**Writeup for Homework 2**

**200333470**

**yhuang64**

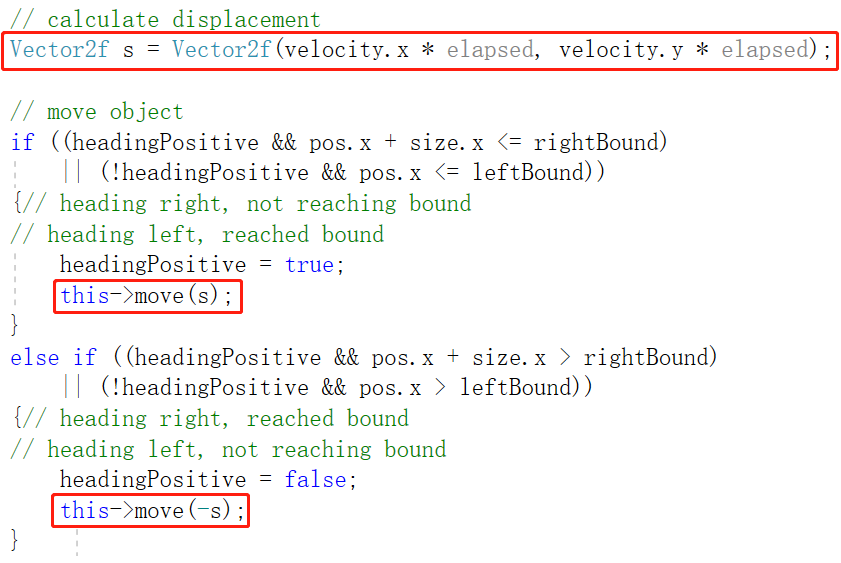
**Yuanmin Huang**

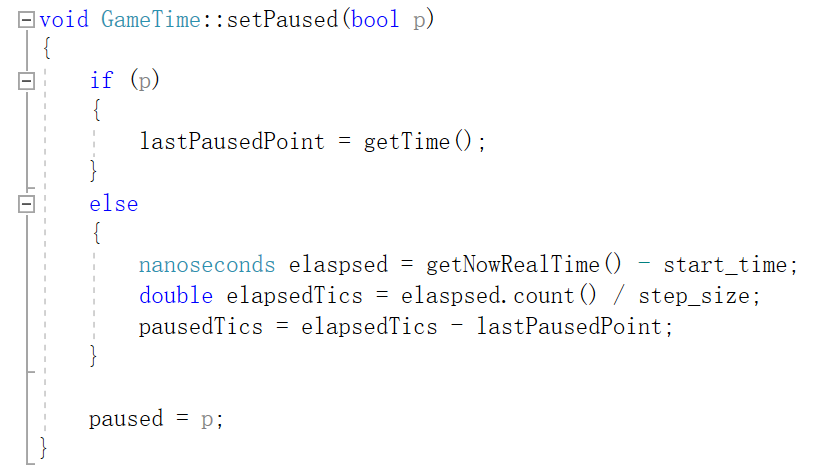
**Section 1**

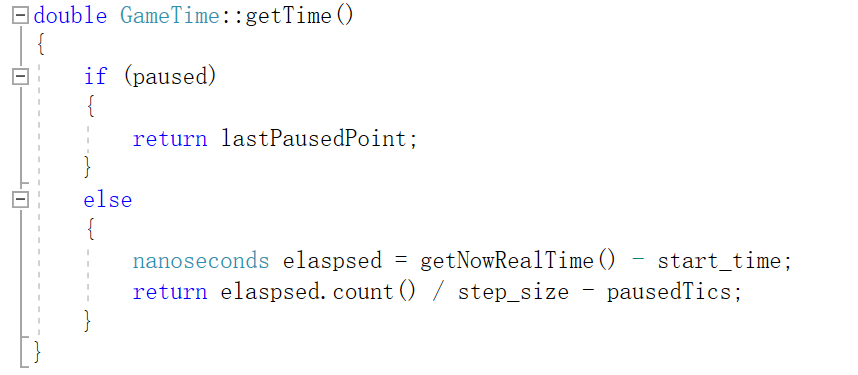
The first section asked us to implement a kind of timeline that was scalable, pausable and on which the movement of a movable object could rely. Thus, I would divide it into three subsections to conquer.

Firstly, it would be refactoring all the movements of movable objects.

Before that, I created three timeline classes according to the notes taken in class. They are Timeline, GameTime and LocalTime. There was one thing to make clear that there was a little point Dr. Roberts had made wrong: the duration\_cast function takes a duration as parameter instead of a time\_point like system\_clock::now(). Thus, I used now().time\_since\_epoch() to solve the problem. Or, it could be solved by take the difference of now() and a time\_point of value 0.

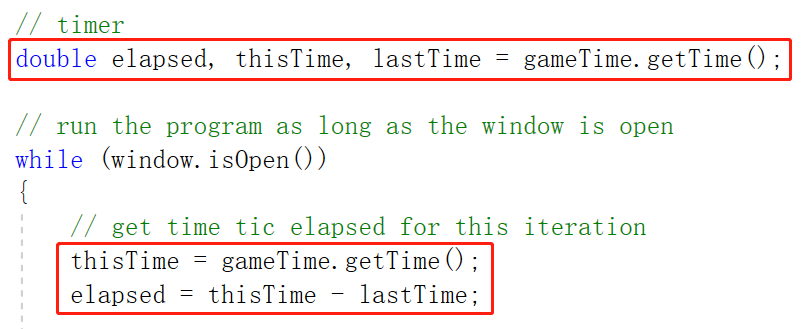
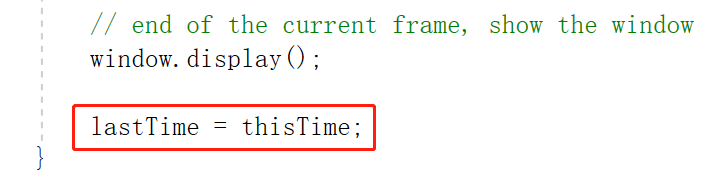
Concerning the refactoring of the movement, I need to refactor the update() function of all subclasses of Movable. I did have a velocity for or movable objects, but my movement was based on iterations of the main loop. So, this time, I must change its foundation to the time elapsed. At first, I added a new attribute *lastTime* to class Movable, which was maintained to help get the time elapsed between each iteration, so that the displacement of objects could be calculated. During the completion of scalable timeline, I moved the attribute to main() as a variable to realize the desired functionality, which I would explain later. Whichever way of implementation is used, the result is the same: getting the displacement of objects and move them accordingly. I also made some changes to the value of the velocity of objects to reach a decent effect of moving based on elapsed time.

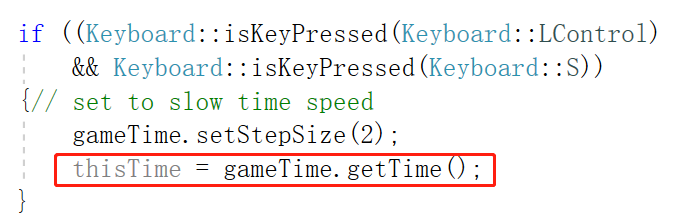
Secondly, I implemented the functionality of pause operation. To achieve this, I added bool paused, double pausedTics and double lastPausedPoint to class Timeline to indicate the status of timeline, the paused time tics and the last time point a pause command was triggered. Moreover, I added a setPaused() function and modified the implementation of getTime().

Take class GameTime as an example, when a pause is triggered, the function setPaused() calls getTime() to record the paused time point and sets *paused* to true. Then, getTime() would always return the paused time point when the timeline was paused, so that all moving objects wouldn’t move because the time elapsed would be 0.

When an un-pause is triggered, function setPaused() would update the paused tics and set *paused* to false, after which calling getTime() would get virtual game time as real time elapsed minus the paused tics.

There is one more thing to point out for pausing is that I made the program to sleep 0.2 second to prevent multi-triggered pause commands. This is a tradeoff because I must suffer from either disordered paused commands or slightly stuck game experience.

Thirdly, it’s the scalable part, whose goal is that the bigger the *step\_size* is, the slower the time flies in game. To implement this, the design in Timeline class is trivial, which is adding a setStepSize() function which resets *step\_size*. However, I met with a problem that the moving objects would fly away right after resetting the *step\_size*, which I found out was caused by the sudden change of the return value of getTime(), which caused the sudden change of elapsed time. Therefore, I pulled the attribute *lastTime* out to main() and maintain *thisTime (got from calling getTime() at the beginning of each iteration)* and *lastTime* every iteration of the main loop (shown in the picture above). Then, I just need to pass the elapsed time to update() to help calculate the displacement. Moreover, at the end of each iteration, I should assign *thisTime* to *lastTime*.

Here comes the non-trivial work. After resetting the *step\_size*, *thisTime* should be reassigned by calling getTime() to narrow the gap between *lastTime (would be assigned thisTime at the end of this iteration)* the new *thisTime* get in the next iteration.

The last work is to set up a mapping of keys and new operations (pause/un-pause, set time speed), which is easy to implement and would be written into README.txt.