UI Randomizer

The UI randomizer is a python script for repeatedly calling the UI generator to create a desired amount of user interfaces with annotations.
The script will process the created annotation files and organize them in a folder structure suitable for a YOLO dataset.

Pre-requisites & Installation

This project uses Poetry for managing dependencies.

The `ui_randomizer.py` script requires the LVGL UI Generator v1 source in order to work. Further information about that generator can be found in its README.

The `ui_randomizer_v2.py` script requires the LVGL UI Generator v2 source in order to work. Further information about that generator can be found in its README.

Since there are two versions of the generator, there are also two versions of the randomizer script. The first version is `ui_randomizer.py` and the second version is `ui_randomizer_v2.py`.

Setting up the virtual environment

1. Install `poetry` package manager. See corresponding documentation for more information.
2. Run `poetry install` to install the dependencies and prepare the virtual environment.

Usage

Randomizer v1:

```
poetry run python src/ui_randomizer.py <arguments>
```

▶ Randomizer v1 Help

```
usage: ui_randomizer.py [-h] -p APP_PATH [-i ITERATIONS] -t WIDGET_TYPES [WIDGET_TYPES ...] [--width WIDTH] [--height HEIGHT] -o OUTPUT_FOLDER
                        [-d DELAY_COUNT] [--split_widgets] [-l LAYOUT] [-r SPLIT_RATIO] (-s | -m MULTI)

Capture UI and create image and annotation with correct folders.

options:
  -h, --help            show this help message and exit
  -p APP_PATH, --app_path APP_PATH
                        Path to the random UI generator binary
  -i ITERATIONS, --iterations ITERATIONS
                        Number of UIs to generate
  -t WIDGET_TYPES [WIDGET_TYPES ...], --widget_types WIDGET_TYPES [WIDGET_TYPES ...]
                        List of widgets to be used in the UI
  --width WIDTH         Width of the UI screenshot
  --height HEIGHT       Height of the UI screenshot
  -o OUTPUT_FOLDER, --output_folder OUTPUT_FOLDER
                        Folder to save the output images
  -d DELAY_COUNT, --delay_count DELAY_COUNT
                        Amount of times the timer handler shall be called with a fixed delay before capturing the UI
  --split_widgets       Split widgets into subfolders (only creates one widget type per iteration)
  -l LAYOUT, --layout LAYOUT
                        Path to the layout file to be used
  -r SPLIT_RATIO, --split_ratio SPLIT_RATIO
                        Split ratio for train, val, test
  -s, --single          Create only a single widget per iteration
  -m MULTI, --multi MULTI
                        Create multiple widgets per iteration
```

Randomizer v2:

```
poetry run python src/ui_randomizer_v2.py <arguments>
```

▶ Randomizer v2 Help

```
usage: ui_randomizer_v2.py [-h] [-mpy MICROPYTHON] [-m MAIN] -o OUTPUT_FOLDER [-r SPLIT_RATIO] [--datalist DATALIST] [-cwd WORKING_DIR] [--clean] [-d DELAY]
                           [--dataset DATASET] [--continue_on_error] [--capture_output] [--normalize] [-v] [--clearml_project CLEARML_PROJECT]
```

```
                          [--clearml_task CLEARML_TASK] [--
clearml_run_as_task] [--clearml_upload] [--normalize_bbox] [--
replace_class_names]
                          {random,design} ...

Invoke the generator and structure the captured UI images into a dataset

positional arguments:
  {random,design}       Generator options
    random              Random UI generator options
    design              Design file generator options

options:
  -h, --help            show this help message and exit
  -mpy MICROPYTHON, --micropython MICROPYTHON
                        Path to the micropython binary
  -m MAIN, --main MAIN  Path to the main script
  -o OUTPUT_FOLDER, --output_folder OUTPUT_FOLDER
                        Folder to save the output images

options:
  Additional options

  -r SPLIT_RATIO, --split_ratio SPLIT_RATIO
                        Split ratio for train, val, test
  --datalist DATALIST   Create a textfile with provided name to write all
images and labels
  -cwd WORKING_DIR, --working_dir WORKING_DIR
                        Working directory for the generator
  --clean               Clean the output folder before generating new data
  -d DELAY, --delay DELAY
                        Fixed delay between each generator call in
milliseconds
  --dataset DATASET     Name of the dataset
  --continue_on_error   Continue running the generator even if an error
occurs
  --capture_output      Capture the output of the generator
  --normalize           Activate normalize functionality of the generator
  -v, --verbose         Enable verbose output

clearml:
  Options for working with ClearML

  --clearml_project CLEARML_PROJECT
                        ClearML Dataset project name
  --clearml_task CLEARML_TASK
                        ClearML Dataset task name prefix
  --clearml_run_as_task
                        Run the randomizer as a ClearML task
  --clearml_upload      Upload the created dataset to ClearML

fixes:
  Annotation fixes

  --normalize_bbox      Post-process annotation files to normalize bounding
boxes
```

```
53  --replace_class_names
54                        Replace class names with their index in annotations
```

Randomizer v1

The v1 generator is written in C, so prior compilation of the binary is required. More information about this setup be found in the LVGL UI Generator v1 README

The randomizer script will call the generator binary with the provided arguments and save the generated UIs in the specified output folder.

The size of the dataset is set by the `--iterations` argument. The script will call the generator the specified amount of times.

By supplying the `--delay_count` argument, you can set the amount of times the timer handler shall be called with a fixed delay before capturing the UI. This is useful for fixing issues with user interfaces not being fully rendered before capturing.

The script will rename and move the generated images and annotations to a folder structure suitable for a YOLO dataset.

By supplying the `--split_widgets` flag, the script will only generate screenshots containing a single widget type per iteration.

You can also specifiy the layout in which widgets will be structured by providing the `--layout` argument. The argument can be either `grid`, `flex`, or `none`.

If you only want to have a single widget per iteration, you can provide the `--single` flag. If you want to have multiple widgets per iteration, you can provide the `--multi` flag followed by the number of widgets you want to have.

The types of widgets used in the generator is specified by the `--widget_types` argument. The argument should be a list of widget types separated by spaces.

Randomizer v2

The v2 generator is written in micropython, so prior compilation of the micropython binary is required. More information about this setup be found in the LVGL UI Generator v2 README

The randomizer has optional annotation fixes that it can apply to the created annotation files. The fixes include normalizing bounding boxes (`--normalize_bbox`) and replacing class names (`--replace_class_names`) with their index in the widget list.

These fixes are used by providing their respective flags in the command line arguments.

Furthermore, you can adjust the split ratio for the dataset by providing the `--split_ratio` argument. The argument should be a string of three numbers separated by commas. The numbers represent the ratio for the train, validation, and test sets respectively.

Since calling the generator can produce a lot of output, it is disabled by default. You can enable printing to the console by providing the `--capture_output` flag.

The script and generator may also produce errors during the generation process and will abort by default. You can choose to continue running if such events occur by providing the `--continue_on_error` flag.

The output of the script produces a dataset, but you can activate an additional datalist output by providing the `--datalist` argument. The argument should be the name of the text file that will be created in the output folder. The file will contain the paths to the images and annotations in the dataset in the form of `image_path annotation_path`.

By default, the script will error if the provided output folder is not empty. You can choose to clean the folder before generating new data by providing the `--clean` flag. Be aware that this will delete all files in the output folder.

To slow down the generation process, you can provide a fixed delay between each generator call in milliseconds by providing the `--delay` argument. This can be useful for fixing issues with user interfaces not being fully rendered before capturing, or to simply watch the generation process.

Modes

The randomizer v2 script supports both random and design modes of the generator.

For details about these modes, see the LVGL UI Generator v2 README.

▶ Help for random mode

```
usage: ui_randomizer_v2.py random [-h] -t WIDGET_TYPES [WIDGET_TYPES ...] [-W WIDTH] [-H HEIGHT] [--split_widgets] [-c COUNT] [-l LAYOUT] [-i ITERATIONS]

options:
  -h, --help            show this help message and exit
  -t WIDGET_TYPES [WIDGET_TYPES ...], --widget_types WIDGET_TYPES [WIDGET_TYPES ...]
                        List of widgets to be used in the UI
  -W WIDTH, --width WIDTH
                        Width of the UI screenshot
  -H HEIGHT, --height HEIGHT
                        Height of the UI screenshot
```

```
  --split_widgets       Split widgets into subfolders (only creates one
widget type per iteration)
  -c COUNT, --count COUNT
                        Number of widgets to create per iteration
  -l LAYOUT, --layout LAYOUT
                        The main container layout of the random UI ["grid",
"flex", "none"]
  -i ITERATIONS, --iterations ITERATIONS
                        Number of UIs to generate
```

▶ Help for design mode

```
usage: ui_randomizer_v2.py design [-h] [-f DESIGN_FOLDER]

options:
  -h, --help            show this help message and exit
  -f DESIGN_FOLDER, --design_folder DESIGN_FOLDER
                        Folder containing the design files
```

ClearML Integration

The randomizer v2 script has integration with ClearML to directly upload the created dataset to a ClearML project.

To use this feature, you need to provide the `--clearml_upload` flag. You can also specify the ClearML project name with the `--clearml_project` argument. Additionally, you can run the generation process as a ClearML task by providing the `--clearml_run_as_task` flag. This will run the script as a ClearML task, which is useful for tracking & storing the progress of the generation process.
You may also specify the task name prefix with the `--clearml_task` argument.

Known issues

License

This project is licensed under the MIT License - see the LICENSE file for details.