Dual State Framework

Functional Specification

Yu Chen - C00151352

2015/04/12 21:36:45

Contents

1 Introduction

The document describes all functionalities estimated to be implemented for the project. It also shows what level of quality of the project should be achieved. Project plan and schedule are embedded in this document.

1.1 Purpose

The main purpose of my project is to create a C++ framework for parallel computing. Parallel computing is the science and art of programming computers that can do more than one operation at once during the same cycle, simultaneously and concurrently. It performs often via having more than one processor.

Next, A benchmark program is required for profiling this framework. The program will be designed as a GUI application which shows the performance in different situations

1.2 Plan

Dual State Framework will be developed by C++ under Mac Os X. OpenMP or Intel TBB is used for implementing Parallel Computing. In addition, it will be compiled and debugged both on Microsoft Windows and Linux as well. This process will be done by Cmake. The API Documentation of this project will be created by Doxygen.

The simple benchmark program will be written by C++ with SFML or SDL library. Git will be used for source control and version control. The project is now available in following link.

https://github.com/kuyoonjo/DualStateFramework

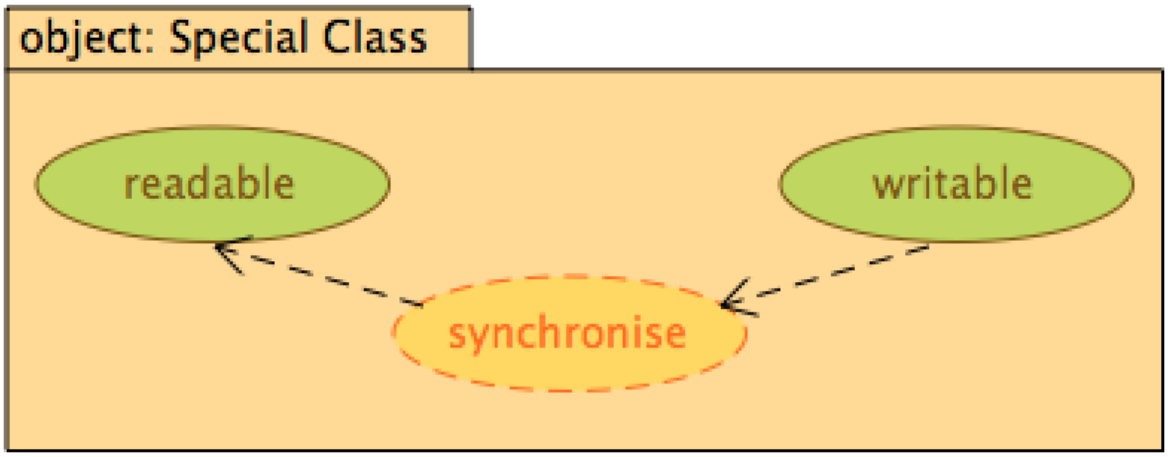
1.3 Potential Users

Dual State Framework can have a variety of uses. Initially the project is concerned around computer games, but it can be used in broad kinds of software. The framework can be used in any program or game that runs parallel processes. Currently almost all of them meet that criteria.

2 | Functionalities

2.1 Dual State

A special class should be designed for the framework. Each object of the class has two states for read operation and write operation. The purpose of the class is for thread security. In each cycle, the state of read operation should not be changed. Users can only change the state of write operation. After this cycle, these two states will be synchronised.



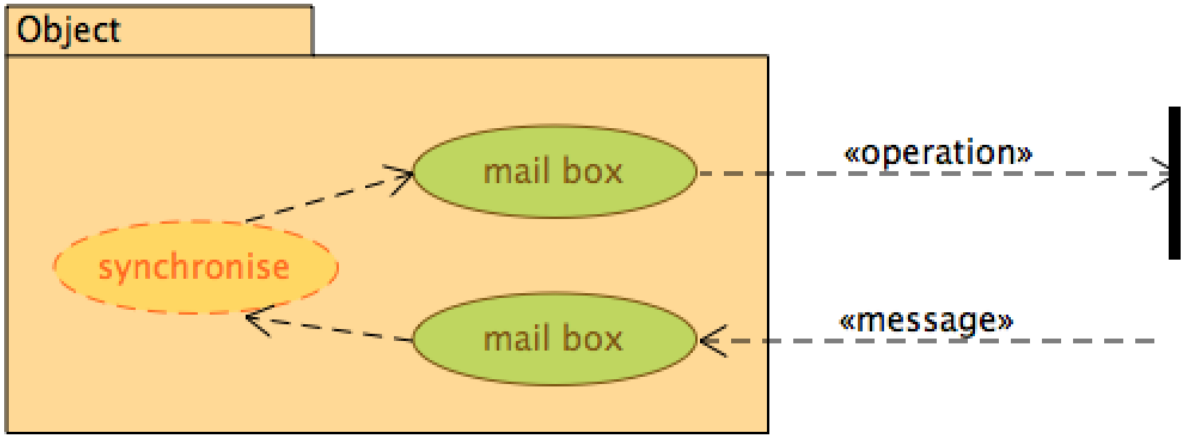
2.2 Message Delivery

Objects in the framework should be able to send messages to other objects. Each message contains: where this message sent from

- operation
- arguments for the operation

2.3 Mailbox

The mailbox stores messages that object received. It should be designed as Dual State. Therefore, each object will have two copies of mailbox. One is for performing the operation, and the other one is for receiving messages.


Object
mail box
synchronise
mail box
«operation»
«message»

3 Benchmark

3.1 Graphics User Interface

The benchmark program should be designed as a GUI application. Also, it should allow users to configure detail settings such as:

- Max number of threads
- Number of objects
- Methods
- Duration

3.2 Methods

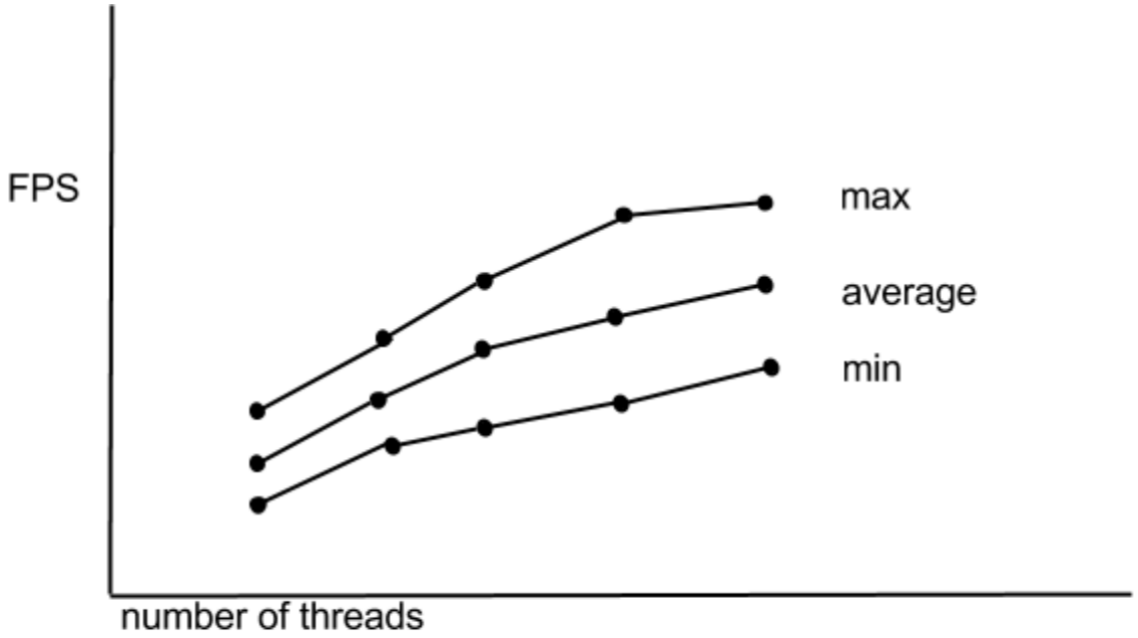
The program should has more than one methods to profile the framework. The following algorithm is recommended.

- Random objects
  - Objects appear in random positions
  - Positions change every frame
- Collision
  - Objects locate to random position
  - Each object has a random velocity
  - After collision, objects get new velocities
- Flocking boids
  - Separation - avoid crowding neighbors
  - Alignment - steer towards average heading of neighbors
  - Cohesion - steer towards average position of neighbors

3.3 Outputs

The output will be a graph which shows all information of the benchmark.

- x axis representing number of threads
- y axis representing frames per second(FPS)
- values with the maximum, the minimum, and the average.


FPS
max
average
min
number of threads

4 | FURPS

4.1 Functionality

The framework will run on multiple platforms. Source code will be able to be compiled on varies of C++ compilers. In addition, installers for different operating system will be created. Popular IDE and OS will be supported.

- Mac OS X - Xcode
  - .dylib dynamic library
  - .framework bundled library
- Windows - Visual Studio
  - .lib compiling time dynamic library
  - .dll runtime dynamic library
- Linux - Makefile
  - .so dynamic library

4.2 Usability

User does not need to know the code behind the scene because that is not important. User who has experienced on C++ should be easy to understand the framework. An API document should be created. The document should contain: Installation

- Namespace Documentation
- Class Documentation
- Simple examples

4.3 Reliability

The framework should run correctly after installation. Memory leaks should not occur at runtime. Deadlock or livelock is not allowed. A deadlock is where a system locks up because two or more processed are waiting for each other to finish. Livelock is similar to deadlock, but except that the states of the processes involved in the livelock constantly change with regard to one another.
To know more about memory leak, deadlock and livelock requires a high level of programming skill that scriptures cannot be assumed to have. The model allows the user do not need to worry about these issues.

4.4 Performance

The best performance of this framework should be based on the running machine. For example, the CPU of my laptop is intel i5, which has two cores and executes four threads. The best performance of my laptop is when I use this framework with four threads.
The framework should not consume too much resources(CPU, RAM).

4.5 Supportability

Ensure that the product can be localized later for different languages and units later on. The maintenance of the system shall be done as per the maintenance contract. Update will be display when it is available.

5 | Project Plan

5.1 Implementation

SCRUM agile development methodology will be used for the implementation phase of the project. It is a great way to keep focused on the individual tasks that need to be completed in order make the project a success. Due to short sprints and constant feedback, it becomes easier to cope with the changes. Changes to be made to the project will not be a big impact on the time schedule for the project.

The duration for the development of the project will be broken down into sprints. The length of sprint for this project will be one week. In each sprint, a set amounts of task are expected to be completed. At the end of each sprint, how far the goal achieved, and how many issues met will be demonstrated to supervisor.

5.2 Documents

The following documents will be done during the time scope:

- Vision Documentation
  - It estimates the value and the risk of the project.
  - It defines the stakeholders and users.
- Functional Specification
  - It describes functionalities needed by the system.
- Design Documentation
  - It shows the design for implementing functionalities that Functional Specification defined.
- Research Documentation
  - It shows technologies obtained by doing the project.
- Technical Manual
  - It demostrates how to use the framework.

5.3 Schedule

- Sprint 1: Generate the Vision Document.
- Sprint 2: Finish the Vision Document, and start the Functional Specification.
- Sprint 3: Finish the Functional Specification, and start the Research Document
- Sprint 4: Start the Design Document

- Sprint 5: Finish the Design Document, and setup develop environment
- Sprint 6: Start coding the main framework. Define interfaces.
- Sprint 7: Implement interfaces defined.
- Sprint 8: Finish the first version.
- Sprint 9: Test the first version.
- Sprint 10: Start coding the profiler
- Sprint 11: Implement the algorithm "Random"
- Sprint 12: Implement the algorithm "Collision"
- Sprint 13: Implement the algorithm "Flocking"
- Sprint 14: Generate the API Document
- Sprint 15: Hand up the final report