An Environment for Learning Interactive Programming

Terry Tang
Department of Computer Science
Rice University
Houston, Texas
terry.tang@rice.edu

Scott Rixner
Department of Computer Science
Rice University
Houston, Texas
rixner@rice.edu

Joe Warren
Department of Computer Science
Rice University
Houston, Texas
jwarren@rice.edu

ABSTRACT

We describe a web-based programming environment designed to support teaching introductory programming for a massive open online class. We discuss some of the thought processes behind the design of this environment and then focus on two key innovations incorporated in our environment: a simplified GUI library for interactive Python programming and a browser-based tool for visualizing the execution of event-driven Python programs.

Categories and Subject Descriptors

K.3.1 [**Computers and Education**]: Computer Uses in Education—*Computer-assisted instruction (CAI)*
; K.3.2 [**Computers and Education**]: Computer and Information Science Education—*Computer science education*

General Terms

Languages, Human Factors, Experimentation

Keywords

CS1, interactive, Python, visualization

1. INTRODUCTION

Learning introductory programming is a critical part of the education of any student interested in Computer Science. While introductory programming has long been a staple of on-campus instruction, the last decade has seen online delivery substantially extend the availability of introductory programming to a wider range of students. In the summer of 2012, the authors committed to designing and teaching a massive open online class (MOOC) on introductory programming for tens of thousands of students. As part of this class, the authors developed a programming environment designed to support novice students learning to code.

The design of the class and its programming environment embodied several important choices that we believe significantly influenced the success of the MOOC.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SIGCSE'14, March 5–8, 2014, Atlanta, GA, USA.
Copyright 2014 ACM 978-1-4503-2605-6/14/03 ...$15.00.
http://dx.doi.org/10.1145/2538862.2538908.

- Due to its popularity and simplicity, Python was chosen as the programming language for this class.
- To engage and motivate students, the class focused primarily on creating simple interactive games in Python.
- For ease of deployment and uniformity, the class environment for coding and running interactive Python programs was entirely web-based.

In early Fall 2012, the authors implemented a web-based environment for creating and running interactive Python programs known as CodeSkulptor[1]. Figure 1 shows CodeSkulptor running a simple interactive program that spawns a separate frame that displays the message "Welcome!". For this program, clicking the button labeled "click me" will change the displayed message to "Good job!". The Python code for this program is displayed in the code pane on the left while the frame overlays a console on the right where textual output is displayed.

In Fall 2012, Spring 2013 and Fall 2013 sessions, CodeSkulptor was used as the primary coding environment for the authors' MOOC, "An Introduction to Interactive Programming in Python"[2]. This class (IIPP) was a substantial success, awarding Statements of Accomplishment to almost 19,000 students and receiving several thousand positive reviews at multiple external MOOC review sites. In our view, the design of CodeSkulptor contributed to much of this success.

Contributions While the design of CodeSkulptor leverages many existing ideas, CodeSkulptor also incorporates several novel features.

- To harness the motivational power of building games, CodeSkulptor needed to support an interactive GUI library. After reviewing the existing GUI libraries for Python, we decided to create a simplified GUI library for interactive Python with a pedagogical focus.
- To assist students in understanding the behavior of their interactive, event-driven programs, we created a browser-based tool that records a trace of the execution of an event-driven program. The tool allows the student to manually fire events such as draws and timers that are normally triggered automatically during program execution. Further, the tool allows the students to step through the resulting trace at either a statement level or an event level while visualizing the associated program state.

[1] `http://www.codeskulptor.org/viz/`
[2] `https://www.coursera.org/course/interactivepython`

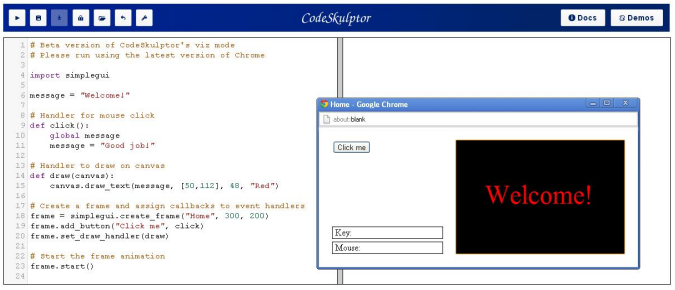

Figure 1: CodeSkulptor

2. RELATED WORK

Our synthesis of CodeSkulptor took place in the context of a large body of related work. We review this work and and discuss our rationale behind the design of CodeSkulptor in the context of this work.

Web-based programming tools In the summer of 2012, a number of web-based environments for programming already existed. Examples of these environments include repl.it, jsfiddle.org, and Cloud9 [1]. All of these environments include two features that we believe are critical for use in MOOCs. The environments offer web-based creation and cloud-based saving and sharing of programs.

Out of these environments, only repl.it offers support for Python that is completely browser-based. Unfortunately, our preliminary tests with repl.it indicated that its performance in running Python programs in a browser was insufficient to support building even games of modest complexity. This situation led us to create CodeSkulptor, which is a fusion of Code Mirror[3] and Skulpt[4], a tool for running Python programs in a web browser. Like the other environments listed above, CodeSkulptor supports cloud-based saving and sharing of programs.

Game-based introductory programming In designing a MOOC to teach introductory programming, one of our first priorities was to create a class that engaged the interest and imagination of our students. One well-established approach is to focus introductory programming classes on building games [11, 8]. This approach has been shown to be effective in maintaining and increasing the motivation of students [2, 10]. Programming environments geared towards games such Alice [3] and Scratch [13] are very popular and have attracted large communities. However, as was the case for an environment like Greenfoot [7] (focused on Java), we decided to teach a language that is more commonly used like Python to retain a wider appeal.

GUI libraries for Python Given our decision to focus on building games in Python, our next task was to decide on a GUI library for use with Python. A number of full-featured GUI libraries such as Tkinter [4] and wxPython [12] are available for Python. Another Python GUI that is specifically game-focused is Pygame [9]. Ultimately, we decided against using an existing GUI library for Python for the reasons discussed in section 3.

Tools for visualizing programs To help novice programmers understand the behavior of their programs, some environments for learning programming include tools to visualize the execution of programs. [14] surveys some of the existing techniques for visualizing program execution with Greenfoot, Jype (Python) [6], and Online Python Tutor [5] being particularly relevant.

Since none of these tools are fully-browser based or provided the capability to visualize the execution of event-driven programs, we decide to create an extension to CodeSkulptor that incorporates the frame/object visualization capabilities of Online Python Tutor while allowing both statement level and event level stepping through a program trace.

3. A GUI LIBRARY FOR PYTHON

Python has a number of GUI libraries (Tkinter, wxPython, Pygame, etc.) available for creating interactive programs. Two issues swayed us against using one of these libraries in our course. First, none of these libraries are designed to support browser-based Python. So we would need to port any existing library that we chose to the browser. Second, these libraries are moderately complex to use since they are designed to create full-featured user interfaces. As a result, the GUIs have a level of complexity in their design that might be overwhelming for novice programmers. These observations led us to choose to design and implement a web-based GUI for Python called SimpleGUI.

3.1 Pedagogical design of SimpleGUI

The SimpleGUI library supports the standard event-driven programming model in which the behavior of an interactive Python program is modeled as a sequence of events triggered either manually via controls such as buttons, input fields, key presses or mouse clicks or automatically via timers or drawing. In this model, the user creates a frame and then associates an event handler with this frame for each of these types of events. To generate visual output on the canvas associated with the frame, the user has a small set of drawing commands for entities such as text, simple geometric shapes, and rotated images. To generate audio output, the user can play start, pause and rewind a sound.

In designing SimpleGUI, our objective was to balance the complexity of the GUI library versus its expressiveness. In particular, we were interested in having the ability to create moderately complex games (on the level of 1980/1990s arcade games) using a relatively easy-to-learn GUI library. For example, full-featured GUIs allow precise control over the button layout, but at the expense of numerous parameters and options. However, for the class of games that we envisioned students building, precise control of button placement was not particularly important. Thus, we provided only limited options for button placement and reduced the complexity of SimpleGUI in that instance.

In general, we added features to SimpleGUI only when these capabilities significantly extended the expressiveness of the library. With this approach, students were able to create a range of games in SimpleGUI using relatively small Python programs. Weekly class projects included Rock-paper-scissors-lizard-Spock (50 lines), Pong (100 lines), Blackjack (200 lines), and Asteroids (350 lines in 2 weeks).

We found that one bonus from taking an event-driven approach was that students could create surprisingly interesting interactive programs using only a small set of programming constructs. For example, weekly projects in the first half of the class like "Guess the number", Stopwatch, and

[3] `http://codemirror.net`
[4] `http://skulpt.org`

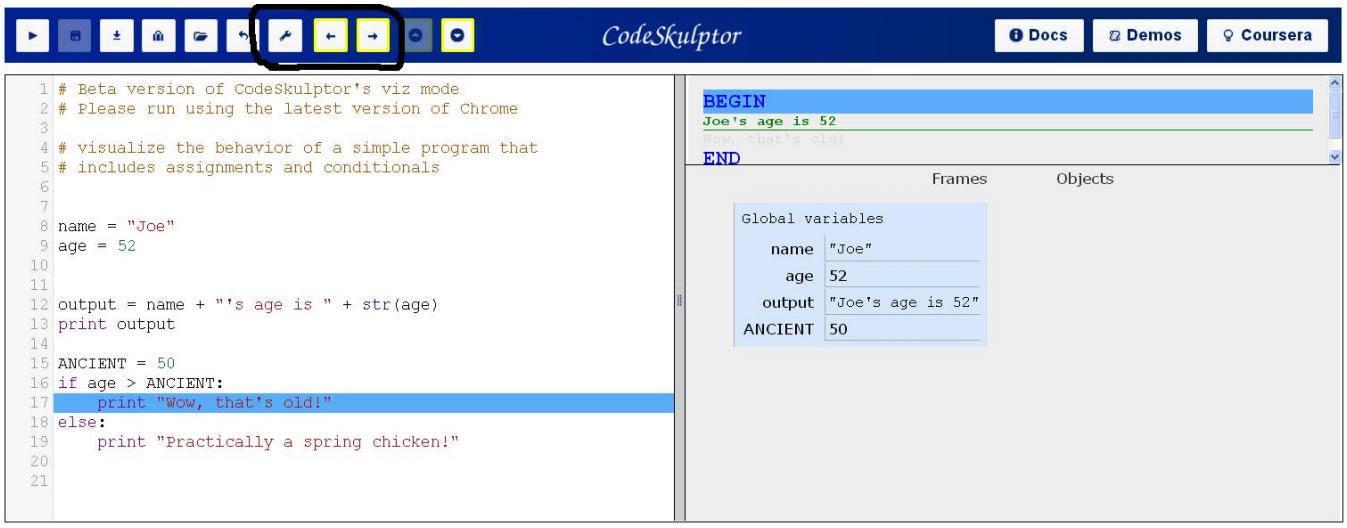

Figure 2: Visualizing the state of a simple test program in CodeSkulptor. Circled buttons activate Viz mode (wrench icon) and step through the program trace at a statement level (left/right arrow icons).

Pong require no knowledge of loops or mutation. As a result, we were able to delay introduction of these more advanced concepts until the second half of the class.

3.2 Technical structure of SimpleGUI

Internally, CodeSkulptor utilizes Skulpt[5] as a Python engine. Specifically, Skulpt translates Python code into Javascript and then executes the resulting Javascript in the browser. This approach provides sufficient performance for modest interactive games, unlike other web-based Python programming environments which either incur too much overhead or execute code on the server-side.

In a similar manner, our implementation of SimpleGUI supports event-driven programming by translating event-driven Python code into its Javascript equivalent. SimpleGUI supports creation of an interactive frame with a control area and drawing canvas as well as audio using a combination of Javascript and HTML5. Specifically, SimpleGUI allows Python programs to:

- Create a frame (`simplegui.create_frame`),
- Create timers (`simplegui.create_timer`),
- Load images (`simplegui.load_image`),
- Load sounds (`simplegui.load_sound`).

The SimpleGUI module can be imported into CodeSkulptor programs using `import simplegui` and provides an easy to use GUI library that runs across web browsers such as Chrome, Firefox and Safari. SimpleGUI is not meant to be a full-featured GUI library. However, it provides a consistent and easy to use environment for building modestly complex games.

4. BROWSER-BASED VISUALIZATION

Many programming environments, especially those designed for pedagogical purposes, incorporate tools that allow users to visualize the execution of their programs. Our visualization tool for CodeSkulptor follows a philosophy similar to that of Online Python Tutor[6], allowing statement level visualization of program state via a frame/object diagram. Here, the frame portion of the diagram displays pending function calls and their associated variables while the object portion displays reference diagrams for mutable objects.

In evaluating Python Tutor for use in IIPP, we hesitated in using Python Tutor as is for two reasons. First, Python Tutor uses a back-end server to construct the execution trace used during visualization. We worried that this design might prove problematic during periods of high demand around due dates for IIPP. Second, Python Tutor lacks the capability to visualize the execution of event-driven programs.

4.1 Visualizing sequential programs

In light of these issues, we decide to integrate Python Tutor's capabilities directly into CodeSkulptor. The key technical obstacle to this integration was avoiding the need for a back-end server to generate the execution trace of the program. Our approach was to add an option to CodeSkulptor that, when enabled, would instrument the Javascript code created during compilation of a Python program by Skulpt. Executing this instrumented Javascript code would then not only evaluate the corresponding Python program, but also generate an execution trace for the Python program. We could then use Python Tutor's frame/object diagram visualization capabilities (available in open source form as Javascript) to display a selected state in this trace.

To activate Viz mode in CodeSkulptor, the user clicks the activation button (the leftmost circled button with the wrench icon) on the upper left row of buttons. This action enables instrumentation of the compiled Python code and updates the UI for CodeSkulptor in two ways.

- A pair of yellow previous/next buttons is placed to the right of the wrench. These buttons allow the user to navigate the execution trace at a statement level.

[5] `http://www.skulpt.org`

[6] `http://pythontutor.com`

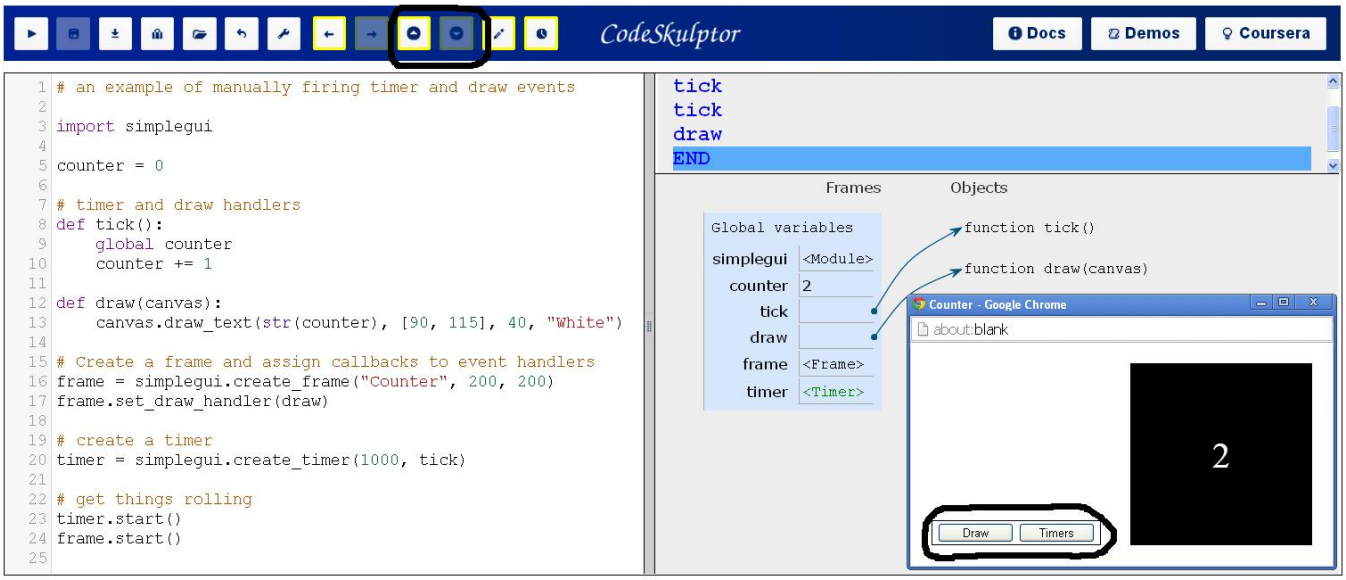

Figure 3: Visualizing the state of a counter program that spawns a frame and displays a counter on the canvas (lower right). The two circled buttons on the frame allow manual firing of draw and timer events. The upper two circled buttons allow event level navigation of the associated program trace.

- The console is restricted to the upper portion of the righthand pane while the lower portion is used to display the frame/object diagram corresponding to the program state for the selected statement.

Figure 2 shows an example of visualizing one particular state during the execution trace of a simple sequential Python program[7]. This program involves no user interaction and simply prints computed output to the console.

After running the program, the user has navigated through the resulting execution trace using the circled buttons and selected the `print` statement highlighted on the left. The output and state of this program **prior** to the execution of the selected statement are highlighted in the console (upper right) and the frame/object diagram (lower right), respectively. Note that the output `"Joe's age is 52"` from a previously executed `print` statement is displayed in the console. On the other hand, the output `"Wow, that's old!"` from the currently highlighted `print` statement is only faintly displayed to indicate that it has not yet been executed.

Viz mode may also be used to debug programs that throw an error. Executing such a program in Viz mode generates a program trace up to the point at which the error is thrown and is very helpful in debugging.

4.2 Visualizing event-driven programs

During normal execution of event-driven programs, timers and draws may automatically generate hundreds of events in a very short period. Attempting to visualize the behavior of such programs by adding `print` statements to the handlers for these events often results in a blizzard of textual output to the console. In Viz mode, CodeSkulptor requires that all events including draws and timer ticks be triggered manually to avoid this issue. When a program including draw or timer events is run in Viz mode, CodeSkulptor automatically adds buttons to the user interface allowing manually firing of these events when desired.

Figure 3 shows the state of a program that creates a frame, updates the value of a counter via a timer and displays the value of that counter on the canvas associated with the frame[8]. The console shows a log of the events that have been manually fired by the user using the circled "draw" and "timer" button on the frame. Specifically, the user has fired two timer `tick()` events, updating the value of `counter` to `2`, as shown in the state diagram. Finally, the user has fired draw event that draws the value of the counter on the canvas using a SimpleGUI canvas operation. Note that the label "END" in the console is highlighted to alert the user that the displayed state corresponds to the end of the current trace.

At this point, the user may navigate between existing events using the circled prev/next event buttons to visualize the program state prior to the execution of each event. Inside a particular event, the user can navigate at a statement level using the next/prev statement buttons. While stepping through a draw event, the canvas is updated after each SimpleGUI canvas operation allowing the user to visualize the composition of the canvas dynamically.

The user may also fire new events by clicking the "draw" and "tick" buttons. These new events are appended to the event log displayed in the console and the execution traces associated with these event are appended to the current program trace. This functionality allows the user to visualize the state of the program interactively via the frame/object diagram. Note that the updated program trace is generated entirely in the browser. This design is essential in enabling the interactivity of this process. Otherwise the browser would need to send the code for the event and the current program state to a remote server to generate a trace for each new event that is fired.

[7] `http://www.codeskulptor.org/viz/#viz_tutorial_state.py`

[8] `http://www.codeskulptor.org/viz/#viz_tutorial_drawing.py`

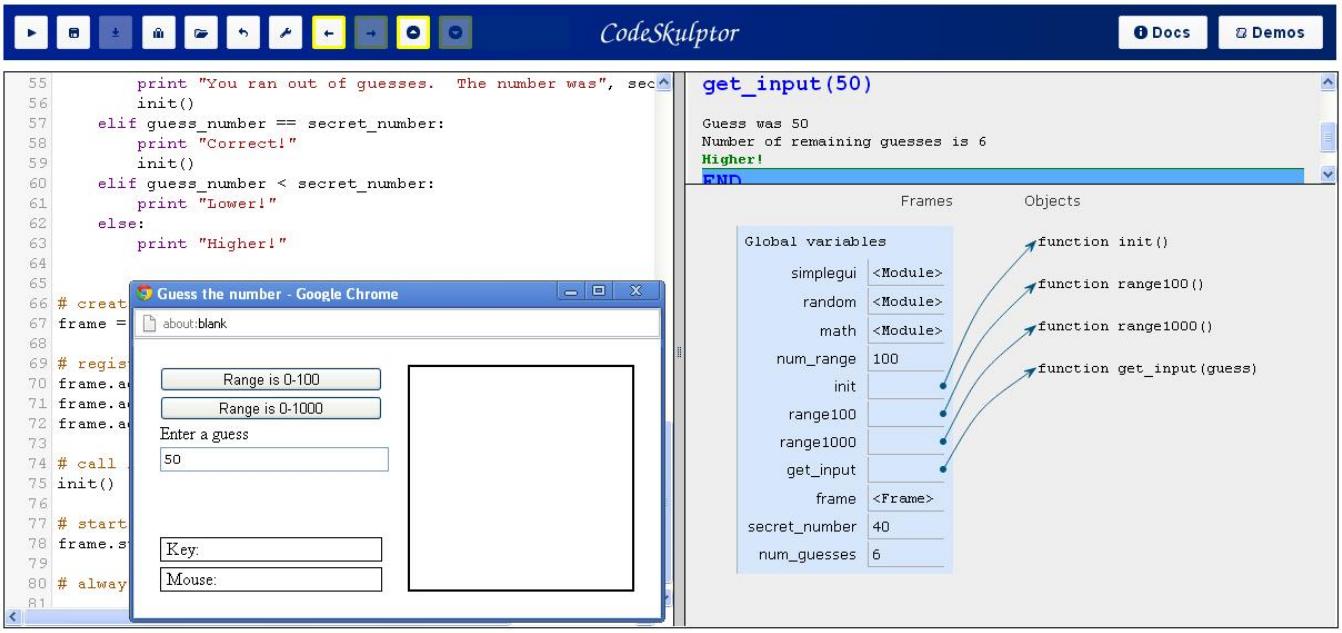

Figure 4: Final state of "Guess the number" after an "input" event

4.3 A pedagogical example

In this subsection, we consider a simple illustrative example of using CodeSkulptor's Viz mode to understand the behavior of an incorrectly-coded interactive program. "Guess the number" is a simple game in which the computer generates a secret number between 0 and 100. The player then inputs numbers in that range and the computer responds `Higher!` or `Lower!` based on the value of the guess and the secret number.

Figure 4 shows the state of an implementation of the game "Guess the number"[9] after the player has entered an initial guess of `"50"`. The value of the secret number, 40, is displayed in the frame/object diagram while the value 50 for the input event is specified in the event log in the console. At this point, the user notes that text message in the console incorrectly displays `Higher!` instead of `Lower!`.

Based on this observation, the user navigates the trace of the execution of this event. Figure 5 shows the state of the program prior to the execution of the incorrect `print` statement. At this point, the user has all of the information in the frame/object diagram necessary to resolve the error. In particular, the `secret_number` is 40 while `guess_number` is 50 yet the execution ended up in the "Higher!" clause of the conditional statement. These clues point to the error: the comparison used in the `elif` clause is reversed causing the `Higher!` and `Lower!` messages to be switched.

The example illustrates the basic approach that we suggest in using CodeSkulptor's Viz mode to understand the behavior of an interactive program.

- CodeSkulptor executes the body of the program and records a trace of the execution. The frame/object diagram and the console display the program state and output.
- The user then manually fires events using either timer/draw buttons or by interacting with the frame. The output of the program is then visualized via the console and canvas while the state is visualized via the frame/object diagram.
- When the user observes an event whose effect on the output or state is unexpected, they then explore execution of the particular event using the statement navigation buttons.

In practice, we have found the ability to manually fire events and immediately visualize program state after each event to be valuable in understanding the behavior of interactive programs.

5. FUTURE WORK

Viz mode records a log of the events associated with a specific execution. In future work, we plan to incorporate the ability to load and save these event logs to facilitate easier testing and debugging of event-driven programs. In particular, we intend to investigate the possibility of creating a unit testing scheme for event-driven programs.

One key task would be to specify the state of the canvas formally after each draw event as a sequence of canvas draw operations. Verifying the correctness of submitted program would boil down to checking the equivalence of this sequence with the sequence produced by the submitted program algorithmically. We believe that this capability is easily within the capability of modern program verification and would enhance our ability to specify the behavior of interactive programs as well as the student's ability to verify that their programs work correctly.

In summary, we have described a web-based environment for creating and running interactive Python programs. The GUI portion of the environment (SimpleGUI) has been successfully used by tens of thousands of students in three sessions of IIPP. Viz mode was released for informal use in Fall

[9] `http://www.codeskulptor.org/viz/#viz_pedogogicalExample.py`

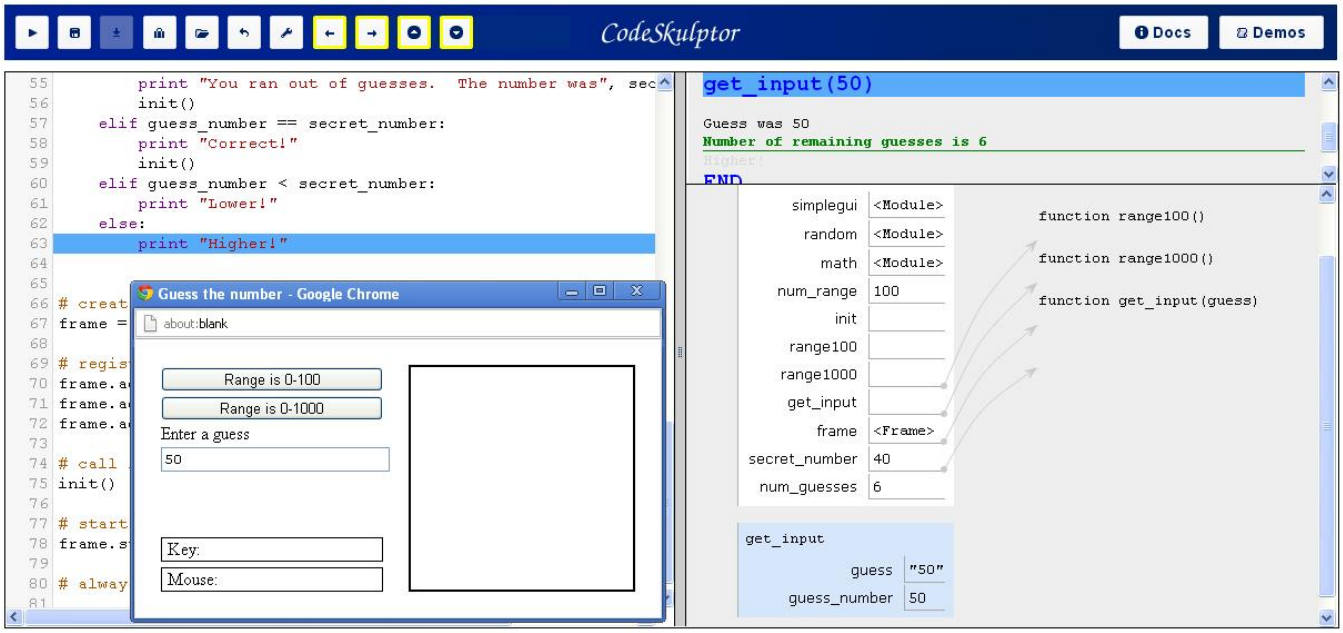

Figure 5: An earlier state during the execution of the "input" event

2013 session of IIPP. In future sessions, we plan to integrate Viz mode into the pedagogy of the class and engage in a user study to assess the effectiveness and usability of Viz mode.

Acknowledgements This research was supported by NSF Award CCF-1320860: "Computer-Aided Grading, Feedback, and Assignment Creation in Massive Online Programming Courses". We would like to extend our thanks to the Online Python Tutor project for making their state visualization code available for use in CodeSkulptor. Finally, we would like to thank John Greiner and Stephen Wong for their assistance in creating and teaching IIPP.

6. REFERENCES

[1] L. Ciortea, C. Zamfir, S. Bucur, V. Chipounov, and G. Candea. Cloud9: a software testing service. *ACM SIGOPS Operating Systems Review*, 43(4):5–10, 2010.

[2] D. C. Cliburn. The effectiveness of games as assignments in an introductory programming course. In *Frontiers in Education Conference, 36th Annual*, pages 6–10. IEEE, 2006.

[3] S. Cooper, W. Dann, and R. Pausch. Alice: a 3-d tool for introductory programming concepts. In *Journal of Computing Sciences in Colleges*, volume 15, pages 107–116. Consortium for Computing Sciences in Colleges, 2000.

[4] J. E. Grayson. *Python and Tkinter programming*. Manning Publications Co., 2000.

[5] P. J. Guo. Online python tutor: Embeddable web-based program visualization for cs education. In *Proceeding of the 44th ACM Technical Symposium on Computer science education*, pages 579–584. ACM, 2013.

[6] J. Helminen and L. Malmi. Jype-a program visualization and programming exercise tool for python. In *Proceedings of the 5th International Symposium on Software Visualization*, pages 153–162. ACM, 2010.

[7] M. Kölling and P. Henriksen. Game programming in introductory courses with direct state manipulation. *ACM SIGCSE Bulletin*, 37(3):59–63, 2005.

[8] S. Leutenegger and J. Edgington. A games first approach to teaching introductory programming. In *ACM SIGCSE Bulletin*, volume 39, pages 115–118. ACM, 2007.

[9] W. McGugan. *Beginning Game Development with Python and Pygame*. Will McGugan, 2007.

[10] A. M. Phelps, C. A. Egert, and J. D. Bayliss. Media impact: Games in the classroom: Using games as a motivator for studying computing: Part 1. *Multimedia, IEEE*, 16(2):4–8, 2009.

[11] R. Rajaravivarma. A games-based approach for teaching the introductory programming course. *ACM SIGCSE Bulletin*, 37(4):98–102, 2005.

[12] N. Rappin and R. Dunn. *wxPython in Action*. Manning, 2006.

[13] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, et al. Scratch: programming for all. *Communications of the ACM*, 52(11):60–67, 2009.

[14] J. Urquiza-Fuentes and J. Á. Velázquez-Iturbide. A survey of successful evaluations of program visualization and algorithm animation systems. *ACM Transactions on Computing Education (TOCE)*, 9(2):9, 2009.