STM for games

Design Manual

Author: Arkadiusz Bielecki
ID: C00139358
Date: 10.01.2013 Project Supervisor: Joseph Kehoe

Table of Contents

1. Introduction

The Design Manual is showing the Model of my application. All the design needs to be described and planned carefully in order to avoid unwanted misunderstandings during the development. I will describe the design of graphical user interface (GUI), all of the algorithms that will be used and metrics - how the measurements will be calculated and shown on the diagram.

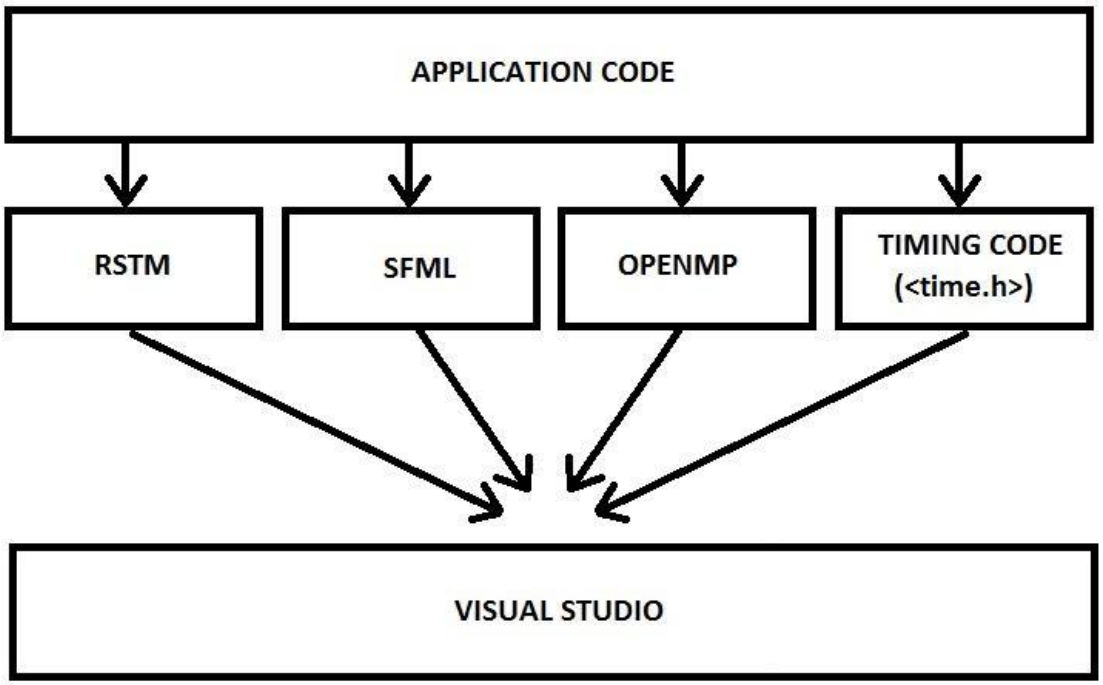
The purpose of my project is to show how Software Transactional Memory (STM) (a concurrency control mechanism) will affect the performance in gaming. I will implement Simple and Fast Multimedia Library on a game, using a standard synchronization, then using various STM algorithms, benchmark them, and compare whether the performance have increased or not.

2. GUI Design

Graphical User Interface that will be used in my project will be kept as simple as possible. The application can't take up too much memory, and doesn't need to look fancy, so I have decided to give it a simple, but at the same time modern look. The main window of my project will have various check boxes - so a user can select desired window size, and what algorithms to measure. The output of my app will show a graph appropriate to the selected options.

3. Model of the Application

Main interaction model of the application is shown on a diagram below:

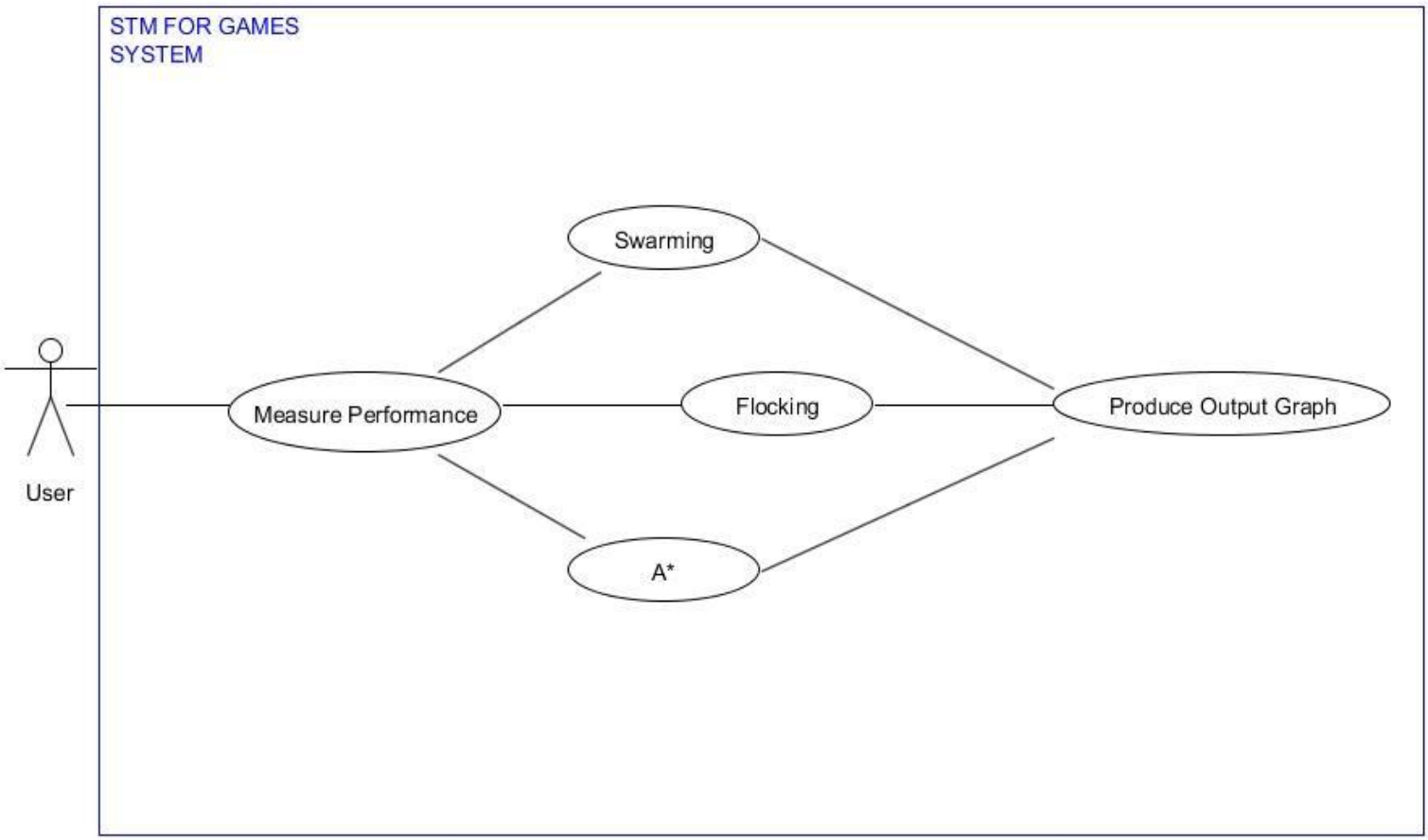


Code will be written on top of **RSTM** and **SFML** algorithms, then implemented in Visual Studio in order to measure the performance and produce the output.

OpenMP will be used to run a specific configuration in parallel threads, it's an API that allows code to run in parallel, under different hardware configurations.

Timing Code will be implemented in order to measure how much time it took to run a specific configuration.

4. Use Case Diagram



5. Use Cases

Brief Use Cases

Measure Swarming Performance

Actors: User

Description: This use case begins when a user wants to use measure system performance using the Swarming algorithm. The user selects appropriate configuration (Swarming Performance and board size). The application will test the code and inherit various testing methods from RSTM Library. Time will be measured, by inheriting from the <time.h> library. Appropriate values will be saved and shown on the output graph, which ends this use case.

Measure Flocking Performance

Actors: User
Description: This use case begins when a user wants to use measure system performance using the Swarming algorithm. The user selects appropriate configuration (Flocking Performance and board size). The application will test the code and inherit various testing methods from RSTM Library. Time will be measured, by inheriting from the <time.h> library. Appropriate values will be saved and shown on the output graph, which ends this use case.

Measure A* Performance

Actors: User
Description: This use case begins when a user wants to use measure system performance using the Swarming algorithm. The user selects appropriate configuration (A* search and board size). The application will test the code and inherit various testing methods from RSTM Library. Time will be measured, by inheriting from the <time.h> library. Appropriate values will be saved and shown on the output graph, which ends this use case.

Detailed Use Cases

Name: Measure Swarming Performance
Actors: User

Main Scenario

1. The use case begins when a new performance measurement, using the Swarming algorithm, needs to be made.
2. The User selects "Measure Performance".
3. User can browse through available algorithms and board size configurations, which are shown on the screen.
4. User can select "Swarming" checkbox.
5. User can select size of the game-board from a drop-list.
6. The selected algorithm and board size are displayed in the screen.
7. User can change selected algorithm and board size before submitting.
8. When the configuration is ready, User can choose to confirm the selections, and calculate the output.

9. Valid graph is shown on the screen, showing appropriate values, according to the selected properties.
10. The use case ends when the user leaves the functionality.

Alternatives.
7a. When the user wants to measure performance under a different algorithm, he can change by ticking the appropriate checkboxes.

Name: Measure Flocking Performance
Actors: User

Main Scenario

1. The use case begins when a new performance measurement, using the Flocking algorithm, needs to be made.
2. The User selects "Measure Performance".
3. User can browse through available algorithms and board size configurations, which are shown on the screen.
4. User can select "Flocking" checkbox.
5. User can select size of the game-board from a drop-list.
6. The selected algorithm and board size are displayed in the screen.
7. User can change selected algorithm and board size before submitting.
8. When the configuration is ready, User can choose to confirm the selections, and calculate the output.
9. Valid graph is shown on the screen, showing appropriate values, according to the selected properties.
10. The use case ends when the user leaves the functionality.

Alternatives.
7a. When the user wants to measure performance under a different algorithm, he can change by ticking the appropriate checkboxes.

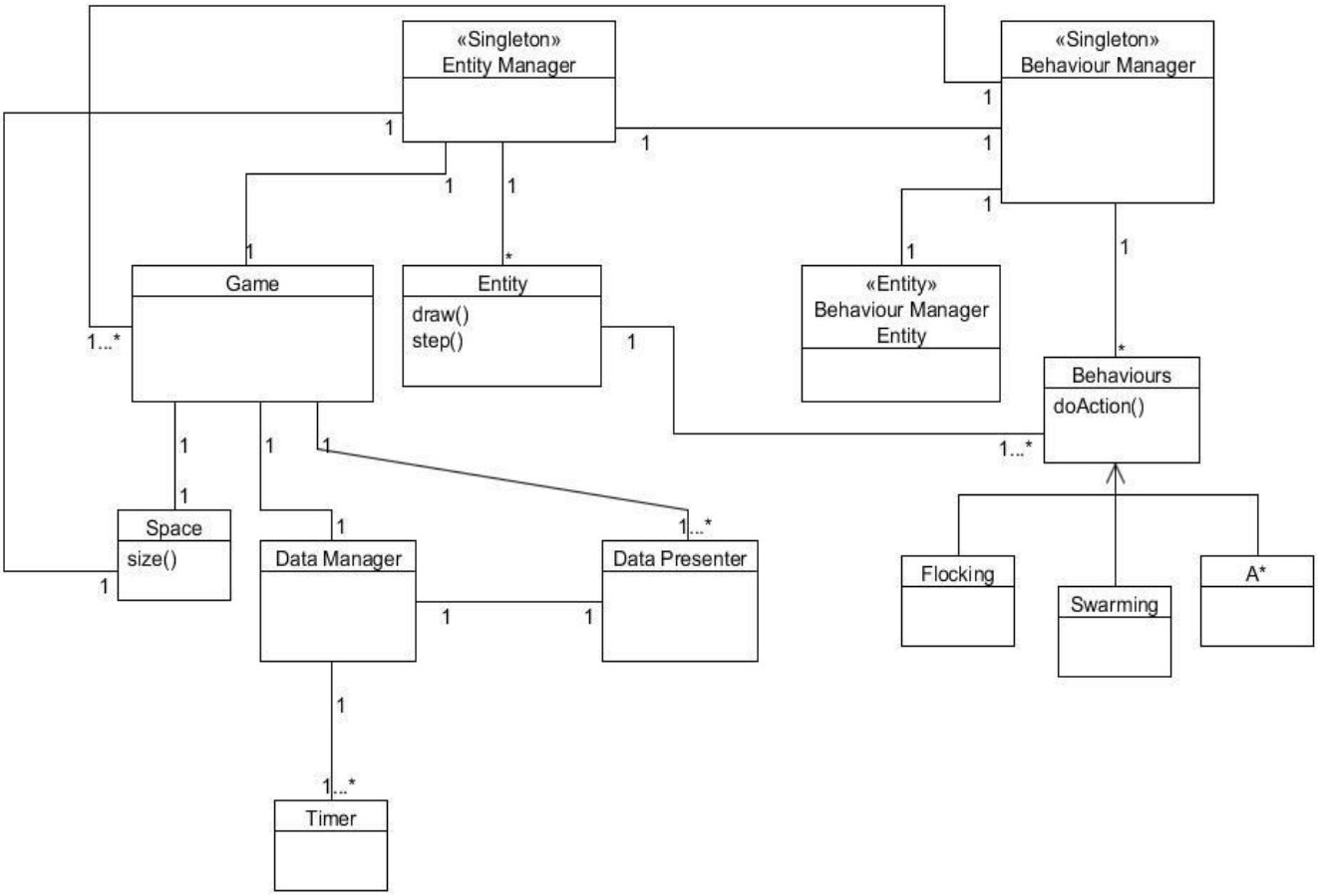
Name: Measure A* Performance
Actors: User

Main Scenario

1. The use case begins when a new performance measurement, using the Swarming algorithm, needs to be made.
2. The User selects “Measure Performance”.
3. User can browse through available algorithms and board size configurations, which are shown on the screen.
4. User can select “A*” checkbox.
5. User can select size of the game-board from a drop-list.
6. The selected algorithm and board size are displayed in the screen.
7. User can change selected algorithm and board size before submitting.
8. When the configuration is ready, User can choose to confirm the selections, and calculate the output.
9. Valid graph is shown on the screen, showing appropriate values, according to the selected properties.
10. The use case ends when the user leaves the functionality.

Alternatives.
7a. When the user wants to measure performance under a different algorithm, he can change by ticking the appropriate checkboxes.

6. Domain Model



Singleton class Behaviour Manager inherits behaviours from appropriate algorithms (Flocking, Swarming, A* - using the RSTM library), and also from Behaviour Manager Entity.

Singleton class Entity Manager inherits from Game and Entity class.

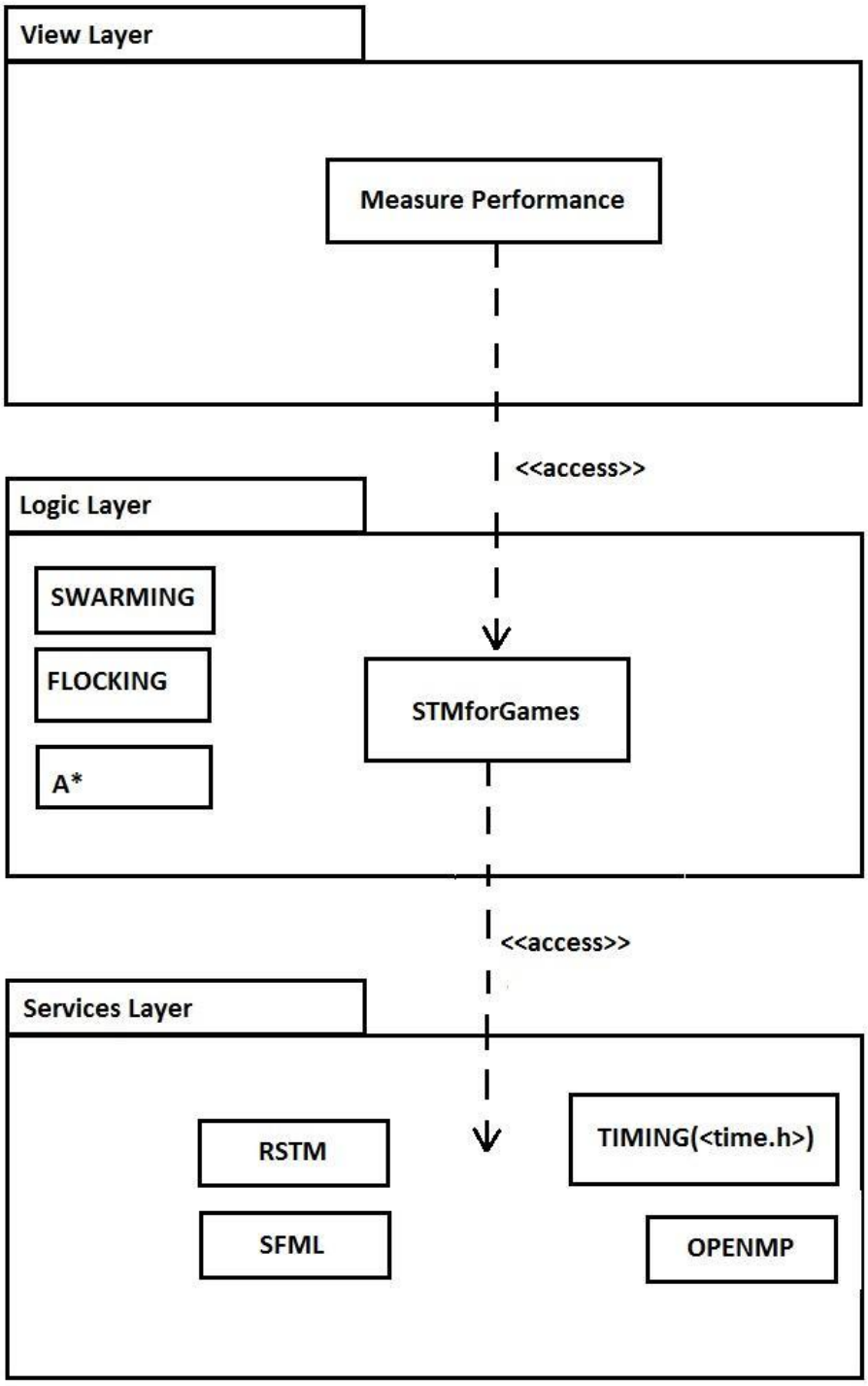
Time is taken from <time.h> library, and then passed to the Game class.

Board size is also passed to the Game class.

OPENMP will be used to run the Algorithms under different processor core configurations.

Behaviour Manager class collects the data, and passes it on the output graph.

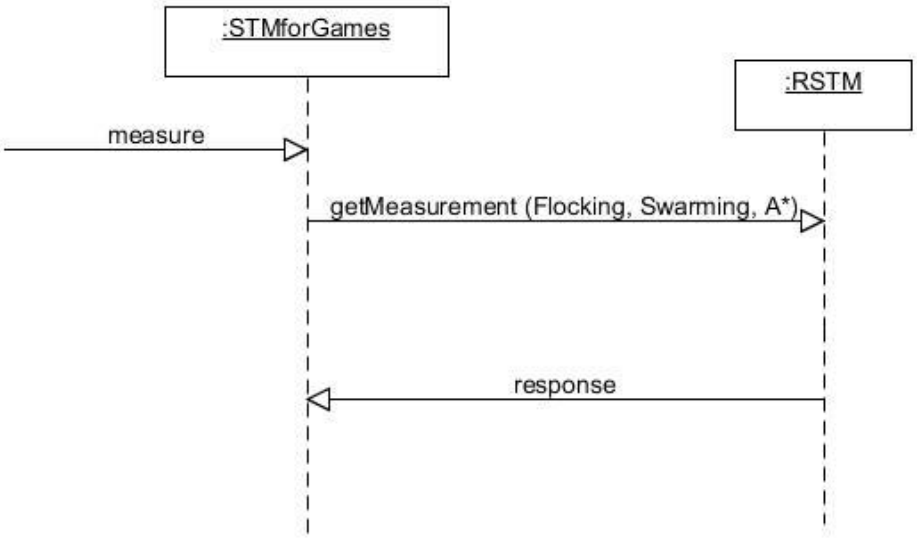
7. Layers


View Layer
Measure Performance
<<access>>
Logic Layer
SWARMING
FLOCKING
A*
STMforGames
<<access>>
Services Layer
RSTM
SFML
TIMING(<time.h>)
OPENMP

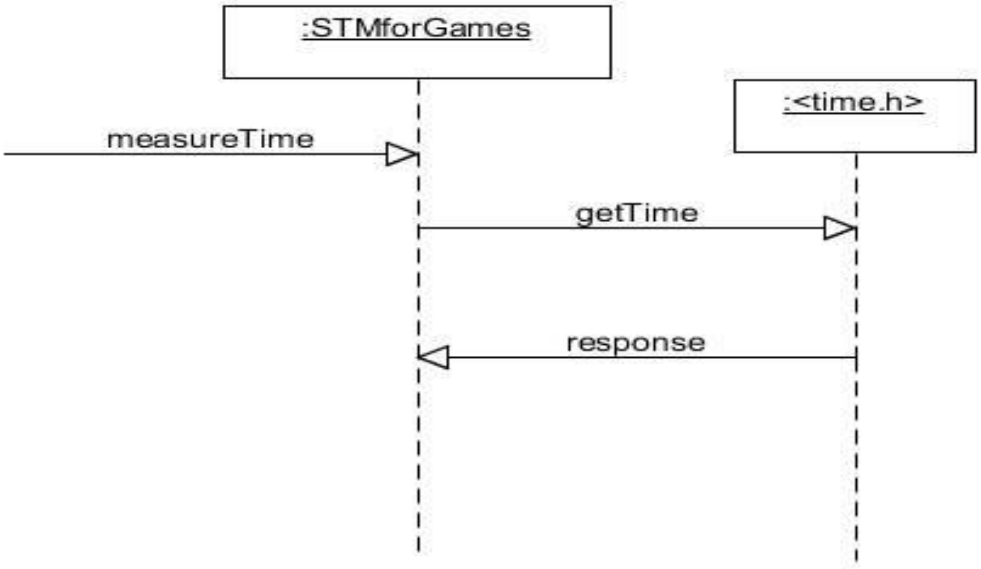
8. Communication

Communication between main application and its components is explained and shown on diagrams below.

Diagram[1] shows communication between components when a measurement is requested (using appropriate algorithm).



Diagram[2] shows communication between components when a time measurement is



requested.

Diagram[3] shows communication between components when a measurement is requested. to run on multiple cores