Unsupervised Graphical User Interface Learning

Przemysław Czaus [1] [0000-0003-1108-525X]

[1] Department of Mathematical Methods of Computer Science, Faculty of Mathematics and Computer Science, University of Warmia and Mazury in Olsztyn, Słoneczna 54, 10-710 Olsztyn, Poland

czaus@matman.uwm.edu.pl

Abstract. While there are many different papers on automatic testing or generation of interfaces[1, 2, 3, 4], the information concerning similar to human perception user interface learning methods are scarce. Given the recent leaps in Artificial Intelligence (AI) one might find this topic very useful in implementing a solution for communication of AI with applications created for human interaction. User interfaces differ between operating systems, hardware possibilities and implementation. The main objective of the study was the analysis of the applications' Graphic User Interface (GUI) and interaction with the graphic interface elements in general. This method can be used to track the change in application behaviour based on any action from the user, without having knowledge about the underlying object structure.

Keywords: artificial intelligence, machine learning, user interface

1 Introduction

Currently, the main component of any human-application interaction is the output data returned in a graphical or textual form. This data is then processed and another input data causes an event that changes (or not) the output. For AI to handle repeating tasks as humans, it must work with existing UI. The main focus of this paper is to prepare an easy to implement algorithm that focuses on tracking change between the screens of the application and in turn create a map of the interface elements. underlying events and changes in the GUI. It may help build more sophisticated mechanisms for modeling changes between views and finding similarities between GUI elements and understanding the key steps needed to be done to achieve an expected result.

In the underlying sections one can find information about:

- Algorithm first tasks to find GUI elements
- Detecting change
- Optimizations
- Mapping interface elements and changes in the UI

2 Example Algorithm

2.1 Generating events

Currently, the main component of any human-application interaction is the output data returned in a graphical or textual form. This data is then processed and another input data causes an event that changes (or not) the output (see Fig. 1)

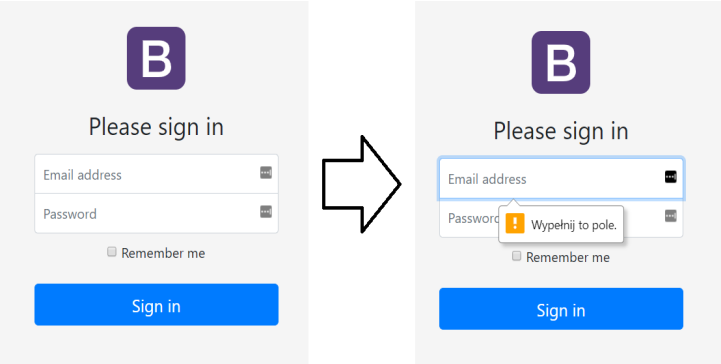

Fig. 1. Change in UI after clicking the sign in button

The first step is identifying the change in the application behaviour that in turn help us identify the event that caused the change. This is something that everyone unwillingly processes while surfing the internet, playing video games or using an application. Where the speed of identifying every action differs depending on the person, the process is roughly the same (link article that has this data) and we start by inputting data (like click on elements or keys on the keyboard) and we look for changes in the output (like an information message that the input was wrong, or animations suggesting data transfer). The output of a single event can be called a frame or a screenshot (see Fig. 2).

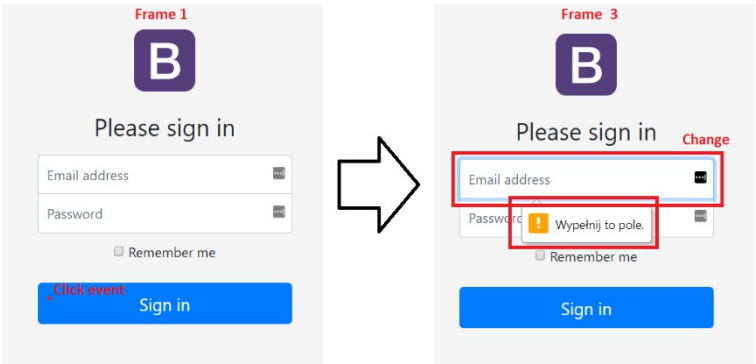

Fig. 1. Click event on the sign in button generates a new frame.

2.2 Optimizations

The first step is to generate a current GUI render image. It isn't important if it's a web page or an application. The main focus is to get the output data that is an image of the working application. Generating an image that contains only the application window and any action is limited to the window borders, ensures that no action outside the application window will interfere with the data processing. One can limit the input size by creating a diff from the first frame and another one is generated some time after the initial render(see Fig. 3)..

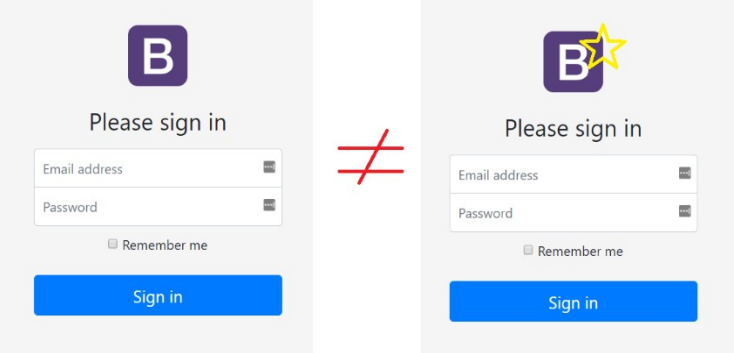

Fig. 1. First and second frames differ

This may cause some delay between starting of the application and processing the data, but may minimize errors on slower systems. Limiting events only to parts of the image that didn't change can significantly lower the number of iterations of the algorithm. This approach can cause problems in applications that animate user interfaces or generate animations before the GUI is initialized(see Fig. 4).

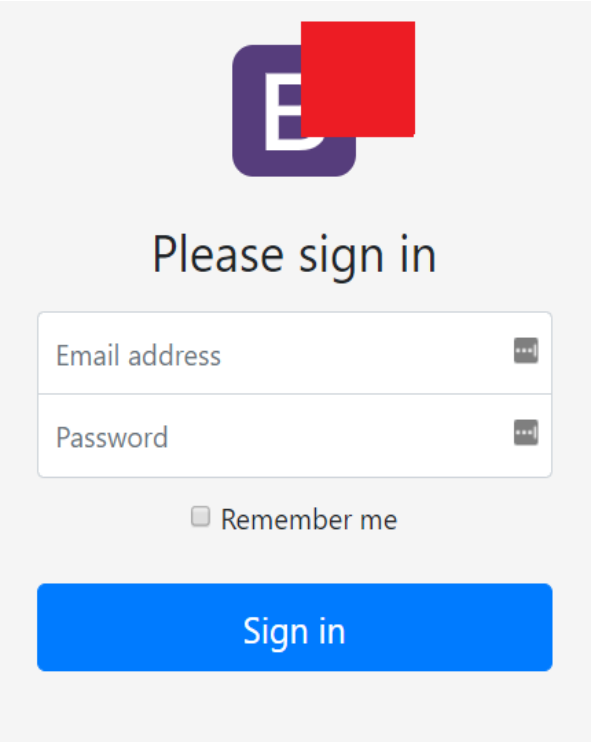

Fig. 1. Red area will be ignored

2.3 Tracking change

The second phase is generating input data that should cause execution of an event. For the purpose of this example coordinates start from [0,0] (the upper left corner of the application window, without the status bar). We can limit those events to occur only in coordinates that didn't change between two first frames. In most cases the interface elements are stationary. This applies to modern video games, web pages or mobile applications. After executing an action (click, long click, swipe etc. on coordinates [0,0]) we should wait for an event to make the necessary changes in the GUI. Depending on the type of an application and processing power some delay should be used after every action and new image generation.

At this point, we have at least 3 states of the application: the initial view (1), view after the first few seconds from generating the view (2), after the last event (3) (see Fig. 5).

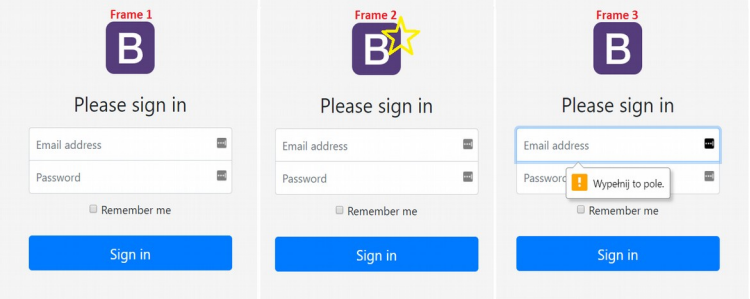

Fig. 1. First three frames

2.4 Mapping interface elements to changes

Change of application state can be checked generating a diff between the first and the last frame (or, if any changes occurred, the second). If we find any differences between the frames we can assume that an event was fired and the GUI changed. Generating a hash based on this image will be useful for future reference. The same operations should be repeated for every co-ordinate of the application interface. This will help us generate a map of interface elements corresponding to frames generated by those click events. Any action that generates the same frame should be grouped as part of the same interface element (see Fig. 6).

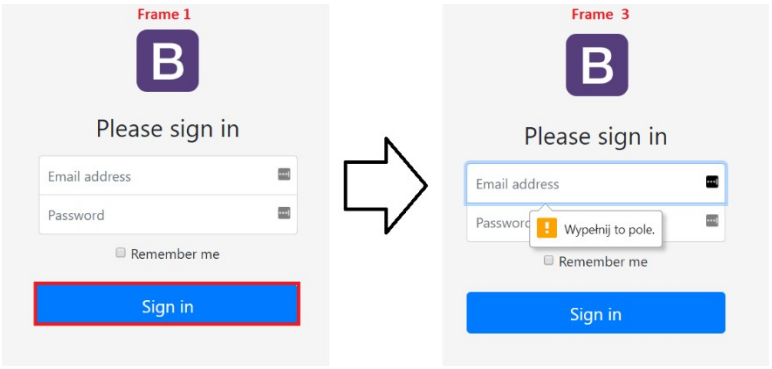

Fig. 1. All click actions on the sign in button generate frame 3

This provides enough data to generate a decision tree for the initial application view with actions that allow us to get to the second level frames.

Treating every next level frame as a starting point allows us to find new views. Checking hashes of every level frames help us identify any actions that can send us to a previous view. After finishing all the iterations we have all the unique frames with the corresponding coordinates and actions made on those interface elements. It's possible to verify if frames have similar features (like information about a single list item) for grouping purposes. With an application fully mapped it's possible to train classifiers to find some similar interface elements which in turn makes the process of analyzing new applications faster(see Fig. 7).

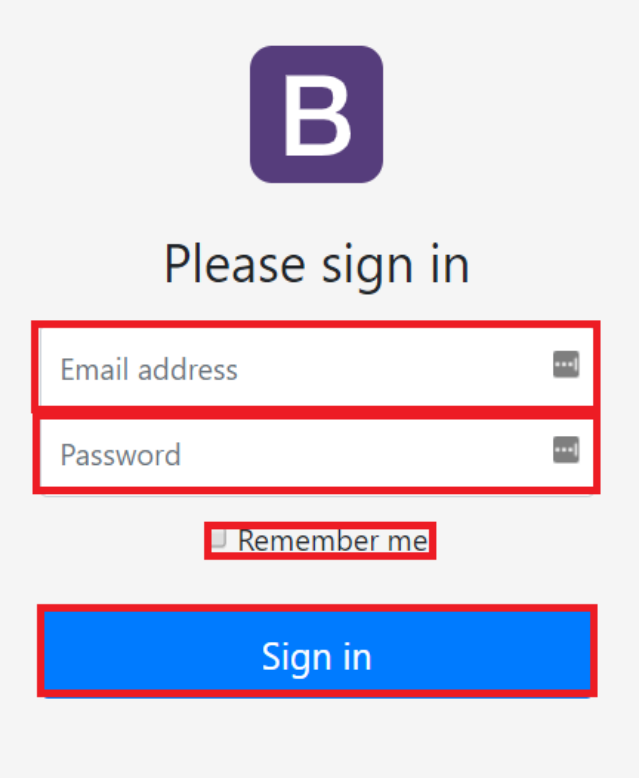

Fig. 1. All identified interface elements that generate change to the GUI

Using the coordinates we can generate smaller images that only contain a single interface element. At this point for unique interface elements, we have datasets with only one row. The grouped elements have as many rows as similar objects were found during the application mapping.

3 Conclusions

Where creating mapping solutions for a single technology may be more precise in finding exact UI elements, a more general approach might give more benefits long term. Without focusing on a single implementation of all the fazes of data processing it's a very interesting topic to test many different approaches to data mining and computer vision. Many optimizations are still possible in many different steps, yet tracking changes is still possible in a closed amount of time.

References

1. Chunyang Chen, Ting Su, Guozhu Meng, Zhenchang Xing, and Yang Liu: From UI Design Image to GUI Skeleton: A Neural Machine Translator to Bootstrap Mobile GUI Implementation. In Proceedings of ICSE '18: 40th International Conference on Software Engineering , Gothenburg, Sweden, pp. 665-676.
2. Shauvik Roy Choudhary, Alessandra Gorla, and Alessandro Orso. 2015. Automated Test Input Generation for Android: Are We There Yet? (E). In 30th IEEE/ACM International Conference on Automated Software Engineering, ASE 201Ssd
3. Ting Su. 2016. FSMdroid: Guided GUI testing of android apps. In Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016 - Companion Volume. 689–691.Fd
4. Ting Su, Guozhu Meng, Yuting Chen, KeWu,Weiming Yang, Yao Yao, Geguang Pu, Yang Liu, and Zhendong Su. 2017. Guided, Stochastic Model-based GUI Testing of Android Apps. In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2017). ACM, New York, NY, USA, 245–256.