pygame时钟

pygame的时钟也是一个类，它在`pygame.time`中，比如创建一个时钟对象:

```
fclock = pygame.time.Clock()
```

这样我们就有了一个名字叫fclock的时钟对象。它拥有一个我们常用的方法:

- **tick(framerate=0)->milliseconds**

  作用：更新时钟。这个方法应该每帧都被运行一次，它返回的是从上次这个方法被调用到这次被调用所经历的时间。如果你使用了framerate这个参数(比如fclock.tick(60))，那么这个函数就会造成延迟，以确保你的游戏运行帧率不会比你传进去的framerate高。所以简而言之就是确定游戏帧率上限。所以我们只需要定义一个常量FPS，再在游戏主循环里加上一句

  ```
  fclock.tick(FPS)
  ```

  就可以让游戏按照固定的帧率运行了。

  以下为原文:

  > **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.

其余的方法原文po在下面，感兴趣的同学可以自行学习或者去官网查看详情:

> - **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.

New in pygame 1.8.

- **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()`.