pygame.time

pygame module for monitoring time

- pygame.time.get_ticks — get the time in milliseconds
- pygame.time.wait — pause the program for an amount of time
- pygame.time.delay — pause the program for an amount of time
- pygame.time.set_timer — repeatedly create an event on the event queue
- pygame.time.Clock — create an object to help track time

Times in pygame are represented in milliseconds (1/1000 seconds). Most platforms have a limited time resolution of around 10 milliseconds. This resolution, in milliseconds, is given in the TIMER_RESOLUTION constant.

pygame.time.get_ticks()

get the time in milliseconds

get_ticks() -> milliseconds

Return the number of milliseconds since pygame.init() was called. Before pygame is initialized this will always be 0.

pygame.time.wait()

pause the program for an amount of time

wait(milliseconds) -> time

Will pause for a given number of milliseconds. This function sleeps the process to share the processor with other programs. A program that waits for even a few milliseconds will consume very little processor time. It is slightly less accurate than the pygame.time.delay() function.

This returns the actual number of milliseconds used.

pygame.time.delay()

pause the program for an amount of time

delay(milliseconds) -> time

Will pause for a given number of milliseconds. This function will use the processor (rather than sleeping) in order to make the delay more accurate than pygame.time.wait().

This returns the actual number of milliseconds used.

pygame.time.set_timer()

repeatedly create an event on the event queue

set_timer(eventid, milliseconds) -> None

Set an event type to appear on the event queue every given number of milliseconds. The first event will not appear until the amount of time has passed.

Every event type can have a separate timer attached to it. It is best to use the value between pygame.USEREVENT and pygame.NUMEVENTS.

To disable the timer for an event, set the milliseconds argument to 0.

pygame.time.Clock

create an object to help track time

Clock() -> Clock

- pygame.time.Clock.tick — update the clock
- pygame.time.Clock.tick_busy_loop — update the clock
- pygame.time.Clock.get_time — time used in the previous tick
- pygame.time.Clock.get_rawtime — actual time used in the previous tick
- pygame.time.Clock.get_fps — compute the clock framerate

Creates a new Clock object that can be used to track an amount of time. The clock also provides several functions to help

control a game's framerate.

tick()

update the clock

tick(framerate=0) -> milliseconds

This method should be called once per frame. It will compute how many milliseconds have passed since the previous call.

If you pass the optional framerate argument the function will delay to keep the game running slower than the given ticks per second. This can be used to help limit the runtime speed of a game. By calling Clock.tick(40) once per frame, the program will never run at more than 40 frames per second.

Note that this function uses SDL_Delay function which is not accurate on every platform, but does not use much CPU. Use tick_busy_loop if you want an accurate timer, and don't mind chewing CPU.

tick_busy_loop()

update the clock

tick_busy_loop(framerate=0) -> milliseconds

This method should be called once per frame. It will compute how many milliseconds have passed since the previous call.

If you pass the optional framerate argument the function will delay to keep the game running slower than the given ticks per second. This can be used to help limit the runtime speed of a game. By calling Clock.tick_busy_loop(40) once per frame, the program will never run at more than 40 frames per second.

Note that this function uses pygame.time.delay()pause the program for an amount of time, which uses lots of CPU in a busy loop to make sure that timing is more accurate.

get_time()

time used in the previous tick

get_time() -> milliseconds

The number of milliseconds that passed between the previous two calls to Clock.tick().

get_rawtime()

actual time used in the previous tick

get_rawtime() -> milliseconds

Similar to Clock.get_time(), but does not include any time used while Clock.tick() was delaying to limit the framerate.

get_fps()

compute the clock framerate

get_fps() -> float

Compute your game's framerate (in frames per second). It is computed by averaging the last ten calls to Clock.tick().