Dokumentation zu ElderAnfängerSRM

InOMatrix,
Henker von ElderCraft

17. März 2021

Inhaltsverzeichnis

---

1 Einleitung

Dieses Programm hilft dabei, das Setzen von Verkaufsschildern auf Anfängergrundstücken zu erleichtern. Dies geschieht dadurch, dass aus einer gegebenen Liste von Grundstücken dessen Teleport-Befehle in die Zwischenablage kopiert werden. Außerdem stellt das Programm eine Art Makro für die Zeilen, die auf die Verkaufsschilder geschrieben werden, zur Verfügung. Wie genau das funktioniert, wird im Benutzerhandbuch erklärt.

Dieses Dokument beschreibt lediglich, wie das Programm technisch aufgebaut ist, indem alle Funktionen genau erklärt werden. Insgesamt ist ElderAnfängerSRM in drei Komponenten aufgeteilt:

- `elderanfaengersrm.py` – das Hauptmodul;
- `terminalausgaben.py` – Modul für alle Ausgaben auf der Konsole;
- `terminalausgaben.ini` – Konfigurationsdatei, hier können die Ausgaben verändert werden.

Diese Dokumentation bezieht sich einzig und allein auf die beiden Python-Module. Wie die Konfigurationsdatei bearbeitet werden kann, steht im Benutzerhandbuch.

2 Dokumentation des Moduls `elderanfaengersrm.py`

Dieses ist das Hauptmodul, welches beim Programmstart aufgerufen wird. In dem Modul ist eine `main()`-Methode implementiert. Wird das Programm gestartet, werden zuerst die Ausgaben auf der Konsole konfiguriert (siehe hierzu `konfigurieren()`, und anschließend die main-Methode ausgeführt.

2.1 Importe externer Bibliotheken

In der nachfolgenden Liste stehen die Bibliotheken, die im Modul importiert werden und, wofür sie gebraucht werden:

- `time` – Bestimmen der Reaktionszeit, nachdem das Programm in der Konsole nach dem nächsten Befehl fragt. Das verhindert, dass ungewollte Eingaben in der Konsole verarbeitet werden, sollte das Programm gerade nicht nach einem Befehl fragen. Das kommt vor, wenn sich das Fenster öffnet, um Grundstücksdaten dort hineinzuschreiben.
- `pynput` – Manipuliert die Tastatureingabe. Es wird für die Makros '*[verkaufen]*' und '*Building-Dave*' verwendet. Durch Drücken der Einfg-Taste werden diese Worte unter gewissen Umständen geschrieben.
- `tkinter` – Modul zum Öffnen und Erstellen des Fensters, in das die Grundstücksdaten hineinkopiert werden.
- `pyperclip` – Manipuliert die Zwischenablage. Durch Drücken der Einfg-Taste wird unter gewissen Umständen der nächste Teleport-Befehl in die Zwischenablage kopiert. Auch, nachdem der Befehl '*weiter*' eingegeben wird, wird die Zwischenablage verändert. Nachdem alle Grundstücke aus der Liste abgearbeitet sind, wird die Zwischenablage geleert.
- `threading` – Modul zum Erstellen von Threads. Im Main-Thread werden die Befehle in der Konsole entgegen genommen. Sollte der Befehl '*start*' eingegeben worden sein, wird das Programm, welches bei Betätigen der Einfg-Taste Teleport-Befehle kopiert bzw. Schildzeilen schreibt, in einem separaten Thread laufen.
- `webbrowser` – Modul zum Öffnen von Internetadressen in einem Browser. Das wird genutzt, wenn man den Befehl '*link öffnen*' eingibt.
- `terminalsausgaben` – Modul, dass die Ausgaben auf der Konsole formatiert.
- `win10toast` – Modul zum Erzeugen der Windows-Notification. Diese erscheint immer, wenn ein Teleport-Befehl in die Zwischenablage kopiert wird. Sie erscheint auch am Ende, wenn alle Grundstücke abgearbeitet sind.

2.2 Globale Variablen

Insgesamt verfügt das Python-Modul `elderanfaengersrm.py` über sieben globale Variablen:

`pause` *(bool)* Der Wert gibt an, ob das Programm (der Thread), das die Schildzeilen schreibt und Teleport-Befehle kopiert, pausiert ist.

`listener` *(pynput.keyboard.Listener)* Ein weiterer Thread, der auf Tastenaschläge reagiert. Im Programm macht der Thread nichts anderes, als auf die Einfg-Taste zu reagieren.

`kopier_thread` *(threading.Thread)* Der Thread, in dem das Schreiben der Schildzeilen bzw. Kopieren der Teleport-Befehle vonstatten geht.

`grundstuecke` *(list)* Die Liste aller Grundstücke, die in dem Fenster eingegeben wurde. Dabei enthält die Liste einfach jede nichtleere Zeile der Eingabe als Listenelement.

`index` *(int)* Der Wert gibt den Index aus `grundstuecke` an, dessen Grundstück gerade bearbeitet wird.

`stop` *(bool)* Der Wert gibt an, ob der Thread, das die Schildzeilen schreibt usw. gestoppt werden soll bzw. gestoppt ist.

`interne_clipboard` *(str)* Der Wert wird mit dem Teleport-Befehl des aktuellen Grundstückes, an dem gearbeitet wird, überschrieben, sofern der Befehl '*pause*' eingegeben wird. Der Inhalt wird in die Zwischenablage kopiert, wird der Befehl '*weiter*' eingegeben.

2.3 Hauptfunktionen

`befehl_link_oeffnen()`

Diese Funktion öffnet den Link git.eldercom.de in einem Browser und gibt eine Nachricht in der Konsole aus, dass der Link geöffnet wird.

Input: -
Returns: *None*

`befehl_grundstuecke_eingeben()`

Diese Funktion öffnet ein Fenster, in das Zeilen aus der `worldguard.csv`-Datei (von git.eldercom.de) hineinkopiert werden können. In diesem Fenster gibt es die folgenden

Steuerelemente:

`textfeld` *(Text)* Das Textfeld, in das hineingeschrieben werden kann. Dort werden die Zeilen der zu bearbeitenden Grundstücke hineinkopiert.

`scrollbar` *(Scrollbar)* Eine Scrollbar, um in dem Textfeld nach oben und nach unten zu scrollen.

`speichern` *(Button)* Der '*Speichern und Beenden*'-Button. Durch einen Klick darauf wird die Funktion `speichern_beenden(fenster, textfeld)` aufgerufen.

`umkehren` *(Button)* Der '*Reihenfolge umkehren*'-Button. Durch einen Klick darauf wird die Funktion `reihenfolge_umkehren(textfeld)` aufgerufen.

`updaten` *(Button)* Der '*Länge aktualisieren*'-Button. Durch einen Klick darauf wird die Funktion `update_label(anzahl, textfeld)` aufgerufen.

`anzahl` *(Label)* Das Label, dass die Anzahl an nichtleeren Zeilen in dem Textfeld angibt. Es gibt auch an, ob die Eingabe im Textfeld syntaktisch in Ordnung ist. Damit das Label aktualisiert wird, muss auf den entsprechenden Button geklickt werden.

Input: -
Returns: *None*

Sobald das Fenster geöffnet bzw. geschlossen wird, erscheint dazu jeweils eine passende Nachricht in der Konsole.

`befehl_start()`

Diese Funktion startet den Thread `kopier_thread`, der die Teleport-Befehle kopiert und Schildzeilen schreibt. Der Thread arbeitet die Funktion `kopieren_und_drucken()` ab.

Input: -
Returns: *None*

Zunächst wird überprüft, ob die Liste mit den Grundstücken `grundstuecke` syntaktisch korrekt ist. Falls nicht, wird der Thread nicht gestartet und es gibt eine passende Fehlerausgabe. Anschließend wird überprüft, ob der Thread nicht schon gestartet wurde (das heißt bereits läuft) oder die Liste der Grundstücke keine Einträge besitzt. In jedem der Fälle wird eine passende Ausgabe in der Konsole erscheinen und nichts weiter passieren. Die Abfrage, ob der Thread bereits läuft, geschieht über die globale Variable `stop`.

Trifft keiner der obigen Fälle zu, wird der Thread gestartet. Dabei werden die folgenden globalen Variablen gesetzt:

```
stop  = False
pause = False
index = 0
```

Es gibt dabei Ausgaben im Terminal, wenn der Thread gestartet wird und ob er erfolgreich gestartet wurde.

`befehl_pause()`

Diese Funktion setzt die globale Variable `pause` auf `True` und speichert den aktuellen Inhalt der Zwischenablage intern ab in `interne_clipboard`.

Input: -
Returns: *None*

Die Variable `pause` wird in `on_release(key)` insbesondere abgefragt und solange sie auf `True` steht, gibt sie auch nicht den Wert `False` zurück. Sollte `pause` bereits auf `True` stehen, wird in der Konsole ausgegeben, dass das Programm bereits pausiert ist. Sollte `stop = True` sein, das heißt der Thread, der die Teleport-Befehle kopiert usw. gar nicht existieren und laufen, wird ebenfalls eine Nachricht ausgegeben, dass es nichts zu pausieren gibt.

`befehl_weiter()`

Diese Funktion setzt die globale Variable `pause` auf `False` und legt den Inhalt von `interne_clipboard` in die Zwischenablage.

Input: -
Returns: *None*

Sollte `pause` bereits auf `False` stehen, wird in der Konsole ausgegeben, dass das Programm nicht pausiert war. Sollte `stop = True` sein, das heißt der Thread, der die Teleport-Befehle kopiert usw. gar nicht existieren und laufen, wird ebenfalls eine Nachricht ausgegeben, dass nichts weiterlaufen kann.

`befehl_status()`

Diese Funktion gibt in der Konsole die gesamte Liste der Grundstücke aus und die Zeilen werden markiert, damit man sieht, welche bereits abgehakt sind und welche nicht.

Input: -
Returns: *None*

Sollte `stop = True` sein, das heißt der Thread, der das Kopieren und Schreiben von Schildzeilen managt gar nicht laufen bzw. existieren, wird eine Meldung in der Konsole erscheinen, dass es keinen Status gibt. Andernfalls werden alle Einträge aus `grundstuecke` ausgegeben; jeweils eine eigene Zeile pro Listeneintrag. Alle Einträge von `0` bis `index-1` werden dunkelgrau ausgegeben, der Eintrag am Index `index` wird in grün ausgegeben und alle weiteren Einträge in der Liste werden in normaler Schriftfarbe dargestellt.

Des Weiteren erscheint die Anzahl der Grundstücke, die noch bearbeitet werden müssen und der entsprechende prozentuale Anteil.

`befehl_stop()`

Diese Funktion stoppt den Thread, der die Teleport-Befehle kopiert und Schildzeilen schreibt.

Input: -
Returns: *None*

Sollte `stop = True` sein, wird eine Nachricht in der Konsole erscheinen, dass das Programm bereits gestoppt ist. Andernfalls wird `stop = False` gesetzt und der Thread `listener`, der auf die Tastenanschläge reagiert, gestoppt. Anschließend wird gewartet, dass auch der Thread `kopier_thread` geschlossen wird. Dann wird die Liste der Grundstücke, also `grundstuecke`, geleert, die Zwischenablage ebenfalls.

Zusätzlich erscheinen Ausgaben auf dem Terminal, wenn das Programm gestoppt wird und erfolgreich gestoppt wurde.

`befehl_exit()`

Diese Funktion beendet alle laufenden Threads und schließt das Programm `ElderAnfaengerSRM` vollständig.

Input: -
Returns: *None*

Sollte `stop = False` sein, das heißt der Thread, in dem das Kopieren der Port-Befehle und Schreiben der Schildzeilen gemanagt wird noch laufen, so wird er beendet, indem `stop = True` gesetzt wird, `listener` gestoppt wird und auf `kopier_thread` gewartet wird, bis er beendet ist.

`befehl_hilfe()`

Diese Funktion gibt die Hilfetabelle mit allen verfügbaren Befehlen in der Konsole aus.

Input: -

Returns: *None*

2.4 Hilfsfunktionen

`on_release(key)`

Diese Funktion liefert die Rückgabe `False`, wenn die Einfg-Taste auf der Tastatur freigelassen wird.

Input:
`key` *(pynput.keyboard.Key)* Die Taste, die losgelassen wird.
Output:
(bool) False, wenn die Einfg-Taste losgelassen wird und `pause = False` ist. In jedem anderen Fall gibt die Funktion gar keinen Wert zurück.

`wait_einfg()`

Diese Funktion „wartet“ darauf, dass die Taste Einfg gedrückt und losgelassen wird. Sobald dies geschehen ist, endet die Funktion ohne einen Rückgabe-Parameter.

Input: -
Returns: *None*

Wird diese Funktion aufgerufen, wird die globale Variable `listener` definiert mit dem Parameter `on_release = on_release`. Sobald die Funktion `on_release(key)` also den Wert `False` liefert, wird `listener` beendet und die Funktion wird verlassen. Das gleiche passiert auch, wenn `listener` von einer Funktion außerhalb gestoppt wird.

`windows_notification(gs)`

Diese Funktion erstellt und gibt eine Windows-Benachrichtigung aus, dass ein Teleport-Befehl in die Zwischenablage kopiert wurde. In der Benachrichtigung steht ebenfalls die dazugehörige Grundstücks-ID.

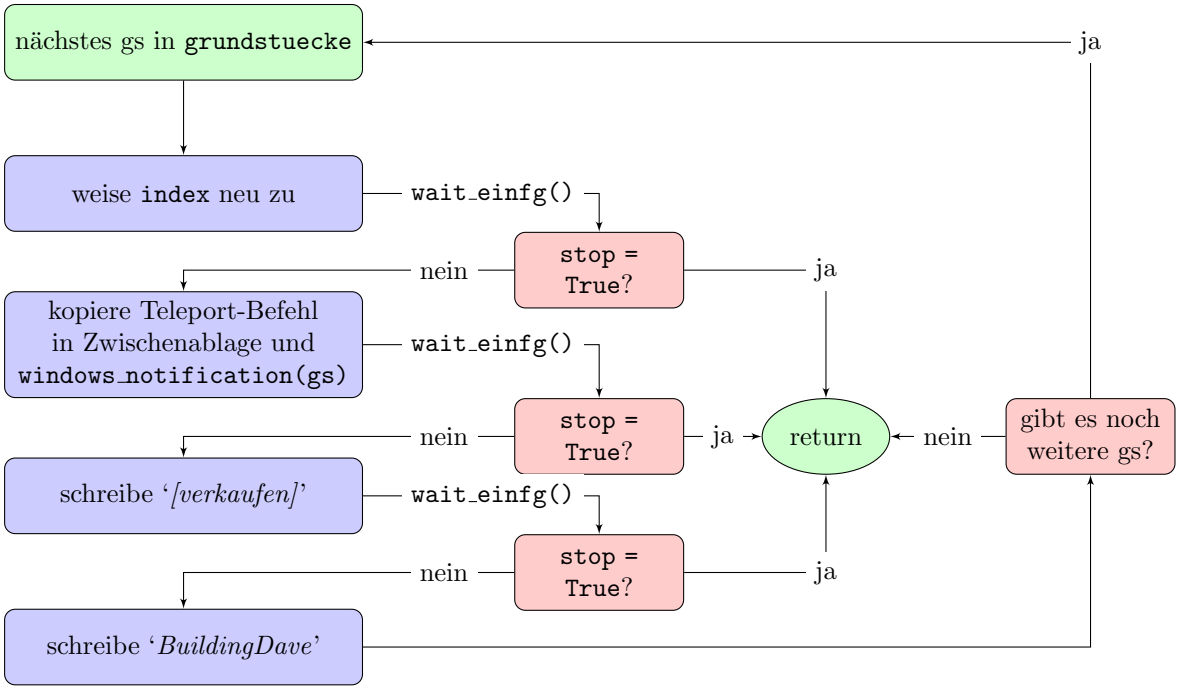
Input:
`gs` *(str)* Die Zeile aus `grundstuecke`, zu der die Benachrichtigung erscheint.
Returns: *None*

`kopieren_und_drucken()`

Diese Funktion geht jedes Grundstück (also jeden Eintrag) aus `grundstuecke` durch und kopiert den Teleport-Befehl in die Zwischenablage und schreibt Schildzeilen, sofern die Einfg-Taste betätigt wird.

Input: -
Returns: *None*

Der Ablauf des Programms lässt sich durch das folgende Diagramm veranschaulichen, wobei es im grünen Rechteck startet:



`grundstueck_ok(gs)`

Diese Funktion untersucht, ob ein String dem syntaktischen Aufbau einer Zeile aus `worldguard.csv` entspricht. Die allgemeine Syntax lautet '`"anf-XXX-XXX","Y","Y","Y","Y","Y","Y","/tp Y Z Y"`', wobei für jedes `X` eine beliebige Ziffer, für jedes `Y` eine beliebige ganze Zahl und für das `Z` eine beliebige positive Zahl eingesetzt wird.

Input:
`gs` *(str)* Die Zeichenfolge, die untersucht wird, ob sie dem obigen Aufbau entspricht.
Output:
(bool) True genau dann, wenn `gs` syntaktisch dem Aufbau entspricht.

`update_label(anzahl, textfeld)`

Diese Funktion aktualisiert das Label, das die Anzahl an Zeilen in dem Textfeld angibt. Es zeigt auch an, falls es syntaktische Fehler in der Liste gibt.

Input:
`anzahl` *(tkinter.Label)* Das Label, dass die Zeilenanzahl angibt.
`textfeld` *(tkinter.Text)* Das Textfeld, in dem der Text steht der auf Zeilenanzahl überprüft wird.
Output: *None*

Der Inhalt aus dem Textfeld wird ausgelesen und in einer Liste gespeichert, die jede Zeile aus dem Textfeld als Listeneintrag enthält. Nun wird jeder Eintrag aus der Liste überprüft mithilfe der Funktion `grundstueck_ok(gs)`:

```
for zeile in textfeld:
    if not (zeile == "" or grundstuecke_ok(zeile)):
```

```
        setze schriftfarbe der zeile rot
    else:
        setze schriftfarbe der zeile weiss
```

Jede syntaktisch fehlerhafte Zeile wird also im Textfeld rot markiert. Sollte es eine solche Zeile geben, steht im Label in rot, dass die Texteingabe Fehler enthält. Sollte es keine rot markierte Zeile geben, steht im Label die Anzahl der nichtleeren Zeilen.

`reihenfolge_umkehren(textfeld)`

Diese Funktion kehrt die Reihenfolge der Zeilen der Eingabe im Textfeld um.

Input:
`textfeld` *(tkinter.Text)* Das Textfeld, in dem der Inhalt steht.
Output: *None*

Der Inhalt aus dem Textfeld wird ausgelesen und in einer Liste zwischengespeichert, die jede Zeile aus dem Textfeld als Listeneintrag enthält. Leerzeilen werden dabei ignoriert, also am Ende gar nicht mehr mit ausgegeben. Anschließend wird der Inhalt des Textfeldes gelöscht. Dann wird der Inhalt neu gesetzt, indem die Liste rückwärts abgelaufen wird und für jeden einzelnen Eintrag eine neue Zeile schreibt.

`speichern_beenden(fenster, textfeld)`

Diese Funktion speichert die Eingabe eines Textes aus einem Fenster und beendet es.

Input:
`fenster` *(tkinter.Tk)* Das Fenster, welches nach Speichern des Textes aus dem Textfeld geschlossen wird.
`textfeld` *(tkinter.Text)* Das Textfeld, dessen Inhalt gespeichert werden soll.
Output: *None*

Der Inhalt wird in die globale Variable `grundstuecke` gespeichert. Jede Zeile des Inhaltes spiegelt dabei einen Listeneintrag wider. Leerzeilen werden allerdings ignoriert.

3 Dokumentation des Moduls terminalausgaben.py

3.1 Importe externer Bibliotheken

3.2 Globale Variablen

3.3 Funktionen zum Lesen der Konfigurationsdatei

konfigurieren()

to_color_code(farbeingabe)

3.4 Feste Ausgaben

print_hrule()

print_befehl_nur_nach_aufforderung()

print_befehl_invalide_gebe_hilfe_ein()

print_programm_wird_beendet()

print_programm_ist_beendet()

3.5 Variable Ausgaben

print_fehler_nachricht(plugin, schluessel, args=None)

print_terminal_nachricht(plugin, schluessel, args=None)

print_hilfe(plugin)

print_beschreibung(plugin)

3.6 Funktionen zur Ausgabe und Erstellung von Tabellen

erstelle_tabelle(text)

get_umbrueche_index(text, max$_laenge$)

setze_farbcodes(text, standardfarbe=Fore.RESET)

entferne_farbcodes(text)