Texteingabe in GMS

Description

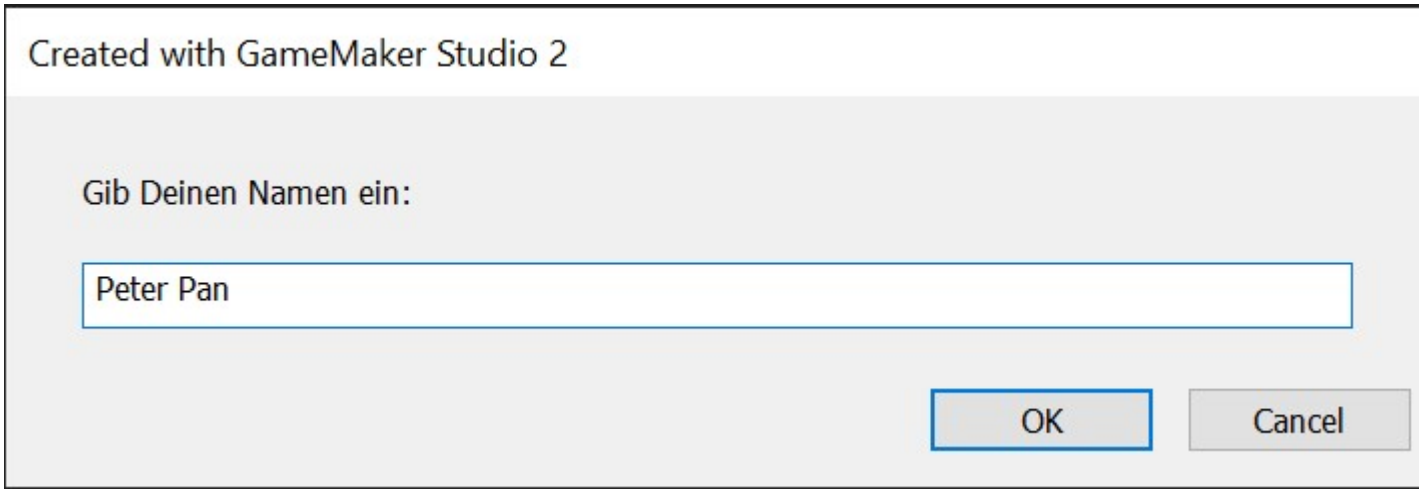
Eines der Schwierigkeiten, vor denen Anfänger in GMS stehen, ist die Texteingabe. Etwa für einen Namen. Dieses Problem lässt sich mit einigen Zeilen Code lösen.

Der ganz billige Weg

Mit der Funktion *get_string()* kann man in GMS ein Fenster anzeigen lassen. Etwa so:

```
txt_input = get_string("Gib Deinen Namen ein:", „Peter Pan");
```

Daraufhin erscheint eine Box, die so aussieht:



get_string

Damit wären wir schon fertig. Der eingegebene Text wird in der Variable *txt_input* gespeichert. Doch wir haben gleich drei Probleme:

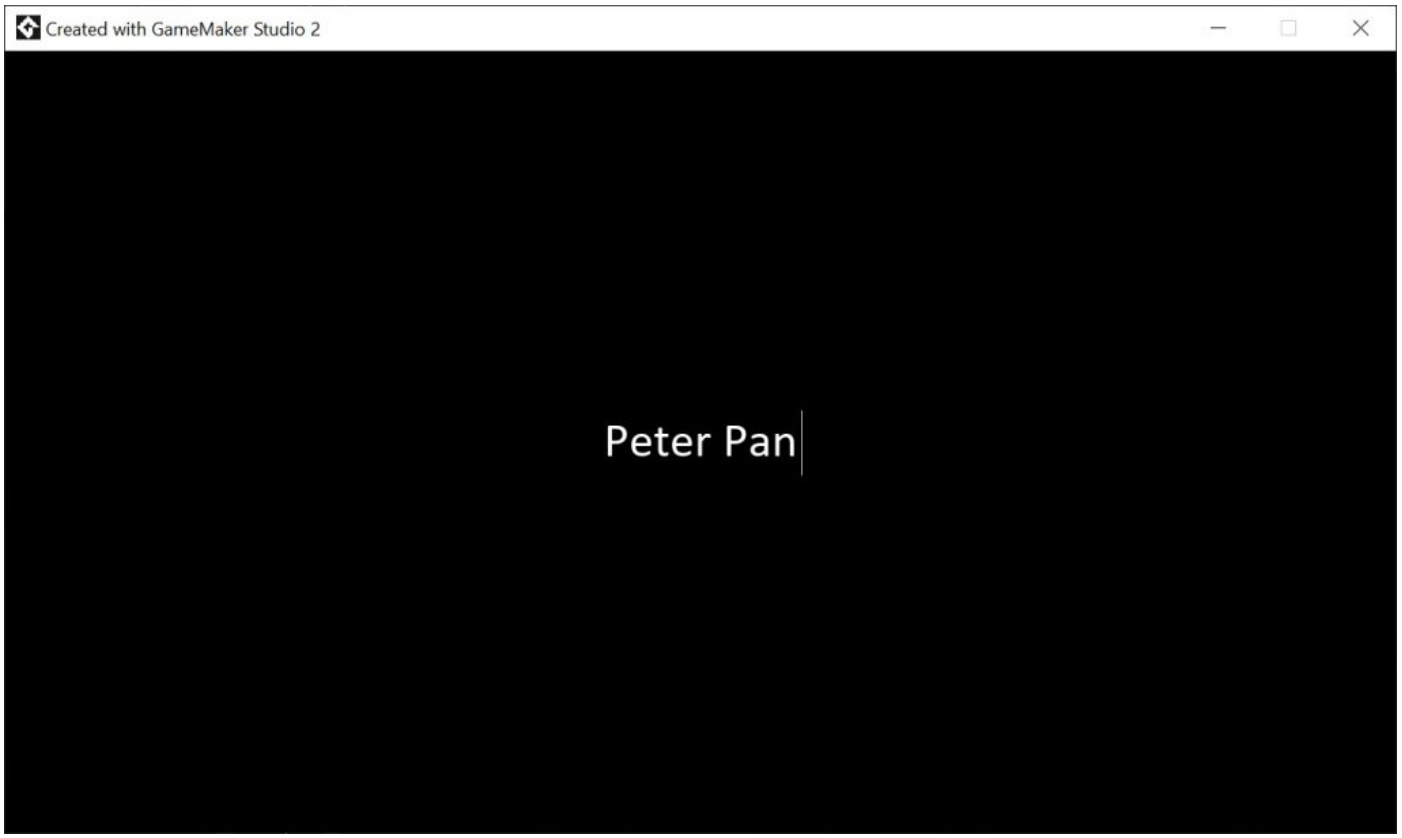
1. Wir können mit dieser Methode die erlaubten Zeichen nicht reduzieren.
2. Wir können min und max der erlaubten Zeichen nicht anpassen.

3. Das sieht ziemlich billig aus und passt mit Sicherheit nicht zum Spiel.

Was wir wollen, ist eine „richtige” Eingabe im Spiel. Die lässt sich dann beliebig designen, etwa mit einem Fenster drum herum. Um es einfach zu halten, hat dieses Beispiel folgende Funktionen:

- Der Spieler kann einen Text eingeben.
- Die erlaubten Zeichen bestehen nur aus Groß- und Kleinbuchstaben.
- Doppelte Leerzeichen werden entfernt.
- Ein blinkender Cursor befindet sich hinter dem Text.

Am Ende soll es so aussehen:



Texteingabe in GMS

Ausgangslage

Für dieses Beispiel legen wir ein neues Projekt an. Mein **Objekt** heißt *o_input*. Die **Schrift** habe ich *f_txt_input* genannt. Dahinter verbirgt sich die Schriftart Calibri in der Größe 36. Mehr brauchen wir dieses Mal nicht.

o_input

Event Create

```
keyboard_string = „";
```

```
txt_input  = „";
dtime   = 7;
delete_timer = dtime;
txt_limit  = 16;
txt_min   = 3;
enable_keys  = „ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz „;
blink_speed  = 30;
blink   = true;
alarm[0]  = blink_speed;
```

keyboard_string ist eine von Game Maker vorgegebene Variable. Sie enthält eine Zeichenkette mit den letzten (maximal) 1024 Zeichen, die auf der Tastatur eingegeben wurden. Diese Zeichenkette enthält nur druckbare Zeichen, aber sie reagiert korrekt auf das Drücken der Rücktaste, indem sie das letzte Zeichen löscht. Bevor wir also starten, wird diese gelöscht.

txt_input ist unser eingegebener Text. Der kann später weiterverarbeitet werden. Im Beispiel erscheint lediglich eine Textbox.

Die Vatiablen *dtime* und *delete_timer* nutzen wir für die Verzögerung, falls der Text oder einzelne Buchstaben mit der Backspace-Taste gelöscht werden sollen.

txt_limit gibt die maximale Anzahl an Zeichen an. *txt_min* entsprechend die minimale Anzahl. Im Beispiel ist die Eingabe erst korrekt, wenn sie mindestens 3, aber höchstens 16 Zeichen hat.

Mit *enable_keys* geben wir vor, welche Zeichen erlaubt sind. Im Beispiel sind es Groß- und Kleinbuchstaben, Zahlen sind nicht erlaubt. Das kann man aber beliebig ergänzen.

blink_speed und *blink* steuern den Cursor. Zuletzt wird der Alarm gestartet, damit dieser laufend an und aus geht.

Event Alarm[0]

```
blink  = !blink;
alarm[0] = blink_speed;
```

Hier schalten wir das Blinken an und aus.

Event Step

```
// Texteingabe
if (keyboard_check(vk_anykey)) && (string_length(txt_input) < txt_limit)
{
 if (string_count(string(keyboard_string), enable_keys) == 1)
 {
  blink   = true;
  alarm[0]  = blink_speed;
  txt_input  = txt_input + string(keyboard_string);
  keyboard_string = „";
 } else {
  keyboard_string = „";
 }
```

```
}

// Backspace
if (keyboard_check(vk_backspace)) && (!keyboard_check_pressed(vk_backspace)) &
{
 txt_input  = string_delete(txt_input, string_length(txt_input), 1);
 delete_timer = 0;
 keyboard_string = „";
}

// Doppelte Leerzeichen löschen
txt_input = string_replace(txt_input, „  „, „ „);

// Enter
if (keyboard_check_pressed(vk_enter) && (string_length(txt_input) >=txt_min))
{
 show_message(txt_input);
}

// Counter für die Zeit zum löschen
if (delete_timer < dtime)
{
 delete_timer++;
}
```

Der Code ist in fünf Abschnitte unterteilt. Schauen wir uns das einzeln an.

Texteingabe

Zunächst fragen wir ab, ob eine Taste gedrückt wurde (*vk_anykey*) und ob der Text kürzer ist als *txt_limit*. Wenn dem so ist, kommt die nächste Abfrage. Hier schauen wir nach, ob das eingegebene Zeichen (*keyboard_string*) einem der erlaubten Zeichen (*enable_keys*) entspricht. Wenn das so ist, Schalten wir den das Blinken immer ein und ergänzen die Variable *txt_input* um das neue Zeichen. Anschließend wird *keyboard_string* zurückgesetzt.

Backspace

Wir schauen, ob die Backspace-Taste gedrückt und gehalten wird. D. h. der Spieler kann den Text mit gedrückter Taste löschen. Hinzu kommt die Abfrage von *delete_timer*.

Ist das erfüllt, wird mit *string_delete(txt_input, string_length(txt_input), 1)* das letzte Zeichen der Variable *txt_input* gelöscht. Anschließend setzen wir *delete_timer* auf 0 und löschen *keyboard_string*.

Doppelte Leerzeichen löschen

Für solche Aufgaben ist die Funktion *string_replace* perfekt.

Enter

Neben dem Tastendruck fragen wir noch ab, ob die minimale Zahl der Zeichen erreicht wurde. Ab dann können wir alles mögliche machen. Im Beispiel zeigen wir den Text noch einmal an. Man könnte auch ein Formular mit mehreren Eingaben machen und somit zum nächsten Feld springen. Oder in den nächsten Raum.

Counter für die Zeit zum löschen

Das ist selbstredend. Wir zählen den Counter für die Verzögerung hoch.

Event Draw

```
draw_set_color(#FFFFFF);
draw_set_alpha(1);
draw_set_font(f_txt_input);
draw_set_halign(fa_center);
draw_set_valign(fa_middle);
draw_text(room_width/2, room_height/2, txt_input);

// Blink Cursor
if (blink)
{
 var lenght = string_width(txt_input);
 var strLenght = string_length(txt_input);
 var offsetX = lenght/2+4;
 draw_set_color(#CCCCCC);

 if (strLenght <= 0)
 {
  draw_line(room_width/2, room_height/2-32, room_width/2, room_height/2+32);
 } else {
  draw_line(room_width/2+offsetX, room_height/2-32, room_width/2+offsetX, room
 }
}
```

Das Anzeigen des Textes ist einfach. Nachdem die ersten fünf Zeilen lediglich Formatierungen sind, wird in Zeile 6 der Text angezeigt. Der Cursor ist schon ein bisschen komplexer.

Wenn *blink* wahr ist, wird es angezeigt. Die Anzeige selbst unterscheidet zwischen „wir haben einen Text" und „wir haben keinen Text". Wenn wir keinen Text haben, also *strLenght <= 0*, dann zeichnen wir den Cursor in der Mitte des Bildschirms. Wenn wir Zeichen haben, kommt ein sog. *offsetX*-Wert hinzu.

offsetX berechnet sich aus *lenght/2+4*. Das kann, je nach Schrift und Größe, abweichen. *lenght* stützt sich auf die Funktion *string_width()*. Diese Funktion gibt die Breite (in Pixeln) der eingegebenen Zeichenkette zurück. Doch warum teilen wir durch zwei? Weil der Text horizontal zentriert angezeigt wird. D. h. die hälfte er errechneten Pixel wandert auf die andere Seite. Sollte der Text linksbündig sein, entfällt das.

Das war es auch schon. Nun kann man das Ganze ausschmücken. In einem meiner aktuellen Projekte

sieht es bspw. so aus:

Textbox in GMS

Rechts ist sogar eine Anzeige für die erlaubte Zeichenlänge. Wenn der Wert < 3 ist, wird es rot angezeigt. Ob die Mailadresse korrekt ist, wird erst bei der Eingabe angezeigt. Wie man das prüft, zeige ich in einem anderen Tutorial.

Date Created
8. April 2022
Author
sven