GUI_FONTNORMAL, "$GUI_FONTNORMAL")
GUICtrlSetBkColor(-1, 0xBEBEBE)
Global $PosYLabel = GUICtrlCreateLabel("Y: N/A", 11, 39, 75, 15)
GUICtrlSetFont(-1, 8.5, 400, $GUI_FONTNORMAL, "$GUI_FONTNORMAL")
GUICtrlSetBkColor(-1, 0xBEBEBE)
Global $HPLabel = GUICtrlCreateLabel("HP: N/A /", 10, 187, 45, 15)
GUICtrlSetFont(-1, 8.5, 400, $GUI_FONTNORMAL, "$GUI_FONTNORMAL")
GUICtrlSetBkColor(-1, 0xBEBEBE)
Global $ChatLabel = GUICtrlCreateLabel("Chat: N/A", 105, 69, 115, 15)
GUICtrlSetFont(-1, 8.5, 400, $GUI_FONTNORMAL, "$GUI_FONTNORMAL")
GUICtrlSetBkColor(-1, 0xBEBEBE)
Global $SicknessLabel = GUICtrlCreateLabel("Sickness: N/A", 105, 53, 115, 15)
GUICtrlSetFont(-1, 8.5, 400, $GUI_FONTNORMAL, "$GUI_FONTNORMAL")
GUICtrlSetBkColor(-1, 0xBEBEBE)
Global $MaxHPLabel = GUICtrlCreateLabel("N/A", 55, 187, 30, 15)
GUICtrlSetFont(-1, 8.5, 400, $GUI_FONTNORMAL, "$GUI_FONTNORMAL")
GUICtrlSetBkColor(-1, 0xBEBEBE)
Global $TargetLabel = GUICtrlCreateLabel("Target: Off", 10, 124, 75, 15)
GUICtrlSetFont(-1, 8.5, 400, $GUI_FONTNORMAL, "$GUI_FONTNORMAL")
GUICtrlSetBkColor(-1, 0xBEBEBE)
Global $HealerLabel = GUICtrlCreateLabel("Healer: Off", 10, 92, 75, 15)
GUICtrlSetFont(-1, 8.5, 400, $GUI_FONTNORMAL, "$GUI_FONTNORMAL")
GUICtrlSetBkColor(-1, 0xBEBEBE)
Global $WalkerLabel = GUICtrlCreateLabel("Walker: Off", 10, 140, 75, 15)
GUICtrlSetFont(-1, 8.5, 400, $GUI_FONTNORMAL, "$GUI_FONTNORMAL")
GUICtrlSetBkColor(-1, 0xBEBEBE)
Global $BackPackLabel = GUICtrlCreateLabel("Weight: N/A", 10, 203, 75, 15)
GUICtrlSetFont(-1, 8.5, 400, $GUI_FONTNORMAL, "$GUI_FONTNORMAL")
```

```
GUICtrlSetBkColor(-1, 0xBEBEBE)
Global $CureLabel = GUICtrlCreateLabel("Cure: Off", 10, 108, 75, 15)
GUICtrlSetFont(-1, 8.5, 400, $GUI_FONTNORMAL, "$GUI_FONTNORMAL")
GUICtrlSetBkColor(-1, 0xBEBEBE)
Global $KillButton = GUICtrlCreateButton("Kill Rogue", 10, 315, 110, 30)
Global $ExitButton = GUICtrlCreateButton("Exit", 120, 315, 110, 30)
Global $ReverseLoopCheckbox = GUICtrlCreateCheckbox("Reversed Walker", 105, 205, 115, 20)
GUICtrlSetFont(-1, 8.5, 400, $GUI_FONTNORMAL, "$GUI_FONTNORMAL")
GUICtrlSetBkColor(-1, 0xBEBEBE)
Global $LootingCheckbox = GUICtrlCreateCheckbox("Autoloot", 107, 185, 115, 20)
GUICtrlSetFont(-1, 8.5, 400, $GUI_FONTNORMAL, "$GUI_FONTNORMAL")
GUICtrlSetBkColor(-1, 0xBEBEBE)
Global $Checkbox = GUICtrlCreateCheckbox("Old Style Pothack", 105, 225, 115, 20)
GUICtrlSetFont(-1, 8.5, 400, $GUI_FONTNORMAL, "$GUI_FONTNORMAL")
GUICtrlSetBkColor(-1, 0xBEBEBE)
Global $Helpers = GUICtrlCreateLabel("HELPERS", 8, 75, 80, 15)
GUICtrlSetFont(-1, 8.5, 400, $GUI_FONTNORMAL, "$GUI_FONTNORMAL")
GUICtrlSetBkColor(-1, 0x808080)
Global $Character = GUICtrlCreateLabel("CHARACTER", 8, 170, 80, 15)
GUICtrlSetFont(-1, 8.5, 400, $GUI_FONTNORMAL, "$GUI_FONTNORMAL")
GUICtrlSetBkColor(-1, 0x808080)
Global $Position = GUICtrlCreateLabel("POSITION", 8, 5, 80, 15)
GUICtrlSetFont(-1, 8.5, 400, $GUI_FONTNORMAL, "$GUI_FONTNORMAL")
GUICtrlSetBkColor(-1, 0x808080)
Global $Information = GUICtrlCreateLabel("INFORMATION", 103, 4, 120, 15)
GUICtrlSetFont(-1, 8.5, 400, $GUI_FONTNORMAL, "$GUI_FONTNORMAL")
GUICtrlSetBkColor(-1, 0x808080)
Global $Options = GUICtrlCreateLabel("OPTIONS", 103, 169, 120, 11)
GUICtrlSetFont(-1, 8.5, 400, $GUI_FONTNORMAL, "$GUI_FONTNORMAL")
GUICtrlSetBkColor(-1, 0x808080)
Global $HealToggle = GUICtrlCreateButton("HEAL", 95, 92, 60, 15)
GUICtrlSetFont(-1, 8.5, 400, $GUI_FONTNORMAL, "$GUI_FONTNORMAL")
Global $CureToggle = GUICtrlCreateButton("CURE", 95, 108, 60, 15)
GUICtrlSetFont(-1, 8.5, 400, $GUI_FONTNORMAL, "$GUI_FONTNORMAL")
Global $TargetToggle = GUICtrlCreateButton("TARGET", 95, 124, 60, 15)
GUICtrlSetFont(-1, 8.5, 400, $GUI_FONTNORMAL, "$GUI_FONTNORMAL")
Global $WalkerToggle = GUICtrlCreateButton("WALKER", 95, 140, 60, 15)
GUICtrlSetFont(-1, 8.5, 400, $GUI_FONTNORMAL, "$GUI_FONTNORMAL")
Global $ToggleAll = GUICtrlCreateButton("ToggleAll", 155, 94, 71, 60)
GUICtrlSetFont(-1, 8.5, 400, $GUI_FONTNORMAL, "$GUI_FONTNORMAL")
Global $HP2Label = GUICtrlCreateLabel("RealHp: N/A", 11, 224, 76, 21)
GUICtrlSetBkColor(-1, 0x9D9597)

Global $healSlider = GUICtrlCreateSlider(10, 270, 226, 36)
GUICtrlSetData($healSlider, 85)




GUISetState(@SW_SHOW)

; ------------------------------------------------------------------------
;   :                      STREAMLINED MAIN LOOP
; ------------------------------------------------------------------------
While $Running
    Local $msg = GUIGetMsg()
    ; ---- Handle GUI messages first ----
    Switch $msg
```

```
        Case $ExitButton, $GUI_EVENT_CLOSE
            _WinAPI_CloseHandle($hProcess)
            GUIDelete($Gui)
            Exit
        Case $KillButton
            Local $pidCheck = ProcessExists($ProcessName)
            If $pidCheck Then ProcessClose($pidCheck)
        Case $HealToggle
            ToggleHealer()
        Case $CureToggle
            ToggleCure()
        Case $TargetToggle
            ToggleTarget()
        Case $WalkerToggle
            ToggleWalker()
        Case $ToggleAll
            ToggleAllHelpers()
    EndSwitch

    ; ---- Now background work ----
    Local $ProcessID = ProcessExists($ProcessName)
    If Not $ProcessID Then
        If $hProcess <> 0 Then _WinAPI_CloseHandle($hProcess)
        $hProcess = 0
        $BaseAddress = 0
        Sleep(10)
        ContinueLoop
    EndIf

    If $hProcess = 0 Then
        ConnectToBaseAddress()
        If $BaseAddress = 0 Or $hProcess = 0 Then
            Sleep(10)
            ContinueLoop
        Else
            ChangeAddressToBase()
        EndIf
    EndIf

    GUIReadMemory()

    If $Chat = 0 Then
        If $CureStatus = 1 And $Chat = 0 Then CureMe()
        If $HealerStatus = 1 And $Chat = 0 Then TimeToHeal()
        If $TargetStatus = 1 Then AttackModeReader()
        If GUICtrlRead($LootingCheckbox) = $GUI_CHECKED Then HandleLootQueue()

        If $MoveToLocationsStatus = 1 And Not $LootQueued And $Chat = 0 Then
            Local $result = MoveToLocationsStep($aLocations, $iCurrentIndex)
            If @error Then $MoveToLocationsStatus = 0
        EndIf
    EndIf

    Sleep(50) ; lighter sleep, much more responsive
WEnd

GUIDelete($Gui)
_WinAPI_CloseHandle($hProcess)
```

```
ConsoleWrite("[Debug] Trainer closed by script end" & @CRLF)
Exit

; -----------------------------------------------------------------------------
;                               LOAD CONFIG
; -----------------------------------------------------------------------------
Func LoadButtonConfig()
    Local $sButtonConfigFile = @ScriptDir & "\NewButtonConfig.ini"

    ; Remove old/unused entries
    IniDelete($sButtonConfigFile, "Hotkeys", "TogglePauseHotkey")
    IniDelete($sButtonConfigFile, "Hotkeys", "PlayLocationsHotkey")

    ; Define the hotkeys and default values
    Local $aKeys[7][2] = [ _
            ["HealHotkey", "{" & Chr(96) & "}"], _
            ["CureHotkey", "{-}"], _
            ["TargetHotkey", "{=}"], _
            ["ExitHotkey", "{#}"], _
            ["SaveLocationHotkey", "{F7}"], _
            ["EraseLocationsHotkey", "{F8}"], _
            ["MoveToLocationsHotkey", "{!}"] _
            ]

    Local $bMissingKeys = False
    For $i = 0 To UBound($aKeys) - 1
        Local $sKey = IniRead($sButtonConfigFile, "Hotkeys", $aKeys[$i][0], "")
        If $sKey = "" Then
            ConsoleWrite("[Warning] Missing key: " & $aKeys[$i][0] & ". Will create default config." & @CRLF)
            $bMissingKeys = True
            ExitLoop
        EndIf
    Next

    ; If any key was missing, recreate the default configuration
    If $bMissingKeys Then
        CreateButtonDefaultConfig()
    EndIf

    ; Re-read keys
    For $i = 0 To UBound($aKeys) - 1
        Local $sKey = IniRead($sButtonConfigFile, "Hotkeys", $aKeys[$i][0], $aKeys[$i][1])

        Switch $aKeys[$i][0]
            Case "HealHotkey"
                HotKeySet($sKey, "Hotkeyshit")
            Case "CureHotkey"
                HotKeySet($sKey, "CureKeyShit")
            Case "TargetHotkey"
                HotKeySet($sKey, "TargetKeyShit")
            Case "ExitHotkey"
                HotKeySet($sKey, "KilledWithFire")
            Case "SaveLocationHotkey"
                HotKeySet($sKey, "SaveLocation")
            Case "EraseLocationsHotkey"
                HotKeySet($sKey, "EraseLocations")
            Case "MoveToLocationsHotkey"
                HotKeySet($sKey, "MoveToLocations")
        EndSwitch

        ConsoleWrite("[Info] Hotkey for " & $aKeys[$i][0] & " set to " & $sKey & @CRLF)
    Next
EndFunc   ;==>LoadButtonConfig

Func Min($a, $b)
    If $a < $b Then
        Return $a
    Else
        Return $b
    EndIf
EndFunc   ;==>Min

Func QueueLootPattern()
    Global $LootQueue

    ; Screen click coordinates
    Local $rawX[8] = [320, 350, 385, 325, 385, 325, 350, 385]
    Local $rawY[8] = [325, 320, 325, 355, 355, 385, 390, 385]

    ; Shuffle logic
    Local $used[8] = [False, False, False, False, False, False, False, False]
    For $i = 0 To 7
        Do
            Local $rand = Random(0, 7, 1)
        Until Not $used[$rand]
        $LootQueue[$i][0] = $rawX[$rand]
        $LootQueue[$i][1] = $rawY[$rand]
        $used[$rand] = True
    Next

    ConsoleWrite("[Loot] New loot pattern queued." & @CRLF)
EndFunc   ;==>QueueLootPattern

; ----------------------------------------------------------------------------
;                               HANDLE LOOT QUEUE
; ----------------------------------------------------------------------------
Func HandleLootQueue()
    Global $hProcess, $BaseAddress, $WindowName
    Global $LootQueued, $LootCount, $LootReady
    Global $MoveToLocationsStatus, $PausedWalkerForLoot
    Global $PosXAddress, $PosYAddress
    Global $LastPlayerX, $LastPlayerY
    Global $LootIdleTimer, $LootIdleWaiting

    ; No loot queued? Skip
    If Not $LootQueued Or $LootCount = 0 Then Return
    ; Not finished waiting for idle? Skip
    If Not $LootIdleWaiting Then Return
    ; 750ms idle time check
    If TimerDiff($LootIdleTimer) < 750 Then Return

    ; Check movement
    Local $PlayerX = _ReadMemory($hProcess, $PosXAddress)
    Local $PlayerY = _ReadMemory($hProcess, $PosYAddress)
    If $PlayerX <> $LastPlayerX Or $PlayerY <> $LastPlayerY Then
        ConsoleWrite("[Loot] Player moved before looting. Cancelling." & @CRLF)
        $LootQueued = False
        $LootCount = 0
        $LootReady = False
        $LootIdleWaiting = False
        Return
    EndIf

    ; Pause walker
    If $MoveToLocationsStatus = 1 Then
        $MoveToLocationsStatus = 0
        $PausedWalkerForLoot = True
        ConsoleWrite("[Loot] Walker paused for looting." & @CRLF)
    EndIf

    ; Looting starts
    ; Calculate clicks per tile based on kill count
    Local $clicksPerTile = CalculateLootClicks($LootCount)

    ConsoleWrite("[Loot] Looting now with " & $clicksPerTile & " clicks per tile." & @CRLF)

    Local $memX[8] = [192, 175, 160, 162, 162, 175, 192, 192]
    Local $memY[8] = [161, 159, 161, 176, 191, 194, 191, 176]
    Local $clickX[8] = [385, 350, 320, 325, 325, 350, 385, 385]
    Local $clickY[8] = [325, 320, 325, 355, 385, 390, 385, 355]

    Local $used[8] = [False, False, False, False, False, False, False, False]

    For $i = 0 To 7
        Do
            Local $rand = Random(0, 7, 1)
        Until Not $used[$rand]
        $used[$rand] = True

        _WriteMemory($hProcess, $BaseAddress + 0xA669F0, $memX[$rand])
        _WriteMemory($hProcess, $BaseAddress + 0xB5BC0C, $memY[$rand])

        For $j = 1 To $clicksPerTile
            ControlClick($WindowName, "", "", "right", 1, $clickX[$rand], $clickY[$rand])
            Sleep(1)
        Next

        ConsoleWrite("[Loot] Clicked (" & $clickX[$rand] & "," & $clickY[$rand] & ") x" & $clicksPerTile & @CRLF)
    Next

    ; Reset state
    $LootQueued = False
    $LootCount = 0
    $LootReady = False
    $LootIdleWaiting = False

    ; Resume walker
    If $PausedWalkerForLoot Then
        $MoveToLocationsStatus = 1
        $PausedWalkerForLoot = False
        ConsoleWrite("[Loot] Walker resumed after looting." & @CRLF)
    EndIf
EndFunc   ;==>HandleLootQueue
```

```
Func CalculateLootClicks($kills)
    If $kills <= 0 Then
        Return 0
    ElseIf $kills <= 3 Then
        Return 4
    ElseIf $kills <= 6 Then
        Return 6
    ElseIf $kills <= 9 Then
        Return 8
    ElseIf $kills <= 12 Then
        Return 10
    ElseIf $kills <= 15 Then
        Return 12
    ElseIf $kills <= 18 Then
        Return 14
    Else
        Return 16
    EndIf
EndFunc   ;==>CalculateLootClicks

Func ClickTile($x, $y)
    MouseClick("right", $x, $y, 1, 0)
EndFunc   ;==>ClickTile


Func CreateButtonDefaultConfig()
    Local $sButtonConfigFile = @ScriptDir & "\NewButtonConfig.ini"
    Local $aKeys[7][2] = [ _
            ["HealHotkey", "{" & Chr(96) & "}"], _
            ["CureHotkey", "{-}"], _
            ["TargetHotkey", "{=}"], _
            ["ExitHotkey", "{#}"], _
            ["SaveLocationHotkey", "{F7}"], _
            ["EraseLocationsHotkey", "{F8}"], _
            ["MoveToLocationsHotkey", "{!}"] _
            ]
    For $i = 0 To UBound($aKeys) - 1
        IniWrite($sButtonConfigFile, "Hotkeys", $aKeys[$i][0], $aKeys[$i][1])
    Next
    ConsoleWrite("[Info] Default ButtonConfig.ini created with hotkeys." & @CRLF)
EndFunc   ;==>CreateButtonDefaultConfig

; ------------------------------------------------------------------------------
;   Function to Open Process & Retrieve Base Address
; ------------------------------------------------------------------------------
Func ConnectToBaseAddress()
    Global $hProcess
    Global $ProcessID
    Global $BaseAddress

    $hProcess = _WinAPI_OpenProcess(0x1F0FFF, False, $ProcessID)
    If $hProcess = 0 Then
        ConsoleWrite("[Error] Failed to open process! Try running as administrator." & @CRLF)
        Return
    EndIf

    $BaseAddress = _GetModuleBase_EnumModules($hProcess)
    If $BaseAddress = 0 Then
```

```
        ConsoleWrite("[Error] Failed to obtain a valid base address!" & @CRLF)
    EndIf
EndFunc   ;==>ConnectToBaseAddress

; ----------------------------------------------------------------------------
;                        READ AND UPDATE GUI FROM MEMORY
; ----------------------------------------------------------------------------
Func GUIReadMemory()
    Global $hProcess
    Global $Type, $TypeAddress
    Global $WalkerLabel, $MoveToLocationsStatus
    Global $AttackMode, $AttackModeAddress
    Global $PosXAddress, $PosYAddress
    Global $HPAddress, $MaxHPAddress
    Global $ChattOpenAddress, $Chat
    Global $SicknessAddress, $Sickness
    Global $BackPack, $BackPackMax
    Global $BackPackAddress, $BackPackMaxAddress
    Global $HealerStatus, $CureStatus, $TargetStatus
    Global $HealerLabel, $CureLabel, $TargetLabel
    Global $LootQueued, $LootCount, $LootReady, $LootIdleWaiting

    If $hProcess = 0 Then Return

    ; Read Type
    $Type = _ReadMemory($hProcess, $TypeAddress)
    If $Type = 0 Then
        GUICtrlSetData($TypeLabel, "Type: Player")
    ElseIf $Type = 1 Then
        GUICtrlSetData($TypeLabel, "Type: Monster")
    ElseIf $Type = 2 Then
        GUICtrlSetData($TypeLabel, "Type: NPC")
    ElseIf $Type = 65535 Then
        GUICtrlSetData($TypeLabel, "Type: No Target")
    Else
        GUICtrlSetData($TypeLabel, "Type: Unknown (" & $Type & ")")
    EndIf

    ; Walker On/Off
    If $MoveToLocationsStatus = 0 Then
        GUICtrlSetData($WalkerLabel, "Walker: Off")
    ElseIf $MoveToLocationsStatus = 1 Then
        GUICtrlSetData($WalkerLabel, "Walker: On")
    Else
        GUICtrlSetData($WalkerLabel, "Error: Broken")
    EndIf

    ; Attack Mode
    $AttackMode = _ReadMemory($hProcess, $AttackModeAddress)
    If $AttackMode = 0 Then
        GUICtrlSetData($AttackModeLabel, "Attack Mode: Safe")
    ElseIf $AttackMode = 1 Then
        GUICtrlSetData($AttackModeLabel, "Attack Mode: Attack")
    Else
        GUICtrlSetData($AttackModeLabel, "Attack Mode: No Target")
    EndIf

    ; Position
```

```
Local $PosX = _ReadMemory($hProcess, $PosXAddress)
Local $PosY = _ReadMemory($hProcess, $PosYAddress)
GUICtrlSetData($PosXLabel, "Pos X: " & $PosX)
GUICtrlSetData($PosYLabel, "Pos Y: " & $PosY)

; HP
Local $HP = _ReadMemory($hProcess, $HPAddress)
GUICtrlSetData($HPLabel, "HP: " & $HP)
GUICtrlSetData($HP2Label, "RealHp: " & ($HP / 65536))

; MaxHP
Local $MaxHP = _ReadMemory($hProcess, $MaxHPAddress)
GUICtrlSetData($MaxHPLabel, "MaxHP: " & $MaxHP)

; Chat
Local $ChatVal = _ReadMemory($hProcess, $ChattOpenAddress)
$Chat = $ChatVal
GUICtrlSetData($ChatLabel, "Chat: " & $ChatVal)

; Sickness
Local $SickVal = _ReadMemory($hProcess, $SicknessAddress)
$Sickness = $SickVal
Local $SicknessDescription = GetSicknessDescription($SickVal)
GUICtrlSetData($SicknessLabel, "Sickness: " & $SicknessDescription)

; Backpack Weight
Local $bpWeight = _ReadMemory($hProcess, $BackPackAddress)
Local $bpMax = _ReadMemory($hProcess, $BackPackMaxAddress)
GUICtrlSetData($BackPackLabel, "Weight " & $bpWeight & " / " & $bpMax)

; --- Death Detection via sudden teleport ---
Local Static $lastX = -1, $lastY = -1
If $lastX <> -1 And $lastY <> -1 Then
    Local $dx = Abs($PosX - $lastX)
    Local $dy = Abs($PosY - $lastY)
    If $dx > 25 Or $dy > 25 Then
        ConsoleWrite("[DeathDetect] Large movement detected: Î”X=" & $dx & ", Î”Y=" & $dy & ". Assuming death." & @

        ; Disable all helpers
        If $MoveToLocationsStatus = 1 Then
            $MoveToLocationsStatus = 0
            GUICtrlSetData($WalkerLabel, "Walker: Off")
            ConsoleWrite("[DeathDetect] Walker disabled." & @CRLF)
        EndIf
        ;
        ;           If $TargetStatus = 1 Then
        ;               $TargetStatus = 0
        ;               GUICtrlSetData($TargetLabel, "Target: Off")
        ;               ConsoleWrite("[DeathDetect] Targeting disabled." & @CRLF)
        ;           EndIf
        ;
        ;           If $HealerStatus = 1 Then
        ;               $HealerStatus = 0
        ;               GUICtrlSetData($HealerLabel, "Healer: Off")
        ;               ConsoleWrite("[DeathDetect] Healer disabled." & @CRLF)
        ;           EndIf
        ;
        ;           If $CureStatus = 1 Then
        ;           $CureStatus = 0
        ;           GUICtrlSetData($CureLabel, "Cure: Off")
        ;           ConsoleWrite("[DeathDetect] Cure disabled." & @CRLF)
        ;         EndIf

        ; Clear any loot
        $LootQueued = False
        $LootCount = 0
        $LootReady = False
        $LootIdleWaiting = False
      EndIf
    EndIf
    $lastX = $PosX
    $lastY = $PosY
EndFunc   ;==>GUIReadMemory

Func _ReadMemory($hProc, $pAddress)
    If $hProc = 0 Or $pAddress = 0 Then Return 0

    Local $tBuffer = DllStructCreate("dword")
    Local $aRead = DllCall("kernel32.dll", "bool", "ReadProcessMemory", _
        "handle", $hProc, _
        "ptr", $pAddress, _
        "ptr", DllStructGetPtr($tBuffer), _
        "dword", DllStructGetSize($tBuffer), _
        "ptr", 0)
    If @error Or Not $aRead[0] Then Return 0
    Return DllStructGetData($tBuffer, 1)
EndFunc   ;==>_ReadMemory

Func _GetModuleBase_EnumModules($hProc)
    Local $hPsapi = DllOpen("psapi.dll")
    If $hPsapi = 0 Then Return 0

    Local $tModules = DllStructCreate("ptr[1024]")
    Local $tBytesNeeded = DllStructCreate("dword")
    Local $aCall = DllCall("psapi.dll", "bool", "EnumProcessModules", _
        "handle", $hProc, _
        "ptr", DllStructGetPtr($tModules), _
        "dword", DllStructGetSize($tModules), _
        "ptr", DllStructGetPtr($tBytesNeeded))
    If @error Or Not $aCall[0] Then
      DllClose($hPsapi)
      Return 0
    EndIf

    ; The first module in the list is usually the main EXE
    Local $pBaseAddress = DllStructGetData($tModules, 1, 1)
    DllClose($hPsapi)
    Return $pBaseAddress
EndFunc   ;==>_GetModuleBase_EnumModules

Func ChangeAddressToBase()
    Global $BaseAddress
    Global $TypeOffset, $AttackModeOffset, $PosXOffset, $PosYOffset
    Global $HPOffset, $MaxHPOffset, $ChattOpenOffset, $SicknessOffset
    Global $BackPack, $BackPackMax
    Global $TypeAddress, $AttackModeAddress, $PosXAddress, $PosYAddress
    Global $HPAddress, $MaxHPAddress, $ChattOpenAddress, $SicknessAddress
    Global $BackPackAddress, $BackPackMaxAddress

    $TypeAddress = $BaseAddress + $TypeOffset
    $AttackModeAddress = $BaseAddress + $AttackModeOffset
    $PosXAddress = $BaseAddress + $PosXOffset
    $PosYAddress = $BaseAddress + $PosYOffset
    $HPAddress = $BaseAddress + $HPOffset
    $MaxHPAddress = $BaseAddress + $MaxHPOffset
    $ChattOpenAddress = $BaseAddress + $ChattOpenOffset
    $SicknessAddress = $BaseAddress + $SicknessOffset
    $BackPackAddress = $BaseAddress + $BackPack
    $BackPackMaxAddress = $BaseAddress + $BackPackMax
EndFunc   ;==>ChangeAddressToBase


; ------------------------------------------------------------------------
;                           Hotkey Toggle Functions
; ------------------------------------------------------------------------
#Region ;toggles;
Func Hotkeyshit()
    Global $HealerStatus
    $HealerStatus = Not $HealerStatus
    GUICtrlSetData($HealerLabel, "Healer: " & ($HealerStatus ? "On" : "Off"))
EndFunc   ;==>Hotkeyshit

Func CureKeyShit()
    Global $CureStatus
    $CureStatus = Not $CureStatus
    GUICtrlSetData($CureLabel, "Cure: " & ($CureStatus ? "On" : "Off"))
EndFunc   ;==>CureKeyShit

Func TargetKeyShit()
    Global $TargetStatus
    $TargetStatus = Not $TargetStatus
    GUICtrlSetData($TargetLabel, "Target: " & ($TargetStatus ? "On" : "Off"))
EndFunc   ;==>TargetKeyShit

Func KilledWithFire()
    Global $Debug
    If $Debug Then ConsoleWrite("Killed with fire" & @CRLF)
    Exit
EndFunc   ;==>KilledWithFire


Func ToggleHealer()
    Global $HealerStatus
    $HealerStatus = Not $HealerStatus
    GUICtrlSetData($HealerLabel, "Healer: " & ($HealerStatus ? "On" : "Off"))
    ConsoleWrite("[GUI] Healer toggled to: " & ($HealerStatus ? "On" : "Off") & @CRLF)
EndFunc   ;==>ToggleHealer

Func ToggleCure()
    Global $CureStatus
    $CureStatus = Not $CureStatus
    GUICtrlSetData($CureLabel, "Cure: " & ($CureStatus ? "On" : "Off"))
    ConsoleWrite("[GUI] Cure toggled to: " & ($CureStatus ? "On" : "Off") & @CRLF)
EndFunc   ;==>ToggleCure
```

```
Func ToggleTarget()
    Global $TargetStatus
    $TargetStatus = Not $TargetStatus
    GUICtrlSetData($TargetLabel, "Target: " & ($TargetStatus ? "On" : "Off"))
    ConsoleWrite("[GUI] Target toggled to: " & ($TargetStatus ? "On" : "Off") & @CRLF)
EndFunc   ;==>ToggleTarget

Func ToggleWalker()
    Global $MoveToLocationsStatus, $aLocations, $iCurrentIndex

    If $MoveToLocationsStatus = 0 Then
        MoveToLocations()
        MoveToLocationsStep($aLocations, $iCurrentIndex) ; <<< NEW LINE!
        GUICtrlSetData($WalkerLabel, "Walker: On")
        ConsoleWrite("[GUI] Walker toggled to: On" & @CRLF)
    Else
        $MoveToLocationsStatus = 0
        GUICtrlSetData($WalkerLabel, "Walker: Off")
        ConsoleWrite("[GUI] Walker toggled to: Off" & @CRLF)
    EndIf
EndFunc   ;==>ToggleWalker

Func ToggleAllHelpers()
    Global $HealerStatus, $CureStatus, $TargetStatus, $MoveToLocationsStatus

    Local $TotalOn = 0
    If $HealerStatus Then $TotalOn += 1
    If $CureStatus Then $TotalOn += 1
    If $TargetStatus Then $TotalOn += 1
    If $MoveToLocationsStatus = 1 Then $TotalOn += 1

    If $TotalOn >= 1 Then
        ; Turn all OFF
        $HealerStatus = 0
        $CureStatus = 0
        $TargetStatus = 0
        $MoveToLocationsStatus = 0

        GUICtrlSetData($HealerLabel, "Healer: Off")
        GUICtrlSetData($CureLabel, "Cure: Off")
        GUICtrlSetData($TargetLabel, "Target: Off")
        GUICtrlSetData($WalkerLabel, "Walker: Off")

        ConsoleWrite("[GUI] ToggleAll: All turned OFF" & @CRLF)
    Else
        ; Turn all ON
        $HealerStatus = 1
        $CureStatus = 1
        $TargetStatus = 1
        $MoveToLocationsStatus = 1

        GUICtrlSetData($HealerLabel, "Healer: On")
        GUICtrlSetData($CureLabel, "Cure: On")
        GUICtrlSetData($TargetLabel, "Target: On")
        GUICtrlSetData($WalkerLabel, "Walker: On")

        ConsoleWrite("[GUI] ToggleAll: All turned ON" & @CRLF)
    EndIf
EndFunc   ;==>ToggleAllHelpers
#EndRegion ;toggles;
; --------------------------------------------------------------------------
; Optional: Return a more human label for some â€œSickâ€• codes
; --------------------------------------------------------------------------
Func GetSicknessDescription($Sick)
    Local $SicknessDescription = "Unknown"
    Switch $Sick
        Case 1
            $SicknessDescription = "Poison1 (" & $Sick & ")"
        Case 2
            $SicknessDescription = "Disease1 (" & $Sick & ")"
            ; ...
        Case Else
            $SicknessDescription = $Sick
    EndSwitch
    Return $SicknessDescription
EndFunc   ;==>GetSicknessDescription

; --------------------------------------------------------------------------
;                             LOCATION LOADING
; --------------------------------------------------------------------------
Func LoadLocations()
    If Not FileExists($locationFile) Then
        ConsoleWrite("[Error] Location file not found: " & $locationFile & @CRLF)
        Return SetError(1, 0, 0)
    EndIf

    Local $aLines = FileReadToArray($locationFile)
    If @error Then
        ConsoleWrite("[Error] Failed to read file: " & $locationFile & @CRLF)
        Return SetError(2, 0, 0)
    EndIf

    Local $iLocationCount = 0
    Dim $aTempLocations[UBound($aLines)][2]

    For $i = 0 To UBound($aLines) - 1
        Local $aMatches = StringRegExp($aLines[$i], "X:(\d+);Y:(\d+)", 3)
        If Not @error And UBound($aMatches) = 2 Then
            $aTempLocations[$iLocationCount][0] = Int($aMatches[0])
            $aTempLocations[$iLocationCount][1] = Int($aMatches[1])
            $iLocationCount += 1
        Else
            ConsoleWrite("[Warning] Failed to parse line " & $i & ": " & $aLines[$i] & @CRLF)
        EndIf
    Next

    If $iLocationCount = 0 Then
        ConsoleWrite("[Warning] No valid locations found in " & $locationFile & @CRLF)
        Return SetError(3, 0, 0)
    EndIf

    ReDim $aTempLocations[$iLocationCount][2]
    ConsoleWrite("[Success] Loaded " & $iLocationCount & " locations." & @CRLF)
    Return $aTempLocations
EndFunc   ;==>LoadLocations
```

```
Func SaveLocation()
    Global $hProcess, $PosXAddress, $PosYAddress
    Global $currentLocations, $maxLocations
    Global $aLocations  ; <<< Need this to reload

    Local $x = _ReadMemory($hProcess, $PosXAddress)
    Local $y = _ReadMemory($hProcess, $PosYAddress)
    ConsoleWrite("Attempting to read X: " & $x & " Y: " & $y & @CRLF)

    If @error Then
        ConsoleWrite("[Error] Failed to read memory. Error code: " & @error & @CRLF)
        Return
    EndIf
    If $x == 0 And $y == 0 Then
        ConsoleWrite("[Warning] Read zero for both coordinates. Possibly a bad read." & @CRLF)
        Return
    EndIf

    If Not FileExists($locationFile) Then
        Local $file = FileOpen($locationFile, $FO_CREATEPATH + $FO_OVERWRITE)
        If $file == -1 Then
            ConsoleWrite("[Error] Failed to create file: " & $locationFile & @CRLF)
            Return
        EndIf
        FileClose($file)
        ConsoleWrite("[Info] File created: " & $locationFile & @CRLF)
    EndIf

    Local $data = " : Location" & $currentLocations & "=X:" & $x & ";Y:" & $y & @CRLF
    If $currentLocations < $maxLocations Then
        _FileWriteLog($locationFile, $data)
        If @error Then
            ConsoleWrite("[Error] Failed to write to file: " & $locationFile & @CRLF)
        Else
            ConsoleWrite("[Info] Data written: " & $data)
            $currentLocations += 1

            ; ===== FIX: Reload locations after save =====
            $aLocations = LoadLocations()
            If @error Then
                ConsoleWrite("[Error] Failed to reload locations after save!" & @CRLF)
            Else
                ConsoleWrite("[Info] Locations reloaded successfully after save." & @CRLF)
            EndIf
        EndIf
    Else
        ConsoleWrite("[Info] Maximum locations reached. Stop pressing the button!" & @CRLF)
    EndIf
EndFunc   ;==>SaveLocation

Func EraseLocations()
    FileDelete($locationFile)
    $currentLocations = 1
    ConsoleWrite("Success - All locations erased." & @CRLF)
EndFunc   ;==>EraseLocations

; ------------------------------------------------------------------------------
```

```autoit
;                    LOCATION WALKING
; ---------------------------------------------------------------------------
Func MoveToLocations()
   Global $MoveToLocationsStatus, $hProcess, $PosXAddress, $PosYAddress, $iCurrentIndex, $aLocations

   If $MoveToLocationsStatus = 0 Then
      Local $currentX = _ReadMemory($hProcess, $PosXAddress)
      Local $currentY = _ReadMemory($hProcess, $PosYAddress)
      $iCurrentIndex = FindClosestLocationIndex($currentX, $currentY, $aLocations)

      If $iCurrentIndex = -1 Then
         ConsoleWrite("[Error] Could not find a closest location index (no valid data?)." & @CRLF)
         Return
      EndIf
      $MoveToLocationsStatus = 1
      ConsoleWrite("move on" & @CRLF)

   ElseIf $MoveToLocationsStatus = 1 Then
      $MoveToLocationsStatus = 0
      ConsoleWrite("move off" & @CRLF)

   Else
      MsgBox(0, "Error", "You shouldn't have gotten this error", 5)
   EndIf
EndFunc   ;==>MoveToLocations

Func MoveToLocationsStep($aLocations, ByRef $iCurrentIndex)
   Global $hProcess, $PosXAddress, $PosYAddress, $TypeAddress
   Global $WindowName, $lastX, $lastY
   Global $aTempBlocked[0][2], $ReverseLoopCheckbox
   Global $MoveToLocationsStatus ; <--- ADD THIS LINE to have access to the live status

   Static $lastMoveTime = TimerInit()
   Static $stuckCount = 0
   Static $lastTargetX = -1, $lastTargetY = -1

   ; EARLY EXIT: if toggled off during movement
   If $MoveToLocationsStatus = 0 Then Return SetError(1, 0, "Walker turned off mid-step")

   If Not IsArray($aLocations) Then Return SetError(2, 0, "Invalid input")
   If $iCurrentIndex < 0 Or $iCurrentIndex >= UBound($aLocations) Then Return SetError(3, 0, "Index out of range")

   Local $reverse = (GUICtrlRead($ReverseLoopCheckbox) = $GUI_CHECKED)
   Local $targetX = $aLocations[$iCurrentIndex][0]
   Local $targetY = $aLocations[$iCurrentIndex][1]

   If $lastTargetX <> $targetX Or $lastTargetY <> $targetY Then
      $stuckCount = 0
      $lastTargetX = $targetX
      $lastTargetY = $targetY
   EndIf

   If IsBlockedCoord($targetX, $targetY) Then
      ConsoleWrite("Skipping known blocked coordinate (" & $targetX & ", " & $targetY & ")" & @CRLF)
      $iCurrentIndex = NextIndex($iCurrentIndex, UBound($aLocations), $reverse)
      Return True
   EndIf
```

```
Local $currentX = _ReadMemory($hProcess, $PosXAddress)
Local $currentY = _ReadMemory($hProcess, $PosYAddress)
Local $Type = _ReadMemory($hProcess, $TypeAddress)

; EARLY EXIT: again before move attempt
If $MoveToLocationsStatus = 0 Then Return SetError(4, 0, "Walker turned off mid-step")

If $Type <> 65535 Then Return False

If $currentX = $lastX And $currentY = $lastY Then
   If TimerDiff($lastMoveTime) > 1000 Then
      ConsoleWrite("Detected stuck, trying bypass." & @CRLF)
      Local $beforeX = $currentX
      Local $beforeY = $currentY

      Local $bypassSuccess = TryBypass()

      $currentX = _ReadMemory($hProcess, $PosXAddress)
      $currentY = _ReadMemory($hProcess, $PosYAddress)

      If $bypassSuccess And ($currentX <> $beforeX Or $currentY <> $beforeY) Then
         ConsoleWrite("Bypass moved us away. Marking previous target blocked and skipping." & @CRLF)
         MarkCoordAsBlocked($lastTargetX, $lastTargetY)
         $iCurrentIndex = NextIndex($iCurrentIndex, UBound($aLocations), $reverse)
         $lastMoveTime = TimerInit()
         $lastX = $currentX
         $lastY = $currentY
         Return True
      Else
         $stuckCount += 1
         If $stuckCount >= 3 Then
            MarkCoordAsBlocked($targetX, $targetY)
            ConsoleWrite("Skipping stubborn target at (" & $targetX & ", " & $targetY & ")" & @CRLF)
            $iCurrentIndex = NextIndex($iCurrentIndex, UBound($aLocations), $reverse)
            $stuckCount = 0
            Return True
         EndIf
      EndIf
   EndIf
Else
   $lastMoveTime = TimerInit()
EndIf

$lastX = $currentX
$lastY = $currentY

If $currentX = $targetX And $currentY = $targetY Then
   ConsoleWrite("Arrived at target index: " & $iCurrentIndex & @CRLF)
   $iCurrentIndex = NextIndex($iCurrentIndex, UBound($aLocations), $reverse)
   Return True
EndIf

; FINAL EARLY EXIT check before sending any keys
If $MoveToLocationsStatus = 0 Then Return SetError(5, 0, "Walker turned off mid-step")

If $currentX < $targetX Then
   ControlSend($WindowName, "", "", "{d down}")
   Sleep(30)
```

```
        ControlSend($WindowName, "", "", "{d up}")
    ElseIf $currentX > $targetX Then
        ControlSend($WindowName, "", "", "{a down}")
        Sleep(30)
        ControlSend($WindowName, "", "", "{a up}")
    EndIf

    If $currentY < $targetY Then
        ControlSend($WindowName, "", "", "{s down}")
        Sleep(30)
        ControlSend($WindowName, "", "", "{s up}")
    ElseIf $currentY > $targetY Then
        ControlSend($WindowName, "", "", "{w down}")
        Sleep(30)
        ControlSend($WindowName, "", "", "{w up}")
    EndIf

    Return True
EndFunc   ;==>MoveToLocationsStep

Func NextIndex($iCurrent, $iBound, $reverse)
    If $reverse Then
        $iCurrent -= 1
        If $iCurrent < 0 Then $iCurrent = $iBound - 1
    Else
        $iCurrent += 1
        If $iCurrent >= $iBound Then $iCurrent = 0
    EndIf
    Return $iCurrent
EndFunc   ;==>NextIndex

Func QuickKey($key, $window, $hold)
    ControlSend($window, "", "", StringReplace($key, "}", " down}"))
    Sleep($hold)
    ControlSend($window, "", "", StringReplace($key, "}", " up}"))
EndFunc   ;==>QuickKey


Func TryBypass()
    Global $WindowName, $hProcess, $PosXAddress, $PosYAddress
    Global $lastX, $lastY, $aLocations, $iCurrentIndex

    Local $cx = _ReadMemory($hProcess, $PosXAddress)
    Local $cy = _ReadMemory($hProcess, $PosYAddress)
    Local $tx = $aLocations[$iCurrentIndex][0]
    Local $ty = $aLocations[$iCurrentIndex][1]

    Local $dx = $tx - $cx
    Local $dy = $ty - $cy

    Local $main = "", $side1 = "", $side2 = ""

    If Abs($dx) >= Abs($dy) Then
        If $dx < 0 Then
            $main = "{a}"
            $side1 = "{w}"
            $side2 = "{s}"
        Else
```

```
            $main = "{d}"
            $side1 = "{w}"
            $side2 = "{s}"
        EndIf
    Else
        If $dy < 0 Then
            $main = "{w}"
            $side1 = "{d}"
            $side2 = "{a}"
        Else
            $main = "{s}"
            $side1 = "{a}"
            $side2 = "{d}"
        EndIf
    EndIf

    Local $HoldTime = 75

    QuickKey($side1, $WindowName, $HoldTime)
    QuickKey($side1, $WindowName, $HoldTime)

    Local $nx = _ReadMemory($hProcess, $PosXAddress)
    Local $ny = _ReadMemory($hProcess, $PosYAddress)
    If $nx <> $cx Or $ny <> $cy Then
        ConsoleWrite("Bypass via " & $side1 & " worked. Resuming: " & $main & @CRLF)
        QuickKey($main, $WindowName, $HoldTime)
        $lastX = $nx
        $lastY = $ny
        Return True
    EndIf

    QuickKey($side2, $WindowName, $HoldTime)
    QuickKey($side2, $WindowName, $HoldTime)

    $nx = _ReadMemory($hProcess, $PosXAddress)
    $ny = _ReadMemory($hProcess, $PosYAddress)
    If $nx <> $cx Or $ny <> $cy Then
        ConsoleWrite("Bypass via " & $side2 & " worked. Resuming: " & $main & @CRLF)
        QuickKey($main, $WindowName, $HoldTime)
        $lastX = $nx
        $lastY = $ny
        Return True
    EndIf

    ConsoleWrite("Bypass failed: no movement after sidesteps." & @CRLF)
    Return False
EndFunc   ;==>TryBypass

Func FindClosestLocationIndex($currentX, $currentY, $aLocations)
    If Not IsArray($aLocations) Or UBound($aLocations, 0) = 0 Then
        ConsoleWrite("FindClosestLocationIndex => no valid array." & @CRLF)
        Return -1
    EndIf

    Local $minDist = 999999
    Local $minIndex = -1
    For $i = 0 To UBound($aLocations) - 1
        Local $dx = $currentX - $aLocations[$i][0]
        Local $dy = $currentY - $aLocations[$i][1]
        Local $dist = $dx * $dx + $dy * $dy
        If $dist < $minDist Then
            $minDist = $dist
            $minIndex = $i
        EndIf
    Next

    If $minIndex = -1 Then
        ConsoleWrite("FindClosestLocationIndex => No valid locations found." & @CRLF)
    Else
        ConsoleWrite("FindClosestLocationIndex => Found index: " & $minIndex & " Dist=" & $minDist & @CRLF)
    EndIf
    Return $minIndex
EndFunc   ;==>FindClosestLocationIndex

; ---------------------------------------------------------------------------
;                                  CURE FUNCTION
; ---------------------------------------------------------------------------
Func CureMe()
    Global $Chat, $Checkbox, $Sickness, $sicknessArray
    Global $HealDelay, $LastHealTime, $elapsedTimeSinceHeal
    Global $MovmentSlider, $PosXLabel, $PosYLabel

    If $Chat <> 0 Then Return

    ; Check if we have a sickness that is in the array
    If _ArraySearch($sicknessArray, $Sickness) = -1 Then Return

    Local $Healwait = GUICtrlRead($MovmentSlider)

    Local $currentX = Number(StringRegExpReplace(GUICtrlRead($PosXLabel), "[^\d]", ""))
    Local $currentY = Number(StringRegExpReplace(GUICtrlRead($PosYLabel), "[^\d]", ""))
    Static $lastX = $currentX, $lastY = $currentY
    Static $LastMovementTime = TimerInit()

    $elapsedTimeSinceHeal = TimerDiff($LastHealTime)

    ; Detect movement
    If $currentX <> $lastX Or $currentY <> $lastY Then
        $lastX = $currentX
        $lastY = $currentY
        $LastMovementTime = TimerInit()
    EndIf

    Local $TimeSinceLastMove = TimerDiff($LastMovementTime)

    ; Old style
    If GUICtrlRead($Checkbox) = $GUI_CHECKED Then
        If $elapsedTimeSinceHeal >= $HealDelay Then
            ControlSend("Project Rogue", "", "", "{3}")
            ConsoleWrite("Cure triggered (old style)" & @CRLF)
            $LastHealTime = TimerInit()
        EndIf
    Else
        If $elapsedTimeSinceHeal >= $HealDelay Then
            If $TimeSinceLastMove >= $Healwait Then
                ControlSend("Project Rogue", "", "", "{3}")
            ConsoleWrite("Cure triggered: Stationary for " & $TimeSinceLastMove & "ms." & @CRLF)
            $LastHealTime = TimerInit()
        Else
            ConsoleWrite("No cure: Only stationary for " & $TimeSinceLastMove & "ms." & @CRLF)
        EndIf
    EndIf
  EndIf
EndFunc   ;==>CureMe

; ----------------------------------------------------------------------------
;                               HEALER
; ----------------------------------------------------------------------------
Func TimeToHeal()
    Global $MovmentSlider, $PosXLabel, $PosYLabel, $Checkbox, $HPAddress, $MaxHPAddress
    Global $HealerLabel, $HealDelay, $LastHealTime, $elapsedTimeSinceHeal, $sicknessArray, $Sickness
    Global $Chat, $ChattOpenAddress, $healSlider
    Global $hProcess

    Local $Healwait = GUICtrlRead($MovmentSlider)
    Local $HP = _ReadMemory($hProcess, $HPAddress)
    Local $RealHP = $HP / 65536
    Local $MaxHP = _ReadMemory($hProcess, $MaxHPAddress)
    Local $ChatVal = _ReadMemory($hProcess, $ChattOpenAddress)
    Local $HealThreshold = GUICtrlRead($healSlider) / 100

    Local $currentX = Number(StringRegExpReplace(GUICtrlRead($PosXLabel), "[^\d]", ""))
    Local $currentY = Number(StringRegExpReplace(GUICtrlRead($PosYLabel), "[^\d]", ""))
    Static $lastX = $currentX, $lastY = $currentY
    Static $LastMovementTime = TimerInit()

    $elapsedTimeSinceHeal = TimerDiff($LastHealTime)

    ; --- Detect movement ---
    If $currentX <> $lastX Or $currentY <> $lastY Then
        $lastX = $currentX
        $lastY = $currentY
        $LastMovementTime = TimerInit()
    EndIf

    Local $TimeSinceLastMove = TimerDiff($LastMovementTime)

    ; --- Old style (checkbox) ---
    If GUICtrlRead($Checkbox) = $GUI_CHECKED Then
        If $ChatVal = 0 And _ArraySearch($sicknessArray, $Sickness) = -1 Then
            If $RealHP < ($MaxHP * $HealThreshold) And $elapsedTimeSinceHeal > $HealDelay Then
                ControlSend("Project Rogue", "", "", "{2}")
                ConsoleWrite("Heal triggered (old style): HP < threshold" & @CRLF)
                $LastHealTime = TimerInit()
            EndIf
        EndIf
    Else
        ; --- Normal logic (requires stationary) ---
        If $ChatVal = 0 And _ArraySearch($sicknessArray, $Sickness) = -1 Then
            If $RealHP < ($MaxHP * $HealThreshold) And $elapsedTimeSinceHeal > $HealDelay Then
                If $TimeSinceLastMove >= $Healwait Then
                    ControlSend("Project Rogue", "", "", "{2}")
                    ConsoleWrite("Healed: Stationary for " & $TimeSinceLastMove & "ms | HP < threshold." & @CRLF)
                    $LastHealTime = TimerInit()
            Else
                ConsoleWrite("No heal: Only stationary for " & $TimeSinceLastMove & "ms." & @CRLF)
            EndIf
        EndIf
    EndIf
EndIf
EndFunc   ;==>TimeToHeal

; ----------------------------------------------------------------------------
;                                 TARGETING
; ----------------------------------------------------------------------------
Func AttackModeReader()
    Global $hProcess, $WindowName
    Global $Type, $Chat, $AttackMode
    Global $PosXAddress, $PosYAddress
    Global $LootingCheckbox, $TargetStatus
    Global $LootQueued, $LootCount, $LootReady
    Global $LastPlayerX, $LastPlayerY
    Global $HadTarget, $LastTargetHeld
    Global $currentTime, $TargetDelay
    Global $LootIdleTimer, $LootIdleWaiting

    $Chat = _ReadMemory($hProcess, $ChattOpenAddress)
    $Type = _ReadMemory($hProcess, $TypeAddress)
    $AttackMode = _ReadMemory($hProcess, $AttackModeAddress)

    Local $PlayerX = _ReadMemory($hProcess, $PosXAddress)
    Local $PlayerY = _ReadMemory($hProcess, $PosYAddress)

    ; Cancel loot if player moves
    If $LastPlayerX <> 0 And $LastPlayerY <> 0 Then
        If $PlayerX <> $LastPlayerX Or $PlayerY <> $LastPlayerY Then
            ConsoleWrite("[Loot] Player moved manually, cancelling loot queue." & @CRLF)
            $LootQueued = False
            $LootCount = 0
            $LootReady = False
            $LootIdleWaiting = False
        EndIf
    EndIf

    $LastPlayerX = $PlayerX
    $LastPlayerY = $PlayerY

    ; --- Loot kill detection ---
    If GUICtrlRead($LootingCheckbox) = $GUI_CHECKED Then
        If $Type = 1 Then ; Monster targeted
            If Not $HadTarget Then
                $HadTarget = True
                $LastTargetHeld = TimerInit()

                ; If new monster targeted, cancel loot idle wait
                If $LootIdleWaiting Then
                    ConsoleWrite("[Loot] New target acquired. Cancelling idle wait." & @CRLF)
                    $LootIdleWaiting = False
                EndIf
            ElseIf TimerDiff($LastTargetHeld) >= 100 Then
                ; Held >100ms, stable target
            EndIf
        ElseIf $Type = 65535 Then ; No target (possible kill)
            If $HadTarget Then
                If TimerDiff($LastTargetHeld) >= 100 Then
                    $LootCount += 1
                    $LootQueued = True
                    ConsoleWrite("[Loot] Monster kill detected. Loot count now: " & $LootCount & @CRLF)
                EndIf
                $HadTarget = False
                $LootIdleTimer = TimerInit()
                $LootIdleWaiting = True
            EndIf
        EndIf
    EndIf

    ; --- Targeter Retarget ---
    If $TargetStatus = 1 And $Type = 65535 And $Chat = 0 Then
        If TimerDiff($currentTime) >= $TargetDelay Then
            ControlSend($WindowName, "", "", "{TAB}")
            ConsoleWrite("[Target] Retargeting with TAB..." & @CRLF)
            $currentTime = TimerInit()
        EndIf
    EndIf
EndFunc   ;==>AttackModeReader


Func IsBlockedCoord($x, $y)
    For $i = 0 To UBound($aTempBlocked) - 1
        If $aTempBlocked[$i][0] = $x And $aTempBlocked[$i][1] = $y Then
            Return True
        EndIf
    Next
    Return False
EndFunc   ;==>IsBlockedCoord

Func MarkCoordAsBlocked($x, $y)
    ReDim $aTempBlocked[UBound($aTempBlocked) + 1][2]
    $aTempBlocked[UBound($aTempBlocked) - 1][0] = $x
    $aTempBlocked[UBound($aTempBlocked) - 1][1] = $y
    ConsoleWrite("Marked (" & $x & ", " & $y & ") as blocked." & @CRLF)
EndFunc   ;==>MarkCoordAsBlocked

Func _WriteMemory($hProc, $pAddress, $value)
    Local $tBuffer = DllStructCreate("dword")
    DllStructSetData($tBuffer, 1, $value)
    DllCall("kernel32.dll", "bool", "WriteProcessMemory", _
            "handle", $hProc, _
            "ptr", $pAddress, _
            "ptr", DllStructGetPtr($tBuffer), _
            "dword", DllStructGetSize($tBuffer), _
            "ptr", 0)
EndFunc   ;==>_WriteMemory
```