

# Benchmarking Pygame vs. Tkinter: Performance and Code Complexity in Python GUI Development

Saltanat Biibosunova<sup>1</sup>, Daniyar Biibosunov<sup>2</sup>, Ma Yi Ming<sup>3</sup>, Liu Fang<sup>4</sup>, Fan Xingzhou<sup>5</sup>, and Huang Da<sup>6</sup>

<sup>1–6</sup>Institute of the New Information Technologies, Arabaev Kyrgyz State University, Bishkek, Kyrgyzstan  
<sup>4</sup>328146066@qq.com, <sup>5</sup>1509500278@qq.com

**Abstract**—The research evaluates two Python GUI (Graphical User Interface) development methods through the Tkinter and Pygame libraries. The event loop and drawing facilities of Pygame enable developers to use this library for creating general-purpose GUI applications despite its main purpose being 2D game development. The research evaluates which approach provides better performance and faster response times, and easier development processes.

There are identical functional applications implemented through Tkinter and Pygame to compare their performance. The execution time results demonstrate Pygame outperforms Tkinter by 3.5–3.7 times in both startup and processing speed, while certain tasks show performance improvements of six to seven times. The Pygame codebase needed fewer lines of code and provided a more straightforward abstraction for handling events and rendering graphics.

It shows Pygame provides equivalent performance to Tkinter for applications that need fast GUI rendering and real-time graphics, and animation. The research shows that Tkinter works best for basic form-based interfaces and standard desktop widgets, but Pygame excels at performance-critical GUI applications. The research provides guidance about toolkit selection based on application requirements and suggests future investigation directions.

**Index Terms**—Python, graphical user interface (GUI), Pygame, Tkinter, event-driven programming.

## I. INTRODUCTION

Python is useful for teaching, for prototypes, and for building software that works on platforms. Python also lets us make user interfaces, graphical user interface (GUIs). The standard library provides Tkinter. Tkinter is an object-oriented wrapper around the Tk toolkit. The Tk toolkit uses the windowing APIs of the operating system. The Pygame library runs on the Simple DirectMedia Layer (SDL) [1].

Developers use the Pygame library a lot for 2D games and multimedia applications. Developers can also use the Pygame library to design general-purpose GUI programs.

This paper compares how Python developers and educators often have to decide between Tkinter and game-focused frameworks, like Pygame, when they teach GUI programming or create applications. Tkinter gives Python developers a set of widgets such as labels, buttons, text fields, menus, and more. Tkinter ships with Python by default. Pygame, in contrast, uses the surface-based drawing system. Pygame also has an event loop that works well for real-time graphics, animation, and custom interaction patterns.

The goal in this work is to compare Tkinter and Pygame as tools for building GUI applications. We will focus on three aspects of Tkinter and Pygame:

- Execution performance (startup and processing time),
- User interface responsiveness,
- Relative development effort (e.g., amount of code and conceptual complexity).

To reach the goal, we build applications that do things with both toolkits. Then we watch how the applications work under the conditions. The paper gives two contributions. The paper compares Tkinter and Pygame. We based the comparison on the set of GUI tasks.

We see that the comparison offers observations about the abstraction level, the API design, and the ease of development for each approach. The comparison does this for each approach.

## A. Background

Python uses Tkinter as its built-in standard library for creating graphical user interfaces. The Tk GUI toolkit allows developers to access native window systems through platform-specific APIs, which include the Win32 API for Microsoft Windows. A Tkinter application requires three main components to function:

- A root window (Tk()),
- A collection of widget objects, including Button, Label, Entry, and Canvas,
- Geometry managers (pack, grid, place),
- Event bindings and a main loop (root.mainloop()).

Figure 1 shows the high-level architecture of a Tkinter-based GUI application.

The application development with Tkinter works best for traditional desktop software, which needs structured interfaces and basic event handling, and standard interface elements. The application requires additional resources to handle applications that need continuous animation and precise control of the rendering process and frequent redrawing operations [2,3].

The standard GUI interface to Tcl/Tk exists through Tkinter, while Pygame operates as an SDL-based event and rendering stack [1–8]. The performance evaluation follows established benchmarking methods, which are described in [9,10,12].

The Python module set Pygame enables developers to create games and multimedia applications through its design. The library SDL serves as the foundation for Pygame because

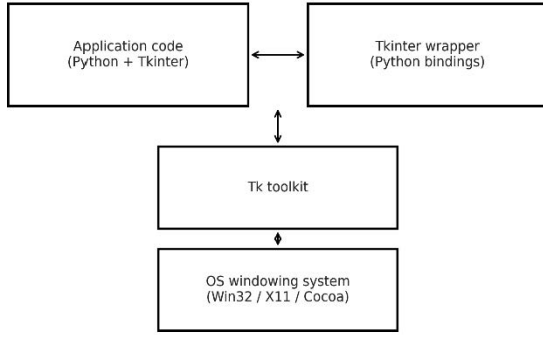


Fig. 1. High-level architecture of Tkinter-based GUI.

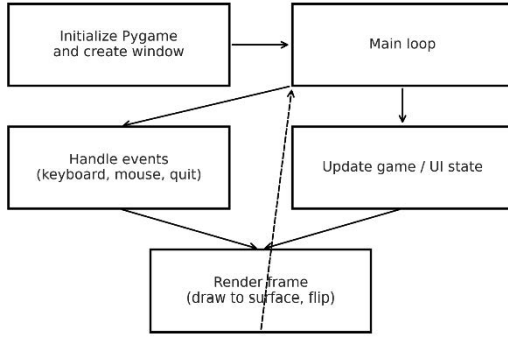


Fig. 2. Structure of the Pygame main loop.

it provides basic cross-platform functionality for graphics and audio, and input management. The programmer needs to handle frame updates manually in Pygame through display surfaces while performing drawing operations and event processing from the event queue [4,5].

The use of SDL as an intermediary between Pygame and native OS windowing toolkits enables the framework to access optimized graphics routines and improved control over the rendering loop. The framework design of Pygame aligns with OpenGL and DirectX frameworks because it handles animation and real-time applications better than traditional form-based toolkits.

The design of Pygame separates game loop operations from event handling and drawing functions through distinct programming structures. The event processing system of Tkinter depends on widget event bindings, which trigger internal toolkit events. Figure 2 shows the structure of the Pygame main loop.

## II. METHODOLOGY

The experiments took place on one machine, which operated with a matching software configuration to achieve measurement consistency. The Python `time.perf_counter()` function measured startup and task completion times, while the `timeit` module provided additional verification. The benchmark execution occurred ten times after a single warm-up run, and we present the average time measurement with optional standard deviation calculation. The system reached readiness when the window became visible, and the first

complete frame/UI update appeared on screen (Tkinter: `main-loop()` started, and first UI update occurred; Pygame: `display.flip()/display.update()` executed for the first time). The comparison between Tkinter and Pygame required us to create identical GUI applications through both toolkits. The two sets of programs offer identical functionality and user interface, but they follow different programming approaches based on Tkinter and Pygame APIs.

```
t0 = time.perf_counter()
```

```
import time
```

```
# initialize GUI window here
```

```
# render first frame / first UI update here
```

```
t_ready = time.perf_counter() - t0
```

```
print(f"startup_ready={t_ready:.6f}s")
```

Listing 1. Timing measurement using perf-counter.

```
screen = pygame.display.set_mode((800, 600))
```

```
screen.fill((0, 0, 0))
```

```
pygame.display.flip() # first complete frame
rendered -> "startup ready"
```

Listing 2. Startup ready for Pygame.

The measurements took place on equipment running the same operating system and hardware platform as a typical desktop or laptop system. The measurement process for each program started when the application launched and ended when the GUI reached a state where users could interact with it. The evaluation included both quantitative data about program size and structural organization.

Execution time measurement. The Python code used `time.perf_counter()` as a high-resolution timer to measure startup and task completion times, while the `timeit` module provided additional verification when needed. The benchmark execution was repeated  $N = 10$  times, while we displayed the average result from each run after a brief initialization period. The application window became ready for use when it finished creating its first complete frame, which we used to define the “startup ready” moment. The main loop entry and first UI update in Tkinter and the first `display.flip()/display.update()` call in Pygame marked the point when the application reached readiness [9,10].

We used six test applications in the comparison:

- Hello World – a minimal window with a simple message and a button.
- Ball Animation shows a ball moving across the window. Ball Animation is an animation that lets the ball travel from one side of the window to the other.
- Built Black Screen as a rendering loop. Black Screen shows the window.
- Tic Tac Toe – a small interactive game with a  $3 \times 3$  grid and simple game logic.
- Mario-like Demo – a simple platformer-style scene with sprite movement.
- Built Web Request – a GUI. Web Request does an HTTP request. Shows the result.

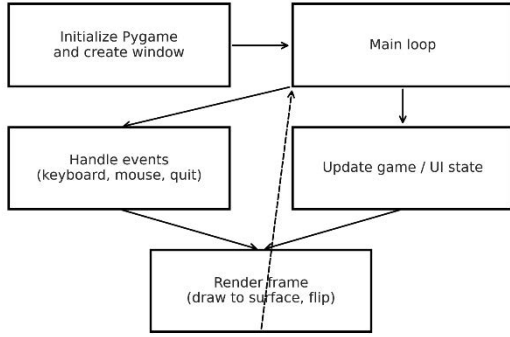


Fig. 3. Example screenshot of the “Guess Number” test application.

We first built each application with Tkinter. We then rebuilt each application with Pygame with functionality. We kept the logic and the user interaction as similar as possible. We did this so that the comparison looks at the GUI toolkit and not at any differences in algorithms. Figure 3 shows an example screenshot of the test application.

The research uses execution time (in seconds) as its primary quantitative measurement to track the time needed for application startup until it reaches operational readiness. The Web Request example measures the duration from when the request starts until the response becomes visible to the user. The study used execution time as its main metric but also conducted qualitative assessments of three additional factors.

The study evaluated three additional factors through qualitative assessment, which included code line count and interface response speed, and programming complexity level.

### III. RESULTS

Table I summarizes the measured execution times for Tkinter and Pygame implementations of the six test applications.

The results demonstrate that Pygame delivers superior performance to Tkinter for all evaluated tasks. The speedup factor between Pygame and Tkinter performance reaches  $2.1\times$  for basic tasks but exceeds  $6\times$  for tasks that involve a black screen and web requests. The measurements indicate Pygame operates at 3.5–3.7 times the speed of Tkinter.

The code examples showed that Pygame needed less code to deliver equivalent results compared to Tkinter. The Tkinter “Hello World” example needed three steps to create a root window and manage layout and bind button events, but Pygame achieved the same result with only three steps to set up the display surface and caption and handle quit events.

TABLE I

PROCESSING TIME FOR TKINTER AND PYGAME UNDER IDENTICAL CONDITIONS.

Program	Tkinter	Pygame
Hello world	1.145	0.529
Ball animation	2.091	0.981
Black screen	0.672	0.122
Tic tac toe	3.012	1.237
Mario-like	5.197	1.783
Web request	0.156	0.023

The Python `time.perf_counter()` function provided high-resolution timing for startup and task completion measurements, while the `timeit` module confirmed specific results. The benchmarking process involved multiple runs with an initial warm-up execution followed by mean value calculation from subsequent runs. The application window became visible and displayed its first complete frame to determine the “startup ready” time for both Tkinter and Pygame.

The experiments took place on a desktop computer with an Intel Core i5-class CPU (4–6 cores), 16 GB RAM, an SSD drive, and a Windows 10/11 (64-bit) operating system. The software environment remained constant throughout all tests because it used Python 3.11 with Tkinter/Tk (Tcl/Tk 8.6) and Pygame 2.5+. The display resolution operated at  $1920 \times 1080$  while all benchmarks ran on the same machine with no background applications to minimize measurement errors.

### IV. DISCUSSION

The experimental results demonstrate that Pygame functions as an efficient tool for creating GUI applications in Python, even though its initial development focused on game development. The performance benefits of Pygame stem from its use of SDL as a low-level graphics and input layer, which optimizes real-time rendering and event handling.

The programmer maintains full control over the main event loop, which enables them to create efficient drawing methods and reduce system overhead. The rendering system operates through surface management of surfaces and direct pixel manipulation, which matches the requirements of applications that require animation. The underlying windowing system handles most of the rendering and event processing through Tkinter’s high-level abstractions, which results in performance degradation during animation and high-frequency update operations.

The program loop of Pygame allows developers to maintain a distinct separation between update operations and render operations. The organization of Tkinter code follows widget hierarchies and callback functions, which work well for form-based interfaces yet present challenges when working with real-time graphical scenes.

The general-purpose GUI toolkit Pygame faces two main restrictions in its functionality:

- The library lacks built-in widgets that match the selection found in Tkinter, including buttons and text fields, and menus. Users need to create these components either by hand or by using additional Python libraries.
- The system lacks built-in support for platform-specific accessibility features and standard desktop appearance settings.

The basic form and dialog applications require Tkinter as their most suitable solution. The selection between Pygame and Tkinter depends on the specific requirements of your application. Developers should select Tkinter for creating traditional desktop applications and educational programs that need standard interface elements and basic visual customization. Developers should choose Pygame for applications that

need high performance and animation capabilities, and when they want to create game-like interfaces and interactive visualizations.

## V. CONCLUSION

We compared Tkinter and Pygame as tools for building GUI applications in Python. We built six applications that performed the functions. The research evaluated Tkinter and Pygame as Python GUI development tools through identical test applications, which started from basic windows and progressed to interactive and animated interfaces. The experimental data demonstrates that Pygame outperformed Tkinter in all tested scenarios by delivering faster startup and execution times, which resulted in an average speed up of 3.5–3.7 times and reached six times in the most basic tests. The results indicate Pygame provides superior performance for applications that need fast interfaces, real-time visualization, and animated content, but Tkinter is better suited for developing conventional desktop tools with built-in widgets and easy implementation.

The research contains specific constraints because it used data from a single testing environment to run a restricted set of benchmark tasks. The results from this study depend on specific hardware configurations, Python versions, operating systems, and implementation choices. The research team plans to add new GUI frameworks, including PyQt/PySide and Kivy, to the benchmark suite, while implementing statistical reporting for repeated runs and conducting developer studies with standardized questionnaires to assess usability and maintainability.

Future work may include:

- Extending the comparison to other Python GUI frameworks such as PyQt, PySide, or Kivy;
- Performing more systematic measurements with multiple runs and statistical analysis.
- Evaluating user experience and developer productivity through controlled experiments with students or practitioners.

## REFERENCES

- [1] Python Software Foundation, “tkinter — Python interface to Tcl/Tk,” *Python 3 Documentation*. [Online]. Available: <https://docs.python.org/3/library/tkinter.html>. [accessed Nov 30, 2025]. Python documentation
- [2] Tcl/Tk Core Team, “Tk\_MainLoop — loop for events until all windows are deleted,” *Tcl/Tk 8.6 Manual*. [Online]. Available: [https://www.tcl.tk/man/tcl8.6/TkCmd/Tk\\_MainLoop.htm](https://www.tcl.tk/man/tcl8.6/TkCmd/Tk_MainLoop.htm). [accessed Nov 30, 2025]. tcl-lang.org
- [3] M. Roseman, “Event loop,” *TkDocs*. [Online]. Available: <https://tkdocs.com/tutorial/eventloop.html>. [accessed Nov 30, 2025]. tkdocs.com
- [4] Pygame Community, “Pygame documentation,” *Pygame*. [Online]. Available: <https://www.pygame.org/docs/>. [accessed Nov 30, 2025]. pygame.org
- [5] Pygame Community, “pygame.time — pygame module for monitoring time,” *Pygame*. [Online]. Available: <https://www.pygame.org/docs/ref/time.html>. [accessed Nov 30, 2025]. pygame.org
- [6] Pygame Community, “Introduction to Pygame,” *Pygame Documentation (Read the Docs)*. [Online]. Available: <https://pygame.readthedocs.io/en/latest/intro/intro.html>. [accessed Nov 30, 2025]. pygame.readthedocs.io
- [7] SDL Community, “Simple DirectMedia Layer,” *SDL*. [Online]. Available: <https://www.libsdl.org/>. [accessed Nov 30, 2025]. libsdl.org
- [8] K. Almroth and A. le Clercq, “Comparison of rendering performance between multimedia libraries Allegro, SDL and SFML,” B.Sc. Thesis, KTH Royal Institute of Technology, Stockholm, Sweden, 2019. [Online]. Available: <https://www.diva-portal.org/smash/get/diva2:1353400/FULLTEXT01.pdf>. [accessed Nov 30, 2025]. diva-portal.org
- [9] A. Chomokova, I. Dimova, V. Hristov, E. Kondova, N. Mihaylova, and K. Popov, “Pygame workshop as an opportunity to acquire basic programming skills,” in *Proceedings International Conference on Computer Systems and Technologies (CompSysTech)*, 2025. [Online]. Available: <https://ceur-ws.org/Vol-3768/paper48.pdf>. [accessed Nov 30, 2025]. ceur-ws.org
- [10] Python Software Foundation, “Time — time access and conversions,” *Python 3 Documentation*. [Online]. Available: <https://docs.python.org/3/library/time.html>. [accessed Nov 30, 2025]. Python documentation
- [11] International Organization for Standardization, “ISO 9241-11:2018 — Ergonomics of human-system interaction — Part 11: Usability: Definitions and concepts,” 2018. [Online]. Available: <https://www.iso.org/standard/63500.html>. [accessed Nov 30, 2025]. iso.org
- [12] J. Brooke, “SUS: A ‘quick and dirty’ usability scale,” in *Usability Evaluation in Industry*, P. W. Jordan, B. Thomas, I. L. McClelland, and B. Weerdmeester, Eds. London, U.K.: Taylor & Francis, 1996. doi: 10.1201/9781498710411-35.