

Dual State Framework

CODE

Yu Chen - C00151352

2015/04/15 17:53:03

Contents

1	Dual State Framework	1
1.1	CMakeLists	1
1.1.1	CMakeLists.txt	1
1.1.2	dsf/CMakeLists.txt	2
1.2	Config	2
1.2.1	Config.h	3
1.3	Declaration	3
1.3.1	Declaration.h	3
1.4	DualStateFramework	4
1.4.1	DualStateFramework.h	4
1.4.2	DualStateFramework.cpp	4
1.5	Export	6
1.5.1	Export.h	6
1.6	Lock	6
1.6.1	Lock.h	6
1.6.2	Lock.cpp	7
1.7	Runnable	7
1.7.1	Runnable.h	7
1.8	Synchronisable	8
1.8.1	Synchronisable.h	8
1.9	SynchronizedObject	8
1.9.1	SynchronizedObject.h	8
1.9.2	SynchronizedObject.cpp	9
1.10	SynchronizedVar	10
1.10.1	SynchronizedVar.h	10
1.10.2	SynchronizedVar.cpp	10
1.11	Task	11
1.11.1	Task.h	11
1.11.2	Task.cpp	11
1.12	TaskArgument	12
1.12.1	TaskArgument.h	12

1.13	TaskBox	12
1.13.1	TaskBox.h	12
1.13.2	TaskBox.cpp	13
1.14	TaskFunction	14
1.14.1	TaskFunction.h	14
2	yctools	15
2.1	Any	15
2.1.1	Any.h	15
2.1.2	Any.cpp	16
2.2	AnyException	17
2.2.1	AnyException.h	17
2.2.2	AnyException.cpp	17
2.3	Config	18
2.3.1	Config.h	18
2.4	Export	18
2.4.1	Export.h	19
2.5	Random	19
2.5.1	Random.h	19
2.5.2	Random.cpp	19
3	Benchmark Program	21
3.1	CMakeLists	21
3.1.1	CMakeLists.txt	21
3.2	BouncingCircleManager	23
3.2.1	BouncingCircleManager.h	23
3.2.2	BouncingCircleManager.cpp	23
3.3	DSFSFML	25
3.3.1	DSFSFML.h	25
3.3.2	DSFSFML.cpp	25
3.4	FlockingBoidManager	26
3.4.1	FlockingBoidManager.h	26
3.4.2	FlockingBoidManager.cpp	26
3.5	FPS	27
3.5.1	FPS.h	27
3.5.2	FPS.cpp	28
3.6	MyDSF	28
3.6.1	MyDSF.h	29
3.6.2	MyDSF.cpp	29
3.7	RandomCircleManager	32
3.7.1	RandomCircleManager.h	32

3.7.2	RandomCircleManager.cpp	33
3.8	ResourcePath	34
3.8.1	ResourcePath.hpp	34
3.8.2	ResourcePath.cpp	34
3.8.3	ResourcePath.mm	34
3.9	SyncBouncingCircle	35
3.9.1	SyncBouncingCircle.h	35
3.9.2	SyncBouncingCircle.cpp	36
3.10	SyncCircle	37
3.10.1	SyncCircle.h	37
3.10.2	SyncCircle.cpp	37
3.11	SyncFlockingBoid	37
3.11.1	SyncFlockingBoid.h	37
3.11.2	SyncFlockingBoid.cpp	38
3.12	SyncVector3D	40
3.12.1	SyncVector3D.h	40
3.12.2	SyncVector3D.cpp	41
3.13	Vector3D	41
3.13.1	Vector3D.h	41
3.13.2	Vector3D.cpp	42
3.14	main	44
3.14.1	main.cpp	44

1 | Dual State Framework

1.1 CMakeLists

1.1.1 CMakeLists.txt

Path: `$PROJECT_DIR/CMakeLists.txt`

```
1 cmake_minimum_required(VERSION 2.8)
2
3 # project name
4 project(dsF)
5
6 # setup version numbers
7 set(VERSION_MAJOR 1)
8 set(VERSION_MINOR 0)
9 set(VERSION_PATCH 0)
10
11
12 # disable the rpath stuff
13 set(CMAKE_SKIP_BUILD_RPATH FALSE)
14
15
16 # detect the architecture (note: this test won't work for cross-compilation)
17
18 include(CheckTypeSize)
19 check_type_size(void* SIZEOF_VOID_PTR)
20 if("${SIZEOF_VOID_PTR}" STREQUAL "4")
21     set(ARCH x86)
22 elseif("${SIZEOF_VOID_PTR}" STREQUAL "8")
23     set(ARCH x64)
24 else()
25     message(FATAL_ERROR "Unsupported architecture")
26     return()
27 endif()
28
29 # configure links and headers
30
31 set(LIBS tbb yctools)
32
33 if (MSVC)
34     # Windows VC
35     # Activate C++ exception handling
36     if (NOT CMAKE_CXX_FLAGS MATCHES "/EHsc")
37         set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} /EHsc")
38     endif ()
39
40     # Set Warning level always to 4
41     if (CMAKE_CXX_FLAGS MATCHES "/W[0-4]")
42         string(REGEX REPLACE "/W[0-4]" "/W4" CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS}")
43     else ()
44         set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} /W4")
45     endif ()
46
47     set(LIBS_DIR ${CMAKE_CURRENT_SOURCE_DIR}/extlibs/libs-msvc/${ARCH}/libs)
48 elseif(APPLE)
49     # Mac OS X Xcode
50     set(CMAKE_MACOSX_RPATH 1)
51     set(LIBS_DIR ${CMAKE_CURRENT_SOURCE_DIR}/extlibs/libs-osx)
52     OPTION(OSX_FRAMEWORK "Build a Mac OS X Framework")
53     ADD_DEFINITIONS(-std=c++11)
54 else()
55     # Unix
56     set(LIBS_DIR ${CMAKE_CURRENT_SOURCE_DIR}/extlibs/libs-unix/${ARCH})
57     ADD_DEFINITIONS(-std=c++11)
58 endif()
```

```

59
60 # Source
61 set(INCROOT ${CMAKE_CURRENT_SOURCE_DIR}/dsf/include)
62 set(SRCROOT ${CMAKE_CURRENT_SOURCE_DIR}/dsf/src)
63 add_subdirectory(${CMAKE_CURRENT_SOURCE_DIR}/dsf)
64
65 # add the header path
66 include_directories(${CMAKE_CURRENT_SOURCE_DIR}/extlibs/headers)
67 include_directories(${INCROOT})
68
69 # output
70 add_library (${PROJECT_NAME} SHARED ${DSF_INC} ${DSF_SRC})
71 set_target_properties(
72     ${PROJECT_NAME} PROPERTIES
73     ARCHIVE_OUTPUT_DIRECTORY "${CMAKE_BINARY_DIR}/build"
74     LIBRARY_OUTPUT_DIRECTORY "${CMAKE_BINARY_DIR}/build"
75     RUNTIME_OUTPUT_DIRECTORY "${CMAKE_BINARY_DIR}/build"
76 )
77
78 if(OSX_FRAMEWORK)
79     set_target_properties(
80         ${PROJECT_NAME} PROPERTIES
81         FRAMEWORK TRUE
82         FRAMEWORK_VERSION ${VERSION_MAJOR}.${VERSION_MINOR}.${VERSION_PATCH}
83         MACOSX_FRAMEWORK_IDENTIFIER ie.itcarlow.yuchen.dsf
84         MACOSX_FRAMEWORK_SHORT_VERSION_STRING ${VERSION_MAJOR}.${VERSION_MINOR}.${VERSION_PATCH}
85         MACOSX_FRAMEWORK_BUNDLE_VERSION ${VERSION_MAJOR}.${VERSION_MINOR}.${VERSION_PATCH}
86         PUBLIC_HEADER "${DSF_INC}"
87     )
88 endif()
89
90 foreach(LIB ${LIBS})
91     find_library(LIB_${LIB} NAMES ${LIB} PATHS ${LIBS_DIR})
92     set_source_files_properties(${LIB_${LIB}} PROPERTIES MACOSX_PACKAGE_LOCATION Frameworks)
93     target_link_libraries(${PROJECT_NAME} ${LIB_${LIB}})
94 endforeach()

```

1.1.2 dsf/CMakeLists.txt

Path: `$PROJECT_DIR/dsf/CMakeLists.txt`

```

1 # source files
2 set(DSF_SRC
3     ${SRCROOT}/dsf/DualStateFramework.cpp
4     ${SRCROOT}/dsf/Lock.cpp
5     ${SRCROOT}/dsf/SynchronizedObject.cpp
6     ${SRCROOT}/dsf/SynchronizedVar.cpp
7     ${SRCROOT}/dsf/Task.cpp
8     ${SRCROOT}/dsf/TaskBox.cpp
9     PARENT_SCOPE
10 )
11 source_group("" FILES ${DSF_SRC})
12
13 # headers
14
15 set(DSF_INC
16     ${INCROOT}/dsf/Config.h
17     ${INCROOT}/dsf/Declaration.h
18     ${INCROOT}/dsf/DualStateFramework.h
19     ${INCROOT}/dsf/Export.h
20     ${INCROOT}/dsf/Lock.h
21     ${INCROOT}/dsf/Runnable.h
22     ${INCROOT}/dsf/SynchronizedObject.h
23     ${INCROOT}/dsf/SynchronizedVar.h
24     ${INCROOT}/dsf/Synchronisable.h
25     ${INCROOT}/dsf/Task.h
26     ${INCROOT}/dsf/TaskArgument.h
27     ${INCROOT}/dsf/TaskBox.h
28     ${INCROOT}/dsf/TaskFunction.h
29     PARENT_SCOPE
30 )
31 source_group("" FILES ${DSF_INC})
32

```

1.2 Config

1.2.1 Config.h

Path: \$PROJECT_DIR/dsf/include/dsf/Config.h

```

1 //
2 //  config.h
3 //  DualStateFramework
4 //
5 //  Created by Yu Chen on 10/13/14.
6 //  Copyright (c) 2014 Yu Chen. All rights reserved.
7 //
8
9 #ifndef dsf_Config_h
10 #define dsf_Config_h
11
12 #if defined(_WIN32)
13
14 // Windows compilers need specific (and different) keywords for export and import
15 #define DSF_API_EXPORT __declspec(dllexport)
16 #define DSF_API_IMPORT __declspec(dllimport)
17
18 // For Visual C++ compilers, we also need to turn off this annoying C4251 warning
19 #ifdef _MSC_VER
20
21 #pragma warning(disable : 4251)
22
23 #endif
24
25 #else // Linux, FreeBSD, Mac OS X
26
27 #if __GNUC__ >= 4
28
29 // GCC 4 has special keywords for showing/hidding symbols,
30 // the same keyword is used for both importing and exporting
31 #define DSF_API_EXPORT __attribute__((__visibility__("default")))
32 #define DSF_API_IMPORT __attribute__((__visibility__("default")))
33
34 #else
35
36 // GCC < 4 has no mechanism to explicitly hide symbols, everything's exported
37 #define DSF_API_EXPORT
38 #define DSF_API_IMPORT
39
40 #endif
41
42 #endif
43
44
45 #endif

```

1.3 Declaration

1.3.1 Declaration.h

Path: \$PROJECT_DIR/dsf/include/dsf/Declaration.h

```

1 //
2 //  Declaration.h
3 //  dsf
4 //
5 //  Created by Yu Chen on 10/17/14.
6 //
7 //
8
9 #ifndef dsf_Declaration_h
10 #define dsf_Declaration_h
11
12 #include "Export.h"
13
14 namespace dsf {
15     class DSF_API Task;
16     class DSF_API TaskBox;
17     class DSF_API DualStateFramework;
18     class DSF_API SynchronizedObject;
19 }
20
21 #endif

```

1.4 DualStateFramework

1.4.1 DualStateFramework.h

Path: \$PROJECT_DIR/dsf/include/dsf/DualStateFramework.h

```

1 //
2 // DualStateFramework.h
3 // dsf
4 //
5 // Created by Yu Chen on 10/17/14.
6 //
7 //
8
9 #ifndef dsf_DualStateFramework_h
10 #define dsf_DualStateFramework_h
11
12 #include <vector>
13
14 #include "TaskBox.h"
15 #include "Task.h"
16 #include "SynchronizedObject.h"
17 #include "Runnable.h"
18
19 namespace dsf {
20     class DSF_API DualStateFramework : public Runnable
21     {
22     public:
23         DualStateFramework();
24         ~DualStateFramework();
25
26         virtual void initialize() = 0;
27         void start();
28         void doOneFrame();
29
30         void add(SynchronizedObject* syncObj);
31         void remove(SynchronizedObject* syncObj);
32
33         void send(SynchronizedObject* to,
34                  SynchronizedObject* from,
35                  TaskFunction* taskFunction,
36                  TaskArgument* args);
37
38         State getState() override;
39         void setNumberOfThreads(int NumberOfThreads);
40
41     private:
42         int NumberOfThreads;
43         std::vector<SynchronizedObject*> syncObjs;
44         State state;
45     protected:
46         virtual void refresh();
47         virtual void run() override;
48     };
49 }
50 #endif

```

1.4.2 DualStateFramework.cpp

Path: \$PROJECT_DIR/dsf/src/dsf/DualStateFramework.cpp

```

1 //
2 // TaskManager.cpp
3 // dsf
4 //
5 // Created by Yu Chen on 10/17/14.
6 //
7 //
8
9 #include <tbb/parallel_for_each.h>
10 #include <dsf/DualStateFramework.h>

```

```

11 #include <tbb/task_scheduler_init.h>
12 #include <algorithm>
13
14 namespace dsf
15 {
16     DualStateFramework::DualStateFramework()
17     {
18         this->NumberOfThreads = tbb::task_scheduler_init::automatic;
19         this->syncObjs = new std::vector<SynchronizedObject*>();
20         this->state = DualStateFramework::State::STOPPED;
21     }
22
23     DualStateFramework::~DualStateFramework()
24     {
25         while (!this->syncObjs->empty()) {
26             SynchronizedObject* syncObj = this->syncObjs->back();
27             this->syncObjs->pop_back();
28             delete syncObj;
29         }
30         delete this->syncObjs;
31     }
32
33     void DualStateFramework::start()
34     {
35         if(!this->syncObjs->empty())
36         {
37             this->doOneFrame();
38             this->start();
39         }
40     }
41
42     void DualStateFramework::doOneFrame()
43     {
44         this->refresh();
45
46         if(!this->syncObjs->empty())
47         {
48             this->state = DualStateFramework::State::RUNNING;
49             this->run();
50             if(this->state == DualStateFramework::State::RUNNING)
51                 this->state = DualStateFramework::State::STOPPED;
52         }
53     }
54
55     void DualStateFramework::add(dsf::SynchronizedObject *syncObj)
56     {
57         this->syncObjs->push_back(syncObj);
58     }
59
60     void DualStateFramework::remove(dsf::SynchronizedObject *syncObj)
61     {
62         syncObj->distroy();
63     }
64
65     void DualStateFramework::send(dsf::SynchronizedObject *to, dsf::SynchronizedObject *from, TaskFunction
66 *taskFunction, TaskArgument *args)
67     {
68         from->send(to, taskFunction, args);
69     }
70
71     DualStateFramework::State DualStateFramework::getState()
72     {
73         return this->state;
74     }
75
76     void DualStateFramework::setNumberOfThreads(int NumberOfThreads)
77     {
78         if(NumberOfThreads == 0)
79             this->NumberOfThreads = tbb::task_scheduler_init::automatic;
80         else
81             this->NumberOfThreads = NumberOfThreads;
82     }
83
84     // Private
85
86     // Protected
87
88     void DualStateFramework::refresh()
89     {
90         this->syncObjs->erase(std::remove_if(this->syncObjs->begin(),
91             this->syncObjs->end(),
92             [](SynchronizedObject* sb)
93             {
94                 if (sb->getState() == SynchronizedObject::State::DELETED)

```

```

101     {
102         delete sb;
103         return true;
104     }
105     return false;
106     }) ,
107     this->syncObjs->end());
108 }
109 void DualStateFramework::run()
110 {
111     tbb::task_scheduler_init init(this->NumberOfThreads);
112     tbb::parallel_for_each(this->syncObjs->begin(),
113         this->syncObjs->end(),
114         [](SynchronizedObject* sb)
115         {
116             if(sb->getState() == SynchronizedObject::State::STOPPED)
117             {
118                 sb->start();
119             }
120         });
121 }
122 }

```

1.5 Export

1.5.1 Export.h

Path: `$PROJECT_DIR/dsf/include/dsf/Export.h`

```

1 //
2 // Export.h
3 // DualStateFramework
4 //
5 // Created by Yu Chen on 10/13/14.
6 // Copyright (c) 2014 Yu Chen. All rights reserved.
7 //
8
9 #ifndef dsf_Export_h
10 #define dsf_Export_h
11
12 // Headers
13 #include "Config.h"
14
15 // Define portable import / export macros
16 #if defined(dsf_EXPORTS)
17 #define DSF_API DSF_API_EXPORT
18 #else
19 #define DSF_API DSF_API_IMPORT
20 #endif
21 #endif

```

1.6 Lock

1.6.1 Lock.h

Path: `$PROJECT_DIR/dsf/include/dsf/Lock.h`

```

1 //
2 // Lockable.h
3 // dsf
4 //
5 // Created by Yu Chen on 12/10/14.
6 //
7 //
8
9 #ifndef dsf_Lock_h
10 #define dsf_Lock_h

```

```

11
12 #include <mutex>
13 #include "Export.h"
14
15 namespace dsf
16 {
17     class DSF_API Lock
18     {
19     protected:
20         std::mutex locker;
21     public:
22         void lock();
23         void unlock();
24     };
25 }
26 #endif

```

1.6.2 Lock.cpp

Path: `$PROJECT_DIR/dsf/src/dsf/Lock.cpp`

```

1 //
2 // Lock.cpp
3 // dsf
4 //
5 // Created by Yu Chen on 12/10/14.
6 //
7 //
8
9 #include <dsf/Lock.h>
10
11 namespace dsf
12 {
13     void Lock::lock()
14     {
15         this->locker.lock();
16     }
17
18     void Lock::unlock()
19     {
20         this->locker.unlock();
21     }
22 }

```

1.7 Runnable

1.7.1 Runnable.h

Path: `$PROJECT_DIR/dsf/include/dsf/Runnable.h`

```

1 //
2 // Runnable.h
3 // dsf
4 //
5 // Created by Yu Chen on 10/17/14.
6 //
7 //
8
9 #ifndef dsf_Runnable_h
10 #define dsf_Runnable_h
11
12 namespace dsf
13 {
14     class Runnable
15     {
16     public:
17         enum State
18         {
19             RUNNING, STOPPED, READY, DELETED
20         };
21         virtual State getState() = 0;
22     protected:
23         virtual void run() = 0;
24     };
25 }
26 #endif

```

1.8 Synchronisable

1.8.1 Synchronisable.h

Path: \$PROJECT_DIR/dsf/include/dsf/Synchronisable.h

```

1 //
2 // Synchronisable.h
3 // dsf
4 //
5 // Created by Yu Chen on 3/24/15.
6 //
7 //
8
9 #ifndef dsf_Synchronisable_h
10 #define dsf_Synchronisable_h
11
12 namespace dsf {
13     template<class T> class Synchronisable
14     {
15     protected:
16         T* next;
17     public:
18         virtual ~Synchronisable() {
19             delete this->next;
20         }
21         virtual void synchronise() = 0;
22     };
23 }
24 #endif

```

1.9 SynchronizedObject

1.9.1 SynchronizedObject.h

Path: \$PROJECT_DIR/dsf/include/dsf/SynchronizedObject.h

```

1 //
2 // SynchronizedObject.h
3 // dsf
4 //
5 // Created by Yu Chen on 10/17/14.
6 //
7 //
8
9 #ifndef dsf_SynchronizedObject_h
10 #define dsf_SynchronizedObject_h
11
12 #include <vector>
13 #include <string>
14
15 #include "Export.h"
16 #include "Declaration.h"
17 #include "Task.h"
18 #include "TaskBox.h"
19 #include "TaskFunction.h"
20 #include "TaskArgument.h"
21 #include "Synchronisable.h"
22 #include "Lock.h"
23 #include "Runnable.h"
24
25 namespace dsf
26 {
27     class DSF_API SynchronizedObject : public Runnable, public TaskBox, public Synchronisable<TaskBox>,
28     public Lock
29     {
30     public:
31         SynchronizedObject();
32         virtual ~SynchronizedObject();
33         State getState() override;
34         int receive();
35     private:
36         State state;
37
38         void push(Task* task);
39     };
40 }

```

```

75         void send(SynchronizedObject* to,
76                   TaskFunction* taskFunction,
77                   TaskArgument* args);
78         void start();
79         void stop();
80         void distroy();
81         friend class DualStateFramework;
82     protected:
83         virtual void run() override = 0;
84         void synchronise() override;
85     };
86 }
87
88 #endif

```

1.9.2 SynchronizedObject.cpp

Path: \$PROJECT_DIR/dsf/src/dsf/SynchronizedObject.cpp

```

1  //
2  //  SynchronizedObject.cpp
3  //  dsf
4  //
5  //  Created by Yu Chen on 10/17/14.
6  //
7  //
8
9  #include <dsf/SynchronizedObject.h>
10
11 namespace dsf
12 {
13     SynchronizedObject::SynchronizedObject()
14         : TaskBox()
15     {
16         this->next = new TaskBox();
17         this->state = State::STOPPED;
18     }
19     SynchronizedObject::~SynchronizedObject()
20     {
21     }
22
23     SynchronizedObject::State SynchronizedObject::getState()
24     {
25         return this->state;
26     }
27
28     int SynchronizedObject::receive()
29     {
30     {
31         int count = 0;
32         while (!this->next->isEmpty())
33         {
34             this->tasks->push_back(this->next->pop());
35             count ++;
36         }
37         return count;
38     }
39
40     void SynchronizedObject::push(dsf::Task *task)
41     {
42         this->lock();
43         this->next->push(task);
44         this->unlock();
45     }
46
47     // Privates
48
49     void SynchronizedObject::send(dsf::SynchronizedObject *to,
50                                   TaskFunction *taskFunction,
51                                   TaskArgument *args)
52     {
53         to->push(new Task(to, this, taskFunction, args));
54     }
55
56     void SynchronizedObject::start()
57     {
58     {
59         this->state = State::RUNNING;
60         this->run();
61         if(this->state == State::RUNNING)
62             this->stop();
63     }
64
65 }
66

```

```

67     void SynchronizedObject::stop()
68     {
69         this->state = State::STOPPED;
70     }
71
72     void SynchronizedObject::distroy()
73     {
74         this->state = State::DELETED;
75     }
76
77
78
79     // Protected
80
81
82     void SynchronizedObject::synchronise()
83     {
84         *((TaskBox*) this) = *this->next;
85     }
86
87 }

```

1.10 SynchronizedVar

1.10.1 SynchronizedVar.h

Path: \$PROJECT_DIR/dsf/include/dsf/SynchronizedVar.h

```

1 //
2 // SynchronizedVar.h
3 // dsf
4 //
5 // Created by Yu Chen on 12/2/14.
6 //
7 //
8
9 #ifndef dsf_SynchronizedVar_h
10 #define dsf_SynchronizedVar_h
11
12 #include "Export.h"
13 #include <yctools/Any.h>
14 #include "Synchronisable.h"
15 #include "Lock.h"
16
17 namespace dsf
18 {
19     class DSF_API SynchronizedVar : public Synchronisable<yc::Any>, public yc::Any, public Lock
20     {
21     public:
22         template<typename T> SynchronizedVar(T && value);
23         template<typename T> void operator=(T && value);
24         void synchronise() override;
25     };
26 }
27
28 namespace dsf
29 {
30
31     template<typename T> SynchronizedVar::SynchronizedVar(T && value)
32     : Any(value)
33     {
34         this->next = new Any(value);
35     }
36
37     template<typename T> void SynchronizedVar::operator=(T && value)
38     {
39         this->lock();
40         this->next = value;
41         this->unlock();
42     }
43 }
44
45 #endif

```

1.10.2 SynchronizedVar.cpp

Path: \$PROJECT_DIR/dsf/src/dsf/SynchronizedVar.cpp


```

1 //
2 // SynchronizedVar.cpp
3 // dsf
4 //
5 // Created by Yu Chen on 12/2/14.
6 //
7 //
8
9 #include <dsf/SynchronizedVar.h>
10
11 namespace dsf
12 {
13     void SynchronizedVar::synchronise()
14     {
15         *((yc::Any*)this) = *this->next;
16     }
17 }
18

```

1.11 Task

1.11.1 Task.h

Path: \$PROJECT_DIR/dsf/include/dsf/Task.h

```

1 //
2 // Parallel.h
3 // DualStateFramework
4 //
5 // Created by Yu Chen on 10/13/14.
6 // Copyright (c) 2014 Yu Chen. All rights reserved.
7 //
8
9 #ifndef dsf_Task_h
10 #define dsf_Task_h
11
12 #include <vector>
13 #include <memory>
14
15 #include "Export.h"
16 #include "Declaration.h"
17 #include "TaskBox.h"
18 #include "TaskFunction.h"
19 #include "TaskArgument.h"
20
21
22 namespace dsf
23 {
24     class DSF_API Task
25     {
26     public:
27         SynchronizedObject* to;
28         SynchronizedObject* from;
29         TaskFunction* taskFunction;
30         TaskArgument* taskArgument;
31         explicit Task(SynchronizedObject* to,
32                     SynchronizedObject* from,
33                     TaskFunction* taskFunction,
34                     TaskArgument* taskArgument);
35         ~Task();
36     };
37 }
38
39 #endif

```

1.11.2 Task.cpp

Path: \$PROJECT_DIR/dsf/src/dsf/Task.cpp

```

1 //
2 // Parallel.cpp
3 // DualStateFramework
4 //
5 // Created by Yu Chen on 10/13/14.
6 // Copyright (c) 2014 Yu Chen. All rights reserved.
7 //
8

```

```

9 #include <dsf/Task.h>
10 namespace dsf
11 {
12     Task::Task(SynchronizedObject* to,
13               SynchronizedObject* from,
14               TaskFunction* taskFunction,
15               TaskArgument* taskArgument)
16     {
17         this->to = to;
18         this->from = from;
19         this->taskFunction = taskFunction;
20         this->taskArgument = taskArgument;
21     }
22
23     Task::~Task()
24     {
25         delete this->taskArgument;
26     }
27 }

```

1.12 TaskArgument

1.12.1 TaskArgument.h

Path: `$PROJECT_DIR/dsf/include/dsf/TaskArgument.h`

```

1 //
2 // TaskArguments.h
3 // dsf
4 //
5 // Created by Yu Chen on 10/17/14.
6 //
7 //
8
9 #ifndef dsf_TaskArgument_h
10 #define dsf_TaskArgument_h
11
12 #include <vector>
13
14 #include <yctools/Any.h>
15 #include <yctools/AnyException.h>
16
17 namespace dsf
18 {
19     typedef yc::Any TaskArgument;
20     typedef yc::Exception::AnyException TaskArgumentException;
21 }
22
23
24
25
26 #endif

```

1.13 TaskBox

1.13.1 TaskBox.h

Path: `$PROJECT_DIR/dsf/include/dsf/TaskBox.h`

```

1 //
2 // Parallelable.h
3 // DSF
4 //
5 // Created by Yu Chen on 10/14/14.
6 //
7 //
8
9 #ifndef dsf_TaskBox_h
10 #define dsf_TaskBox_h
11
12 #include <vector>
13 #include <string>
14
15 #include "Export.h"

```

```

16 #include "Declaration.h"
17 #include "Task.h"
18 #include "Runnable.h"
19
20 namespace dsf
21 {
22     class DSF_API TaskBox
23     {
24     public:
25         TaskBox();
26         virtual ~TaskBox();
27         bool isEmpty();
28         void push(Task* task);
29         Task* pop();
30         void process();
31     protected:
32         std::vector<Task*>* tasks;
33     };
34 }
35 #endif

```

1.13.2 TaskBox.cpp

Path: \$PROJECT_DIR/dsf/src/dsf/TaskBox.cpp

```

1 //
2 // TaskBox.cpp
3 // dsf
4 //
5 // Created by Yu Chen on 10/17/14.
6 //
7 //
8
9 #include <dsf/TaskBox.h>
10
11 namespace dsf
12 {
13     TaskBox::TaskBox()
14     {
15         this->tasks = new std::vector<Task*>();
16     }
17
18     TaskBox::~~TaskBox()
19     {
20         for (std::vector<Task*>::iterator i = this->tasks->begin(); i != this->tasks->end(); ++i)
21         {
22             delete *i;
23         }
24         delete this->tasks;
25     }
26
27     bool TaskBox::isEmpty()
28     {
29         return this->tasks->empty();
30     }
31
32     void TaskBox::push(dsf::Task *task)
33     {
34         this->tasks->push_back(task);
35     }
36
37     Task* TaskBox::pop()
38     {
39         Task* task = this->tasks->back();
40         this->tasks->pop_back();
41         return task;
42     }
43
44     void TaskBox::process()
45     {
46         while (!this->isEmpty())
47         {
48             Task* task = this->pop();
49             (*task->taskFunction)(task->to, task->from, task->taskArgument);
50             delete task;
51         }
52     }
53 }

```

1.14 TaskFunction

1.14.1 TaskFunction.h

Path: `$PROJECT_DIR/dsf/include/dsf/TaskFunction.h`

```
1 //
2 // TaskFunction.h
3 // dsf
4 //
5 // Created by Yu Chen on 10/17/14.
6 //
7 //
8
9 #ifndef dsf_TaskFunction_h
10 #define dsf_TaskFunction_h
11
12 #include <functional>
13 #include <vector>
14
15 #include "TaskArgument.h"
16
17 namespace dsf
18 {
19     typedef std::function<void (dsf::SynchronizedObject*, dsf::SynchronizedObject*, TaskArgument*)>
        TaskFunction;
20     typedef void function;
21 }
22
23 #endif
```

2 | yctools

2.1 Any

2.1.1 Any.h

Path: \$PROJECT_DIR/dsf/include/yctools/Any.h

```
1 //
2 // Any.h
3 // YCTools
4 //
5 // Created by Yu Chen on 11/13/14.
6 // Copyright (c) 2014 Yu Chen. All rights reserved.
7 //
8
9 #ifndef YCTools_Any_h
10 #define YCTools_Any_h
11
12 #include <memory>
13 #include <typeindex>
14 #include <string>
15
16 #include "Export.h"
17 #include "AnyException.h"
18
19 namespace yc
20 {
21     class YCTOOLS_API Any
22     {
23     public:
24         Any(void);
25         Any(Any& that);
26         Any(Any && that);
27         template<typename T> Any(T && value);
28
29         bool isNull() const;
30         template<class T> bool is() const;
31         template<class T> typename std::decay<T>::type& to();
32
33         template<class T> operator T() const;
34         Any& operator=(const Any& any);
35
36     private:
37         class Base;
38         template<typename T> class Derived;
39         std::unique_ptr<Base> Clone() const;
40         std::unique_ptr<Base> pointer;
41         std::type_index typeIndex;
42     };
43 }
44
45
46 // Class yc::Any
47 namespace yc
48 {
49     {
50
51         template<typename T> Any::Any(T && value) :
52             pointer(new Derived <typename std::decay<T>::type> (std::forward<T>(value))),
53             typeIndex(std::type_index(typeid(typename std::decay<T>::type)))
54         {}
55
56         template<class T> bool Any::is() const
57         {
58             return typeIndex == std::type_index(typeid(T));
59         }
60     }
61 }
```

```

60     }
61
62     template<class T> typename std::decay<T>::type& Any::to()
63     {
64         if (!this->is<T>())
65         {
66             throw Exception::AnyException(typeid(T).name(), typeIndex.name());
67         }
68
69         typedef typename std::decay<T>::type U;
70         auto derived = static_cast<Derived<U>*> (pointer.get());
71         return derived->value;
72     }
73     template<class T> Any::operator T() const
74     {
75         return this->to<typename std::decay<T>::type>();
76     }
77
78 }
79
80 // Inner Classes
81
82 namespace yc {
83     class YCTOOLS_API Any::Base
84     {
85     public:
86         virtual ~Base();
87         virtual std::unique_ptr<Base> Clone() const = 0;
88     };
89     template<typename T>
90     class Any::Derived : public Any::Base
91     {
92     public:
93         template<typename U>
94         Derived(U && value) : value(std::forward<U>(value)) { }
95
96         std::unique_ptr<Base> Clone() const
97         {
98             return std::unique_ptr<Base>(new Derived<T>(this->value));
99         }
100
101         T value;
102     };
103 }
104
105 #endif

```

2.1.2 Any.cpp

Path: \$PROJECT_DIR/dsf/src/yctools/Any.cpp

```

1 //
2 // Any.cpp
3 // YCTools
4 //
5 // Created by Yu Chen on 11/13/14.
6 // Copyright (c) 2014 Yu Chen. All rights reserved.
7 //
8
9 #include "../include/yctools/Any.h"
10
11 namespace yc
12 {
13     Any::Any() :
14         typeIndex(std::type_index(typeid(void) ))
15     {}
16
17     Any::Any(Any& that) :
18         pointer(that.Clone()),
19         typeIndex(that.typeIndex)
20     {}
21
22     Any::Any(Any && that) :
23         pointer(std::move(that.pointer)),
24         typeIndex(that.typeIndex)
25     {}
26
27     bool Any::isNull() const
28     {
29         return !bool(pointer);
30     }

```

```

30     }
31
32     Any& Any::operator=(const Any& any)
33     {
34         if (this->pointer == any.pointer)
35             return *this;
36
37         this->pointer = any.Clone();
38         this->typeIndex = any.typeIndex;
39         return *this;
40     }
41
42     Any::Base::~Base() {}
43
44     std::unique_ptr<Any::Base> Any::Clone() const
45     {
46         if (pointer != nullptr)
47             return pointer->Clone();
48
49         return nullptr;
50     }
51 }

```

2.2 AnyException

2.2.1 AnyException.h

Path: \$PROJECT_DIR/dsf/include/yctools/AnyException.h

```

1 //
2 // AnyException.h
3 // YCTools
4 //
5 // Created by Yu Chen on 11/13/14.
6 // Copyright (c) 2014 Yu Chen. All rights reserved.
7 //
8
9 #ifndef YCTools_AnyException_h
10 #define YCTools_AnyException_h
11
12 #include <exception>
13 #include <string>
14
15 #include "Export.h"
16 namespace yc
17 {
18     namespace Exception
19     {
20         class YCTOOLS_API AnyException : public std::exception
21         {
22         public:
23             AnyException(std::string from,
24                         std::string to);
25             virtual const char* what() const throw() override;
26             std::string details() const throw();
27         private:
28             std::string from;
29             std::string to;
30         };
31     }
32 }
33
34 #endif

```

2.2.2 AnyException.cpp

Path: \$PROJECT_DIR/dsf/src/yctools/AnyException.cpp

```

1 //
2 // AnyException.cpp
3 // YCTools
4 //
5 // Created by Yu Chen on 11/13/14.
6 // Copyright (c) 2014 Yu Chen. All rights reserved.
7 //
8
9 #include "../include/yctools/AnyException.h"

```

```

10
11 namespace yc
12 {
13     namespace Exception
14     {
15
16         AnyException::AnyException(std::string from,
17                                     std::string to) :
18             from(from),
19             to(to)
20         {}
21
22         const char* AnyException::what() const throw()
23         {
24             return "\"Any\" exception happened!";
25         }
26         std::string AnyException::details() const throw()
27         {
28             return "Can not cast " + from + " to " + to;
29         }
30     }
31 }

```

2.3 Config

2.3.1 Config.h

Path: `$PROJECT_DIR/dsf/include/yctools/Config.h`

```

1 //
2 //  config.h
3 //  YCTOOLS
4 //
5 //  Created by Yu Chen on 10/13/14.
6 //  Copyright (c) 2014 Yu Chen. All rights reserved.
7 //
8
9 #ifndef yctools_Config_h
10 #define yctools_Config_h
11
12 #if defined(_WIN32)
13
14 // Windows compilers need specific (and different) keywords for export and import
15 #define YCTOOLS_API_EXPORT __declspec(dllexport)
16 #define YCTOOLS_API_IMPORT __declspec(dllimport)
17
18 // For Visual C++ compilers, we also need to turn off this annoying C4251 warning
19 #ifndef _MSC_VER
20
21 #pragma warning(disable : 4251)
22
23 #endif
24
25 #else // Linux, FreeBSD, Mac OS X
26
27 #if __GNUC__ >= 4
28
29 // GCC 4 has special keywords for showing/hidding symbols,
30 // the same keyword is used for both importing and exporting
31 #define YCTOOLS_API_EXPORT __attribute__((__visibility__("default")))
32 #define YCTOOLS_API_IMPORT __attribute__((__visibility__("default")))
33
34 #else
35
36 // GCC < 4 has no mechanism to explicitly hide symbols, everything's exported
37 #define YCTOOLS_API_EXPORT
38 #define YCTOOLS_API_IMPORT
39
40 #endif
41
42 #endif
43
44
45 #endif

```

2.4 Export

2.4.1 Export.h

Path: \$PROJECT_DIR/dsf/include/yctools/Export.h

```
1 //
2 // Export.h
3 // YCTOOLS
4 //
5 // Created by Yu Chen on 10/13/14.
6 // Copyright (c) 2014 Yu Chen. All rights reserved.
7 //
8
9 #ifndef yctools_Export_h
10 #define yctools_Export_h
11
12 // Headers
13 #include "Config.h"
14
15 // Define portable import / export macros
16 #if defined(YCTOOLS_EXPORTS)
17 #define YCTOOLS_API YCTOOLS_API_EXPORT
18 #else
19 #define YCTOOLS_API YCTOOLS_API_IMPORT
20 #endif
21 #endif
```

2.5 Random

2.5.1 Random.h

Path: \$PROJECT_DIR/dsf/include/yctools/Random.h

```
1 //
2 // Random.h
3 // yctools
4 //
5 // Created by Yu Chen on 2/9/15.
6 //
7 //
8
9 #ifndef yctools_Random_h
10 #define yctools_Random_h
11
12 #include <stdlib.h>
13 #include <ctime>
14 #include <random>
15
16 #include "Export.h"
17 namespace yc
18 {
19     class YCTOOLS_API Random
20     {
21     public:
22         int randInt(int min, int max);
23         float randFloat(float min, float max);
24     };
25 }
26
27 #endif
```

2.5.2 Random.cpp

Path: \$PROJECT_DIR/dsf/src/yctools/Random.cpp

```
1 //
2 // Random.cpp
3 // yctools
4 //
5 // Created by Yu Chen on 2/9/15.
```

```
6 //
7 //
8
9
10 #include <yctools/Random.h>
11
12 int yc::Random::randInt(int min, int max)
13 {
14     std::random_device rd;
15     std::default_random_engine generator(rd());
16     std::uniform_int_distribution<int> distribution(min, max);
17     int dice_roll = distribution(generator);
18     return dice_roll;
19 }
20
21 float yc::Random::randFloat(float min, float max)
22 {
23     std::random_device rd;
24     std::default_random_engine generator(rd());
25     std::uniform_real_distribution<float> distribution(min, max);
26     float dice_roll = distribution(generator);
27     return dice_roll;
28 }
```

3 | Benchmark Program

3.1 CMakeLists

3.1.1 CMakeLists.txt

Path: `$PROJECT_DIR/profiler/CMakeLists.txt`

```
1 cmake_minimum_required(VERSION 2.8)
2
3 # project name
4 project(profiler)
5
6 # setup version numbers
7 set(VERSION_MAJOR 1)
8 set(VERSION_MINOR 0)
9 set(VERSION_PATCH 0)
10
11
12 # disable the rpath stuff
13 set(CMAKE_SKIP_BUILD_RPATH FALSE)
14
15
16 # detect the architecture (note: this test won't work for cross-compilation)
17 include(CheckTypeSize)
18 check_type_size(void* SIZEOF_VOID_PTR)
19 if("${SIZEOF_VOID_PTR}" STREQUAL "4")
20     set(ARCH x86)
21 elseif("${SIZEOF_VOID_PTR}" STREQUAL "8")
22     set(ARCH x64)
23 else()
24     message(FATAL_ERROR "Unsupported architecture")
25     return()
26 endif()
27
28 # links
29 set(SFML_LIBS sfml-audio sfml-graphics sfml-network sfml-system sfml-window)
30 set(DSF_LIBS tbb yctools dsf)
31 set(LIBS ${SFML_LIBS} ${DSF_LIBS})
32
33 # headers
34 include_directories(${CMAKE_CURRENT_SOURCE_DIR}/extlibs/headers)
35 set(INCROOT ${CMAKE_CURRENT_SOURCE_DIR}/include)
36 set(INC
37     ${INCROOT}/BouncingCircleManager.h
38     ${INCROOT}/MyDSF.h
39     ${INCROOT}/SyncFlockingBoid.h
40     ${INCROOT}/DSFSFML.h
41     ${INCROOT}/RandomCircleManager.h
42     ${INCROOT}/SyncVector3D.h
43     ${INCROOT}/ResourcePath.hpp
44     ${INCROOT}/Vector3D.h
45     ${INCROOT}/FPS.h
46     ${INCROOT}/SyncBouncingCircle.h
47     ${INCROOT}/FlockingBoidManager.h
48     ${INCROOT}/SyncCircle.h
49 )
50
51 # source
52 set(SRCROOT ${CMAKE_CURRENT_SOURCE_DIR}/src)
53 set(SRC
54     ${SRCROOT}/BouncingCircleManager.cpp
55     ${SRCROOT}/DSFSFML.cpp
56     ${SRCROOT}/SyncBouncingCircle.cpp
57     ${SRCROOT}/SyncCircle.cpp
58     ${SRCROOT}/FPS.cpp
```

```

59     ${SRCROOT}/SyncFlockingBoid.cpp
60     ${SRCROOT}/FlockingBoidManager.cpp
61     ${SRCROOT}/SyncVector3D.cpp
62     ${SRCROOT}/MyDSF.cpp
63     ${SRCROOT}/Vector3D.cpp
64     ${SRCROOT}/RandomCircleManager.cpp
65     ${SRCROOT}/main.cpp
66 )
67
68 # resources
69 set(RESRCROOT ${PROJECT_SOURCE_DIR}/resource)
70
71 # os configurations
72 if (MSVC)
73     # Windows VC
74     # Activate C++ exception handling
75     if (NOT CMAKE_CXX_FLAGS MATCHES "/EHsc")
76         set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} /EHsc")
77     endif ()
78
79     # Set Warning level always to 4
80     if (CMAKE_CXX_FLAGS MATCHES "/W[0-4]")
81         string(REGEX REPLACE "/W[0-4]" "/W4" CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS}")
82     else ()
83         set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} /W4")
84     endif ()
85
86     # Add Math definitions
87     add_definitions(-D_USE_MATH_DEFINES)
88
89     # Add Source
90     set(SRC ${SRC} ${SRCROOT}/ResourcePath.cpp)
91     add_executable(${PROJECT_NAME} ${INC} ${SRC})
92     # Add Links
93     set(LIBS_DIR ${CMAKE_CURRENT_SOURCE_DIR}/extlibs/libs-msvc/${ARCH}/libs)
94     set(DEPENDENCIES_DIR ${CMAKE_CURRENT_SOURCE_DIR}/extlibs/libs-msvc/${ARCH}/dlls)
95     set(DEPENDENCIES_TARGET_DIR ${TARGET_FILE_DIR:${PROJECT_NAME}})
96     set(RESOURCES_TARGET_DIR ${TARGET_FILE_DIR:${PROJECT_NAME}}/Resources)
97
98 elseif (APPLE)
99     # Mac OS X Xcode
100     set(CMAKE_MACOSX_RPATH 1)
101     ADD_DEFINITIONS(-std=c++11)
102
103     # Add Source
104     set(SRC ${SRC} ${SRCROOT}/ResourcePath.mm)
105     add_executable(${PROJECT_NAME} MACOSX_BUNDLE ${INC} ${SRC})
106     # Add Links
107     SET_TARGET_PROPERTIES(${PROJECT_NAME} PROPERTIES
108         XCODE_ATTRIBUTE_LD_RUNPATH_SEARCH_PATHS @executable_path/../../Frameworks/)
109     set(LIBS_DIR ${CMAKE_CURRENT_SOURCE_DIR}/extlibs/libs-osx)
110     set(DEPENDENCIES_DIR ${LIBS_DIR})
111     set(LIBS ${LIBS} SFML)
112     set(DEPENDENCIES_TARGET_DIR ${TARGET_FILE_DIR:${PROJECT_NAME}}/../../Frameworks)
113     set(RESOURCES_TARGET_DIR ${TARGET_FILE_DIR:${PROJECT_NAME}}/../../Resources)
114 else()
115     # Unix
116     ADD_DEFINITIONS(-std=c++11)
117     # Add Source
118     set(SRC ${SRC} ${SRCROOT}/ResourcePath.cpp)
119     add_executable(${PROJECT_NAME} ${INC} ${SRC})
120     # Add Links
121     set(LIBS_DIR ${CMAKE_CURRENT_SOURCE_DIR}/extlibs/libs-unix/${ARCH})
122     set(DEPENDENCIES_DIR ${LIBS_DIR})
123     set(DEPENDENCIES_TARGET_DIR ${TARGET_FILE_DIR:${PROJECT_NAME}})
124     set(RESOURCES_TARGET_DIR ${TARGET_FILE_DIR:${PROJECT_NAME}}/Resources)
125
126 endif()
127
128
129 foreach(LIB ${LIBS})
130     find_library(LIB_${LIB} NAMES ${LIB} PATHS ${LIBS_DIR})
131     target_link_libraries(${PROJECT_NAME} ${LIB_${LIB}})
132 endforeach()
133
134 set_target_properties(
135     ${PROJECT_NAME} PROPERTIES
136     ARCHIVE_OUTPUT_DIRECTORY "${CMAKE_BINARY_DIR}/build"
137     LIBRARY_OUTPUT_DIRECTORY "${CMAKE_BINARY_DIR}/build"
138     RUNTIME_OUTPUT_DIRECTORY "${CMAKE_BINARY_DIR}/build"
139 )
140
141 # copy dependencies and resources
142 add_custom_command(TARGET ${PROJECT_NAME} POST_BUILD
143     COMMAND ${CMAKE_COMMAND} -E copy_directory
144         ${DEPENDENCIES_DIR}
145         ${DEPENDENCIES_TARGET_DIR})

```

```

146 add_custom_command(TARGET ${PROJECT_NAME} POST_BUILD
147                     COMMAND ${CMAKE_COMMAND} -E copy_directory
148                         "${PROJECT_SOURCE_DIR}/resource"
149                         ${RESOURCES_TARGET_DIR})

```

3.2 BouncingCircleManager

3.2.1 BouncingCircleManager.h

Path: `$PROJECT_DIR/profiler/include/BouncingCircleManager.h`

```

1 //
2 // BouncingCircleManager.h
3 // profiler
4 //
5 // Created by Yu Chen on 2/22/15.
6 //
7 //
8
9 #ifndef profiler_BouncingCircleManager_h
10 #define profiler_BouncingCircleManager_h
11
12 #include "MyDSF.h"
13 #include "SyncBouncingCircle.h"
14 #include <yctools/Random.h>
15 #include <vector>
16
17 class BouncingCircleManager
18 {
19 public:
20     BouncingCircleManager(MyDSF* dsf);
21     ~BouncingCircleManager();
22     MyDSF* dsf;
23     dsf::TaskFunction* create;
24     dsf::TaskFunction* update;
25     dsf::TaskFunction* destroy;
26     std::vector<SyncBouncingCircle*>* createRandomCircles(int number, int radius, int boundX, int boundY);
27 private:
28     SyncBouncingCircle* createRandomCircle(int radius, int boundX, int boundY);
29 };
30
31
32 #endif

```

3.2.2 BouncingCircleManager.cpp

Path: `$PROJECT_DIR/profiler/src/BouncingCircleManager.cpp`

```

1 //
2 // BouncingCircleManager.cpp
3 // profiler
4 //
5 // Created by Yu Chen on 2/22/15.
6 //
7 //
8
9 #include "../include/BouncingCircleManager.h"
10 #include <cmath>
11
12 BouncingCircleManager::BouncingCircleManager(MyDSF* dsf)
13 {
14     this->dsf = dsf;
15     this->create = new dsf::TaskFunction([this](dsf::SynchronizedObject* to, dsf::SynchronizedObject* from,
16     dsf::TaskArgument* args)
17     {
18         SyncBouncingCircle* syncObj;
19         std::vector<SyncBouncingCircle*>* syncObjs;
20         std::tie(syncObj, syncObjs) = args->to<
21         std::tuple<SyncBouncingCircle*, std::vector<SyncBouncingCircle*>*>>();
22         if(this->dsf->window->isOpen())
23         {
24             this->dsf->send(to, from, this->update, new
25             dsf::TaskArgument(std::make_tuple(syncObj, syncObjs)));
26             this->dsf->lock();
27             this->dsf->drawables->push_back(syncObj);
28             this->dsf->unlock();
29         }
30     });
31 }

```

```

27                                     else
28                                     {
29                                     this->dsf->send(to, from, this->destroy, new
dsf::TaskArgument(syncObj));
30                                     }
31                                     });
32     this->update = new dsf::TaskFunction([this](dsf::SynchronizedObject* to, dsf::SynchronizedObject* from,
dsf::TaskArgument* args)
33                                     {
34         SyncBouncingCircle* syncObj;
35         std::vector<SyncBouncingCircle*>* syncObjs;
36         std::tie(syncObj, syncObjs) = args->to<
std::tuple<SyncBouncingCircle*, std::vector<SyncBouncingCircle*>*>>();
37         if(this->dsf->window->isOpen())
38         {
39             auto size = this->dsf->window->getSize();
40             for(auto & sb : *syncObjs)
41             {
42                 if(sb != syncObj)
43                     syncObj->collide(sb);
44             }
45             syncObj->move(size.x, size.y);
46             this->dsf->send(to, from, this->update, new
dsf::TaskArgument(std::make_tuple(syncObj, syncObjs)));
47         }
48         else
49         {
50             this->dsf->send(to, from, this->destroy, new
dsf::TaskArgument(syncObj));
51         }
52         });
53     this->destroy = new dsf::TaskFunction([this](dsf::SynchronizedObject* to, dsf::SynchronizedObject* from
, dsf::TaskArgument* args)
54                                     {
55         auto syncObj = args->to<SyncBouncingCircle*>();
56         auto drawable = (sf::Drawable*) syncObj;
57         this->dsf->lock();
58         this->dsf->drawables->erase(
59                                     std::remove_if(
60         drawables->begin(),
61                                     this->dsf->
62         drawables->end(),
63                                     [&](sf::Drawable*
64         d)
65                                     {
66                                     return d ==
67                                     this->dsf->drawables->end());
68         this->dsf->remove(to);
69         this->dsf->unlock();
70     });
71 BouncingCircleManager::~BouncingCircleManager()
72 {
73     delete this->create;
74     delete this->update;
75     delete this->destroy;
76 }
77
78 std::vector<SyncBouncingCircle*>* BouncingCircleManager::createRandomCircles(int number, int radius, int
boundX, int boundY) {
79     int numberOfCircles = number;
80     auto circles = new std::vector<SyncBouncingCircle*>();
81     SyncBouncingCircle* circle;
82     while(circles->size() < numberOfCircles) {
83         std::vector<SyncBouncingCircle*>::iterator itr = circles->begin();
84         circle = createRandomCircle(radius, boundX, boundY);
85         while (itr != circles->end()) {
86             if (circle->hasCollision(*itr)) {
87                 delete circle;
88                 circle = createRandomCircle(radius, boundX, boundY);
89                 itr = circles->begin();
90             } else {
91                 itr++;
92             }
93         }
94         circles->push_back(circle);
95     }
96     return circles;
97 }
98
99 SyncBouncingCircle* BouncingCircleManager::createRandomCircle(int radius, int boundX, int boundY) {
100     SyncBouncingCircle* circle = new SyncBouncingCircle();
101     int direction = yc::Random().randInt(1, 360);
102     int posX = yc::Random().randInt(0, boundX - radius * 2);

```

```

103     int posy = yc::Random().randInt(0, boundY - radius * 2);
104     float x = cosf(direction * M_PI / 180);
105     float y = sinf(direction * M_PI / 180);
106     circle->setRadius(radius);
107     circle->setPosition(sf::Vector2<float>(posx, posy));
108     circle->setVelocity(sf::Vector2<float>(x, y));
109     return circle;
110 }

```

3.3 DSFSFML

3.3.1 DSFSFML.h

Path: `$PROJECT_DIR/profiler/include/DSFSFML.h`

```

1 //
2 // DSFSFML.h
3 // profiler
4 //
5 // Created by Yu Chen on 2/16/15.
6 //
7 //
8
9 #ifndef profiler_DSFSFML_h
10 #define profiler_DSFSFML_h
11
12 #include <SFML/Graphics.hpp>
13 #include <vector>
14
15 namespace dsf
16 {
17     namespace sfml
18     {
19         class RenderWindow
20         {
21         public:
22             explicit RenderWindow();
23             virtual ~RenderWindow();
24             sf::RenderWindow* window;
25             std::vector<sf::Drawable*>* drawables;
26         protected:
27             virtual void draw() = 0;
28         };
29     }
30 }
31 #endif

```

3.3.2 DSFSFML.cpp

Path: `$PROJECT_DIR/profiler/src/DSFSFML.cpp`

```

1 //
2 // DSFSFML.cpp
3 // profiler
4 //
5 // Created by Yu Chen on 2/17/15.
6 //
7 //
8
9 #include "../include/DSFSFML.h"
10
11 namespace dsf
12 {
13     namespace sfml
14     {
15         RenderWindow::RenderWindow()
16         {
17             this->window = new sf::RenderWindow();
18             this->drawables = new std::vector<sf::Drawable*>();
19         }
20
21         RenderWindow::~~RenderWindow()
22         {
23             delete this->window;
24             delete this->drawables;
25         }
26     }

```

```
27 }
28
```

3.4 FlockingBoidManager

3.4.1 FlockingBoidManager.h

Path: \$PROJECT_DIR/profiler/include/FlockingBoidManager.h

```
1 //
2 // FlockingBoidManager.h
3 // profiler
4 //
5 // Created by Yu Chen on 3/12/15.
6 //
7 //
8
9 #ifndef profiler_FlockingBoidManager_h
10 #define profiler_FlockingBoidManager_h
11
12 #include "MyDSF.h"
13 #include "SyncFlockingBoid.h"
14 #include <yctools/Random.h>
15 #include <vector>
16
17 class FlockingBoidManager
18 {
19 public:
20     FlockingBoidManager(MyDSF* dsf);
21     ~FlockingBoidManager();
22     MyDSF* dsf;
23     dsf::TaskFunction* create;
24     dsf::TaskFunction* update;
25     dsf::TaskFunction* destroy;
26 };
27
28 #endif
```

3.4.2 FlockingBoidManager.cpp

Path: \$PROJECT_DIR/profiler/src/FlockingBoidManager.cpp

```
1 //
2 // FlockingBoidManager.cpp
3 // profiler
4 //
5 // Created by Yu Chen on 3/12/15.
6 //
7 //
8
9 #include "../include/FlockingBoidManager.h"
10
11 FlockingBoidManager::FlockingBoidManager(MyDSF* dsf)
12 {
13     this->dsf = dsf;
14     this->create = new dsf::TaskFunction([this](dsf::SynchronizedObject* to, dsf::SynchronizedObject* from,
15         dsf::TaskArgument* args)
16     {
17         SyncFlockingBoid* syncObj;
18         std::vector<SyncFlockingBoid*> syncObjs;
19         std::tie(syncObj, syncObjs) = args->to<
20             std::tuple<SyncFlockingBoid*, std::vector<SyncFlockingBoid*>*>>();
21         if(this->dsf->window->isOpen())
22         {
23             this->dsf->send(to, from, this->update, new
24             dsf::TaskArgument(std::make_tuple(syncObj, syncObjs)));
25             this->dsf->lock();
26             this->dsf->drawables->push_back(syncObj);
27             this->dsf->unlock();
28         }
29         else
30         {
31             this->dsf->send(to, from, this->destroy, new
32             dsf::TaskArgument(syncObj));
33         }
34     });
35 }
```



```

32     this->update = new dsf::TaskFunction([this](dsf::SynchronizedObject* to, dsf::SynchronizedObject* from,
dsf::TaskArgument* args)
33         {
34             SyncFlockingBoid* syncObj;
35             std::vector<SyncFlockingBoid*>* syncObjs;
36             std::tie(syncObj, syncObjs) = args->to<
std::tuple<SyncFlockingBoid*, std::vector<SyncFlockingBoid*>*>>();
37             if(this->dsf->window->isOpen())
38             {
39                 syncObj->run(syncObjs, this->dsf->window);
40                 this->dsf->send(to, from, this->update, new
dsf::TaskArgument(std::make_tuple(syncObj, syncObjs)));
41             }
42             else
43             {
44                 this->dsf->send(to, from, this->destroy, new
dsf::TaskArgument(syncObj));
45             }
46         });
47     this->destroy = new dsf::TaskFunction([this](dsf::SynchronizedObject* to, dsf::SynchronizedObject* from
, dsf::TaskArgument* args)
48         {
49             auto syncObj = args->to<SyncFlockingBoid*>();
50             auto drawable = (sf::Drawable*) syncObj;
51             this->dsf->lock();
52             this->dsf->drawables->erase(
53                 std::remove_if(
54                     this->dsf->
55                     this->dsf->
56                     [&](sf::Drawable*
57                     {
58                         return d ==
59                         drawable;
60                     })),
61             this->dsf->drawables->end());
62             this->dsf->remove(to);
63             this->dsf->unlock();
64         });
65 FlockingBoidManager::~FlockingBoidManager()
66 {
67     delete this->create;
68     delete this->update;
69     delete this->destroy;
70 }

```

3.5 FPS

3.5.1 FPS.h

Path: \$PROJECT_DIR/profiler/include/FPS.h

```

1 //
2 // FPS.h
3 // profiler
4 //
5 // Created by Yu Chen on 2/8/15.
6 //
7 //
8
9 #ifndef profiler_FPS_h
10 #define profiler_FPS_h
11
12 #include <SFML/Graphics.hpp>
13
14 class FPS
15 {
16 public:
17     FPS(float refreshTime = 1.0, float startTime = 1.0);
18     ~FPS();
19     void refresh();
20     void restart();
21     float current;
22     float average;
23     float max;
24     float min;
25 private:
26     bool started;

```

```

27     float refreshTime;
28     float startTime;
29     sf::Clock clock;
30     sf::Clock clockFps;
31     sf::Clock clockStart;
32     float temp;
33 };
34
35 #endif

```

3.5.2 FPS.cpp

Path: \$PROJECT_DIR/profiler/src/FPS.cpp

```

1 //
2 // FPS.cpp
3 // profiler
4 //
5 // Created by Yu Chen on 2/8/15.
6 //
7 //
8
9 #include "../include/FPS.h"
10
11 FPS::FPS(float refreshTime, float startTime)
12 {
13     this->refreshTime = refreshTime;
14     this->startTime = startTime;
15 }
16
17 FPS::~FPS()
18 {
19 }
20
21
22 void FPS::refresh()
23 {
24     if(started)
25     {
26         if(!this->temp)
27             this->temp = 1.0f / this->clockFps.getElapsedTime().asSeconds();
28         else
29             this->temp = (this->temp + 1.0f / this->clockFps.getElapsedTime().asSeconds()) / 2.0f;
30         this->clockFps.restart();
31         if(this->clock.getElapsedTime().asSeconds() >= refreshTime)
32         {
33             this->current = this->temp;
34             this->temp = 0;
35             if(this->average)
36                 this->average = (this->average + this->current) / 2.0f;
37             else
38                 this->average = this->current;
39             if(!this->max || this->max < this->current)
40                 this->max = this->current;
41             if(!this->min || this->min > this->current)
42                 this->min = this->current;
43
44             this->clock.restart();
45         }
46     }
47     else if(this->clockStart.getElapsedTime().asSeconds() >= startTime)
48     {
49         this->started = true;
50         this->clock.restart();
51         this->clockFps.restart();
52         this->current = this->average = this->max = this->min = this->temp = 0;
53     }
54 }
55
56 void FPS::restart()
57 {
58     this->clockStart.restart();
59     this->started = false;
60     this->current = this->average = this->max = this->min = this->temp = 0;
61 }

```

3.6 MyDSF

3.6.1 MyDSF.h

Path: \$PROJECT_DIR/profiler/include/MyDSF.h

```

1 //
2 // MyDSF.h
3 // profiler
4 //
5 // Created by Yu Chen on 2/8/15.
6 //
7 //
8
9 #ifndef profiler_MyDSF_h
10 #define profiler_MyDSF_h
11
12 #include <dsf/DualStateFramework.h>
13 #include <dsf/TaskFunction.h>
14 #include <dsf/Lock.h>
15 #include "DSFSFML.h"
16 #include "FPS.h"
17 #include "ResourcePath.hpp"
18
19 class MyDSF : public dsf::DualStateFramework, public dsf::sfml::RenderWindow, public dsf::Lock
20 {
21 private:
22     class Sender;
23     FPS* fps;
24     std::vector<std::tuple<float,float,float>> fpsList;
25     sf::Clock* clock;
26     sf::Font font;
27     unsigned int numberOfCores = 1;
28     std::vector<std::tuple<float,float,float>> stretch(std::vector<std::tuple<float,float,float>> arr,
29                                                       std::vector<std::tuple<float,float,float>> strelen,
30                                                       int maxlen);
31 public:
32     MyDSF();
33     ~MyDSF();
34     void initialize() override;
35     Sender* sender;
36     unsigned int duration = 10;
37     unsigned int maxNumberOfCores = 8;
38 protected:
39     void refresh() override;
40     void run() override;
41     void draw() override;
42 };
43
44 class MyDSF::Sender : public dsf::SynchronizedObject
45 {
46 public:
47     Sender(dsf::DualStateFramework* dsf);
48     ~Sender();
49     dsf::DualStateFramework* dsf;
50     dsf::TaskFunction* create;
51     dsf::TaskFunction* update;
52     dsf::TaskFunction* destroy;
53 protected:
54     void run() override;
55 };
56
57
58 #endif

```

3.6.2 MyDSF.cpp

Path: \$PROJECT_DIR/profiler/src/MyDSF.cpp

```

1 //
2 // MyDSF.cpp
3 // profiler
4 //
5 // Created by Yu Chen on 2/8/15.
6 //
7 //
8
9 #include "../include/MyDSF.h"
10 #include "../include/FPS.h"
11 #include <dsf/TaskArgument.h>
12 #include <SFML/Graphics.hpp>
13 #include <iostream>
14 #include <yctools/Random.h>

```

```

15
16 MyDSF::MyDSF()
17 : DualStateFramework()
18 {
19     this->initialize();
20 }
21 MyDSF::~MyDSF()
22 {
23     delete this->fps;
24     delete this->clock;
25 }
26
27 void MyDSF::initialize()
28 {
29     this->sender = new Sender(this);
30     this->fps = new FPS();
31     this->clock = new sf::Clock();
32     this->add(this->sender);
33     this->send(this->sender, this->sender, this->sender->create, new dsf::TaskArgument((
    dsf::sfml::RenderWindow*)this));
34     this->window->create(sf::VideoMode(800, 600), "DSF Profiler");
35     this->setNumberOfThreads(numberOfCores);
36     this->font.loadFromFile(resourcePath() + "sansation.ttf");
37 }
38
39 void MyDSF::refresh()
40 {
41     dsf::DualStateFramework::refresh();
42 }
43
44 void MyDSF::run()
45 {
46     if(this->numberOfCores <= this->maxNumberOfCores
47         && this->clock->getElapsedTime().asSeconds() >= this->duration)
48     {
49         this->fpsList.push_back(std::make_tuple(this->fps->average,
50                                                 this->fps->min,
51                                                 this->fps->max));
52         this->clock->restart();
53         this->fps->restart();
54         this->numberOfCores ++;
55         this->setNumberOfThreads(this->numberOfCores);
56     }
57     dsf::DualStateFramework::run();
58     if (this->window->isOpen())
59     {
60         sf::Event event;
61         while (this->window->pollEvent(event))
62         {
63             if (event.type == sf::Event::Closed)
64             {
65                 this->window->close();
66             }
67         }
68         this->window->clear();
69         this->draw();
70         this->fps->refresh();
71         if(this->numberOfCores <= this->maxNumberOfCores)
72         {
73             if(this->fps->current)
74             {
75                 std::string msg = "Number of Core: " + std::to_string(this->numberOfCores) + "\n";
76                 msg += "FPS \n";
77                 msg += " Current: " + std::to_string(fps->current) + "\n";
78                 msg += " Average: " + std::to_string(fps->average) + "\n";
79                 msg += " Min: " + std::to_string(fps->min) + "\n";
80                 msg += " Max: " + std::to_string(fps->max);
81                 sf::Text text(msg, font);
82                 this->window->draw(text);
83             }
84         }
85         else
86         {
87             const float width = 700;
88             const float height = 500;
89             const sf::Vector2<float> origin(50, 550);
90             const float thickness = 3;
91             const float barThickness = 10;
92             sf::RectangleShape x(sf::Vector2<float>(width, thickness));
93             sf::RectangleShape y(sf::Vector2<float>(thickness, height));
94             sf::RectangleShape fill(sf::Vector2<float>(thickness, thickness));
95             x.setPosition(origin);
96             y.setPosition(origin);
97             fill.setPosition(origin - sf::Vector2<float>(thickness, 0));
98             y.rotate(180);
99             this->window->draw(x);
100             this->window->draw(y);

```

```

101         this->window->draw(fill);
102         auto bars = stretch(this->fpsList, this->fpsList, height);
103         for(int i = 0; i < this->fpsList.size(); i++)
104         {
105             unsigned int charSize = 12;
106             float x = width / (this->fpsList.size() + 1) * (i + 1);
107             float average;
108             float min;
109             float max;
110             float averageBar;
111             float minBar;
112             float maxBar;
113             std::tie(averageBar, minBar, maxBar) = bars[i];
114             std::tie(average, min, max) = this->fpsList[i];
115             sf::RectangleShape bar(sf::Vector2<float>(barThickness, maxBar - minBar));
116             bar.setPosition(origin + sf::Vector2<float>(x, -minBar));
117             bar.rotate(180);
118             sf::Text textAverage(std::to_string(average), this->font);
119             textAverage.setCharacterSize(charSize);
120             textAverage.setPosition(origin + sf::Vector2<float>(x, -averageBar));
121             sf::Text textMin(std::to_string(min), this->font);
122             textMin.setCharacterSize(charSize);
123             textMin.setPosition(origin + sf::Vector2<float>(x, -minBar));
124             sf::Text textMax(std::to_string(max), this->font);
125             textMax.setCharacterSize(charSize);
126             textMax.setPosition(origin + sf::Vector2<float>(x, -maxBar));
127             sf::Text textCore(std::to_string(i + 1), this->font);
128             textCore.setCharacterSize(14);
129             textCore.setPosition(origin + sf::Vector2<float>(x, 0));
130             this->window->draw(bar);
131             this->window->draw(textAverage);
132             this->window->draw(textMin);
133             this->window->draw(textMax);
134             this->window->draw(textCore);
135         }
136     }
137     this->window->display();
138 }
139 }
140
141 void MyDSF::draw()
142 {
143     std::for_each(this->drawables->begin(), this->drawables->end(), [this](sf::Drawable* drawable)
144     {
145         this->window->draw(*drawable);
146     });
147 }
148
149 MyDSF::Sender::Sender(dsf::DualStateFramework* dsf) : SynchronizedObject()
150 {
151     this->dsf = dsf;
152     this->create = new dsf::TaskFunction([this](dsf::SynchronizedObject* to, dsf::SynchronizedObject* from,
153     dsf::TaskArgument* args)
154     {
155         auto rw = args->to<dsf::sfml::RenderWindow*>();
156         if(rw->window->isOpen())
157         {
158             this->dsf->send(to, from, this->update, new
159             dsf::TaskArgument(rw));
160         }
161         else
162         {
163             this->dsf->send(to, from, this->destroy, nullptr);
164         }
165     });
166     this->update = new dsf::TaskFunction([this](dsf::SynchronizedObject* to, dsf::SynchronizedObject* from,
167     dsf::TaskArgument* args)
168     {
169         auto rw = args->to<dsf::sfml::RenderWindow*>();
170         if(rw->window->isOpen())
171         {
172             this->dsf->send(to, from, this->update, new
173             dsf::TaskArgument(rw));
174         }
175         else
176         {
177             this->dsf->send(to, from, this->destroy, nullptr);
178         }
179     });
180     this->destroy = new dsf::TaskFunction([this](dsf::SynchronizedObject* to, dsf::SynchronizedObject* from,
181     dsf::TaskArgument* args)
182     {
183         this->dsf->remove(to);
184     });
185 }
186
187 MyDSF::Sender::~Sender()

```

```

183 {
184     delete this->create;
185     delete this->update;
186     delete this->destroy;
187 }
188
189 void MyDSF::Sender::run()
190 {
191     if(this->receive())
192         this->process();
193 }
194
195 std::vector<std::tuple<float, float, float>> MyDSF::stretch(std::vector<std::tuple<float, float, float>> arr,
196                                                         std::vector<std::tuple<float, float, float>> strelen,
197                                                         int maxlen)
198 {
199     bool canDouble = true;
200     for(int i = 0; i < arr.size(); i++)
201     {
202         float average;
203         float min;
204         float max;
205         float averageOrigin;
206         float minOrigin;
207         float maxOrigin;
208         std::tie(average, min, max) = arr[i];
209         std::tie(averageOrigin, minOrigin, maxOrigin) = strelen[i];
210         if(max + maxOrigin > maxlen)
211         {
212             canDouble = false;
213             break;
214         }
215     }
216     if(canDouble)
217     {
218         for(int i = 0; i < arr.size(); i++)
219         {
220             float average;
221             float min;
222             float max;
223             float averageOrigin;
224             float minOrigin;
225             float maxOrigin;
226             std::tie(average, min, max) = arr[i];
227             std::tie(averageOrigin, minOrigin, maxOrigin) = strelen[i];
228             arr[i] = std::make_tuple(average + averageOrigin,
229                                     min + minOrigin,
230                                     max + maxOrigin);
231         }
232         return stretch(arr, strelen, maxlen);
233     }
234     return arr;
235 }

```

3.7 RandomCircleManager

3.7.1 RandomCircleManager.h

Path: \$PROJECT_DIR/profiler/include/RandomCircleManager.h

```

1 //
2 // RandomCircleManager.h
3 // profiler
4 //
5 // Created by Yu Chen on 2/21/15.
6 //
7 //
8
9 #ifndef profiler_RandomCircleManager_h
10 #define profiler_RandomCircleManager_h
11
12 #include "MyDSF.h"
13 #include "SyncCircle.h"
14 #include <yctools/Random.h>
15
16 class RandomCircleManager
17 {
18 public:
19     RandomCircleManager(MyDSF* dsf);
20     ~RandomCircleManager();
21     MyDSF* dsf;

```

```

22     dsf::TaskFunction* create;
23     dsf::TaskFunction* update;
24     dsf::TaskFunction* destroy;
25 };
26
27 #endif

```

3.7.2 RandomCircleManager.cpp

Path: \$PROJECT_DIR/profiler/src/RandomCircleManager.cpp

```

1 //
2 // RandomCircleManager.cpp
3 // profiler
4 //
5 // Created by Yu Chen on 2/21/15.
6 //
7 //
8
9 #include "../include/RandomCircleManager.h"
10
11 RandomCircleManager::RandomCircleManager(MyDSF* dsf)
12 {
13     this->dsf = dsf;
14     this->create = new dsf::TaskFunction([this](dsf::SynchronizedObject* to, dsf::SynchronizedObject* from,
15         dsf::TaskArgument* args)
16     {
17         auto syncObj = args->to<SyncCircle*>();
18         if(this->dsf->window->isOpen())
19         {
20             this->dsf->send(to, from, this->update, new
21             dsf::TaskArgument(syncObj));
22             this->dsf->lock();
23             this->dsf->drawables->push_back(syncObj);
24             this->dsf->unlock();
25         }
26         else
27         {
28             this->dsf->send(to, from, this->destroy, new
29             dsf::TaskArgument(syncObj));
30         }
31     });
32     this->update = new dsf::TaskFunction([this](dsf::SynchronizedObject* to, dsf::SynchronizedObject* from,
33         dsf::TaskArgument* args)
34     {
35         auto syncObj = args->to<SyncCircle*>();
36         if(this->dsf->window->isOpen())
37         {
38             auto size = this->dsf->window->getSize();
39             auto radius = syncObj->getRadius();
40             syncObj->setPosition(
41                 yc::Random().randInt(0, size.x - 2 *
42                 radius),
43                 yc::Random().randInt(0, size.y - 2 *
44                 radius)
45             );
46             this->dsf->send(to, from, this->update, new
47             dsf::TaskArgument(syncObj));
48         }
49         else
50         {
51             this->dsf->send(to, from, this->destroy, new
52             dsf::TaskArgument(syncObj));
53         }
54     });
55     this->destroy = new dsf::TaskFunction([this](dsf::SynchronizedObject* to, dsf::SynchronizedObject* from,
56         dsf::TaskArgument* args)
57     {
58         auto syncObj = args->to<SyncCircle*>();
59         auto drawable = (sf::Drawable*) syncObj;
60         this->dsf->lock();
61         this->dsf->drawables->erase(
62             std::remove_if(
63                 this->dsf->
64                 drawables->begin(),
65                 this->dsf->
66                 drawables->end(),
67                 [&](sf::Drawable*
68                 d)
69                 {
70                     return d ==
71                     drawable;
72                 })
73             ),
74         );
75     });
76 }

```

```

60                                     this->dsf->drawables->end());
61                                     this->dsf->remove(to);
62                                     this->dsf->unlock();
63                                 });
64 }
65 RandomCircleManager::~RandomCircleManager()
66 {
67     delete this->create;
68     delete this->update;
69     delete this->destroy;
70 }

```

3.8 ResourcePath

3.8.1 ResourcePath.hpp

Path: `$PROJECT_DIR/profiler/include/ResourcePath.hpp`

```

1 //
2 // SFML - Simple and Fast Multimedia Library
3 // Copyright (C) 2007-2013 Marco Antognini (antognini.marco@gmail.com),
4 //                                     Laurent Gomila (laurent.gom@gmail.com),
5 //                                     Laurent Gomila (laurent.gom@gmail.com),
6 //
7 // This software is provided 'as-is', without any express or implied warranty.
8 // In no event will the authors be held liable for any damages arising from the use of this software.
9 //
10 // Permission is granted to anyone to use this software for any purpose,
11 // including commercial applications, and to alter it and redistribute it freely,
12 // subject to the following restrictions:
13 //
14 // 1. The origin of this software must not be misrepresented;
15 //    you must not claim that you wrote the original software.
16 //    If you use this software in a product, an acknowledgment
17 //    in the product documentation would be appreciated but is not required.
18 //
19 // 2. Altered source versions must be plainly marked as such,
20 //    and must not be misrepresented as being the original software.
21 //
22 // 3. This notice may not be removed or altered from any source distribution.
23 //
24
25
26 #ifndef RESOURCE_PATH_HPP
27 #define RESOURCE_PATH_HPP
28
29 // Headers
30 #include <string>
31
32 std::string resourcePath(void);
33
34 #endif

```

3.8.2 ResourcePath.cpp

Path: `$PROJECT_DIR/profiler/src/ResourcePath.cpp`

```

1 #include "../include/ResourcePath.hpp"
2
3 std::string resourcePath(void)
4 {
5     return "Resources/";
6 }

```

3.8.3 ResourcePath.mm

Path: `$PROJECT_DIR/profiler/src/ResourcePath.mm`

```

1 //
2 // SFML - Simple and Fast Multimedia Library
3 // Copyright (C) 2007-2013 Marco Antognini (antognini.marco@gmail.com),
4 //                                     Laurent Gomila (laurent.gom@gmail.com),
5 //                                     Laurent Gomila (laurent.gom@gmail.com),
6 //
7 // This software is provided 'as-is', without any express or implied warranty.

```



```

8 // In no event will the authors be held liable for any damages arising from the use of this software.
9 //
10 // Permission is granted to anyone to use this software for any purpose,
11 // including commercial applications, and to alter it and redistribute it freely,
12 // subject to the following restrictions:
13 //
14 // 1. The origin of this software must not be misrepresented;
15 //    you must not claim that you wrote the original software.
16 //    If you use this software in a product, an acknowledgment
17 //    in the product documentation would be appreciated but is not required.
18 //
19 // 2. Altered source versions must be plainly marked as such,
20 //    and must not be misrepresented as being the original software.
21 //
22 // 3. This notice may not be removed or altered from any source distribution.
23 //
24
25
26 // Headers
27 #include "../include/ResourcePath.hpp"
28 #import <Foundation/Foundation.h>
29
30 std::string resourcePath(void)
31 {
32     NSAutoreleasePool* pool = [[NSAutoreleasePool alloc] init];
33
34     std::string rpath;
35     NSBundle* bundle = [NSBundle mainBundle];
36
37     if (bundle == nil) {
38 #ifdef DEBUG
39         NSLog(@"bundle is nil... thus no resources path can be found.");
40 #endif
41     } else {
42         NSString* path = [bundle resourcePath];
43         rpath = [path UTF8String] + std::string("/");
44     }
45
46     [pool drain];
47
48     return rpath;
49 }

```

3.9 SyncBouncingCircle

3.9.1 SyncBouncingCircle.h

Path: \$PROJECT_DIR/profiler/include/SyncBouncingCircle.h

```

1 //
2 // SyncBouncingCircle.h
3 // profiler
4 //
5 // Created by Yu Chen on 2/21/15.
6 //
7 //
8
9 #ifndef profiler_SyncBouncingCircle_h
10 #define profiler_SyncBouncingCircle_h
11
12 #include <dsf/SynchronizedObject.h>
13 #include <SFML/Graphics.hpp>
14 #include <cmath>
15
16 class SyncBouncingCircle : public dsf::SynchronizedObject, public sf::CircleShape
17 {
18 public:
19     SyncBouncingCircle();
20     sf::Vector2<float> getVelocity();
21     void setVelocity(const sf::Vector2<float>& velocity);
22     float getMass();
23     void move(int width, int height);
24     void collide(SyncBouncingCircle* sbc);
25     bool hasCollision(SyncBouncingCircle* sbc);
26 protected:
27     void run() override;
28 private:
29     sf::Vector2<float> velocity;
30     float mass = 1;
31 };
32
33 #endif

```

3.9.2 SyncBouncingCircle.cpp

Path: \$PROJECT_DIR/profiler/src/SyncBouncingCircle.cpp

```

1 //
2 // SyncBouncingCircle.cpp
3 // profiler
4 //
5 // Created by Yu Chen on 2/21/15.
6 //
7 //
8
9 #include "../include/SyncBouncingCircle.h"
10
11
12 SyncBouncingCircle::SyncBouncingCircle() : SynchronizedObject(), CircleShape()
13 {
14 }
15
16 void SyncBouncingCircle::run()
17 {
18     if(this->receive())
19         this->process();
20 }
21
22 sf::Vector2<float> SyncBouncingCircle::getVelocity() {
23     return this->velocity;
24 }
25 void SyncBouncingCircle::setVelocity(const sf::Vector2<float>& velocity) {
26     this->velocity = velocity;
27 }
28 float SyncBouncingCircle::getMass() {
29     return this->mass;
30 }
31 void SyncBouncingCircle::move(int width, int height) {
32     sf::Vector2<float> nextPosition = this->getPosition() + this->velocity;
33     this->setPosition(nextPosition);
34     if (getPosition().x <= 0 || getPosition().x >= width - this->getRadius() * 2) {
35         this->velocity = sf::Vector2<float>(-this->velocity.x, this->velocity.y);
36     }
37     if (getPosition().y <= 0 || getPosition().y >= height - this->getRadius() * 2) {
38         this->velocity = sf::Vector2<float>(this->velocity.x, -this->velocity.y);
39     }
40 }
41
42 void SyncBouncingCircle::collide(SyncBouncingCircle *sbc)
43 {
44     if (this->hasCollision(sbc))
45     {
46         sf::Vector2<float> v1 = this->getVelocity();
47         sf::Vector2<float> v2 = sbc->getVelocity();
48         sf::Vector2<float> pos1 = this->getPosition();
49         sf::Vector2<float> pos2 = sbc->getPosition();
50         sf::Vector2<float> n = sf::Vector2<float>(pos2.x - pos1.x, pos2.y - pos1.y);
51         sf::Vector2<float> un = n / sqrtf(n.x * n.x + n.y * n.y);
52         sf::Vector2<float> ut = sf::Vector2<float>(-un.y, un.x);
53         float v1n = un.x * v1.x + un.y * v1.y;
54         float v1t = ut.x * v1.x + ut.y * v1.y;
55         float v2n = un.x * v2.x + un.y * v2.y;
56         float v2t = ut.x * v2.x + ut.y * v2.y;
57         float m1 = this->getMass();
58         float m2 = sbc->getMass();
59         float v_1t = v1t;
60         float v_2t = v2t;
61         float v_1n = (v1n * (m1 - m2) + 2 * m2 * v2n) / (m1 + m2);
62         float v_2n = (v2n * (m2 - m1) + 2 * m1 * v1n) / (m1 + m2);
63         sf::Vector2<float> v__1n = v_1n * un;
64         sf::Vector2<float> v__1t = v_1t * ut;
65         sf::Vector2<float> v__2n = v_2n * un;
66         sf::Vector2<float> v__2t = v_2t * ut;
67         sf::Vector2<float> v_1 = v__1n + v__1t;
68         sf::Vector2<float> v_2 = v__2n + v__2t;
69         this->setVelocity(v_1);
70         sbc->setVelocity(v_2);
71     }
72 }
73
74 bool SyncBouncingCircle::hasCollision(SyncBouncingCircle *sbc)
75 {
76     float distanceSqr = std::pow(this->getPosition().x - sbc->getPosition().x, 2) + std::pow(this->
77         getPosition().y - sbc->getPosition().y, 2);
78     return distanceSqr <= std::pow(this->getRadius() + sbc->getRadius(), 2);
79 }

```

3.10 SyncCircle

3.10.1 SyncCircle.h

Path: `$PROJECT_DIR/profiler/include/SyncCircle.h`

```

1 //
2 //  SyncCircle.h
3 //  profiler
4 //
5 //  Created by Yu Chen on 2/21/15.
6 //
7 //
8
9 #ifndef profiler_SyncCircle_h
10 #define profiler_SyncCircle_h
11
12 #include <dsf/SynchronizedObject.h>
13 #include <SFML/Graphics.hpp>
14
15 class SyncCircle : public dsf::SynchronizedObject, public sf::CircleShape
16 {
17 public:
18     SyncCircle();
19 protected:
20     void run() override;
21 };
22
23 #endif

```

3.10.2 SyncCircle.cpp

Path: `$PROJECT_DIR/profiler/src/SyncCircle.cpp`

```

1 //
2 //  SyncCircle.cpp
3 //  profiler
4 //
5 //  Created by Yu Chen on 2/21/15.
6 //
7 //
8
9 #include "../include/SyncCircle.h"
10
11 SyncCircle::SyncCircle() : SynchronizedObject(), CircleShape()
12 {
13 }
14
15 void SyncCircle::run()
16 {
17     if (this->receive())
18         this->process();
19 }

```

3.11 SyncFlockingBoid

3.11.1 SyncFlockingBoid.h

Path: `$PROJECT_DIR/profiler/include/SyncFlockingBoid.h`

```

1 //
2 //  SyncFlockingBoid.h
3 //  profiler
4 //
5 //  Created by Yu Chen on 3/12/15.
6 //
7 //
8
9 #ifndef profiler_SyncFlockingBoid_h
10 #define profiler_SyncFlockingBoid_h
11
12 #include <dsf/SynchronizedObject.h>

```

```

13 #include <dsf/SynchronizedVar.h>
14 #include <SFML/Graphics.hpp>
15 #include <yctools/Random.h>
16 #include "SyncVector3D.h"
17
18 class SyncFlockingBoid : public dsf::SynchronizedObject, public sf::CircleShape
19 {
20 public:
21     SyncVector3D* loc;
22     SyncVector3D* vel;
23     SyncVector3D* acc;
24     float r;
25     float maxforce;    // Maximum steering force
26     float maxspeed;    // Maximum speed
27     SyncFlockingBoid();
28     ~SyncFlockingBoid();
29     SyncFlockingBoid(Vector3D* loc, float ms, float mf);
30     void run(std::vector<SyncFlockingBoid*>* boids, sf::RenderWindow* window);
31     Vector3D steer(Vector3D* target, bool slowdown);
32     Vector3D separate (std::vector<SyncFlockingBoid*>* boids);
33     Vector3D align (std::vector<SyncFlockingBoid*>* boids);
34     Vector3D cohesion (std::vector<SyncFlockingBoid*>* boids);
35 protected:
36     void run() override;
37 };
38
39 #endif

```

3.11.2 SyncFlockingBoid.cpp

Path: \$PROJECT_DIR/profiler/src/SyncFlockingBoid.cpp

```

1 //
2 // SyncFlockingBoid.cpp
3 // profiler
4 //
5 // Created by Yu Chen on 3/12/15.
6 //
7 //
8
9 #include "../include/SyncFlockingBoid.h"
10
11 SyncFlockingBoid::SyncFlockingBoid() : SynchronizedObject(), CircleShape()
12 {
13 }
14 SyncFlockingBoid::~SyncFlockingBoid()
15 {
16     delete this->acc;
17     delete this->vel;
18     delete this->loc;
19 }
20 SyncFlockingBoid::SyncFlockingBoid(Vector3D* loc, float ms, float mf) : SynchronizedObject(), CircleShape()
21 {
22     this->acc = new SyncVector3D(0,0);
23     this->vel = new SyncVector3D(yctools::Random().randFloat(-1,1), yctools::Random().randFloat(-1,1));
24     this->loc = new SyncVector3D(loc->getX(), loc->getY(), loc->getZ());
25     delete loc;
26     r = 2.0f;
27     maxspeed = ms;
28     maxforce = mf;
29 }
30
31 void SyncFlockingBoid::run(std::vector<SyncFlockingBoid*>* boids, sf::RenderWindow* window) {
32     //flock(boids);
33     //update();
34     //borders(window->getSize().x, window->getSize().y);
35     //render(window);
36
37     // We accumulate a new acceleration each time based on three rules
38     Vector3D sep = separate(boids); // Separation
39     Vector3D ali = align(boids);    // Alignment
40     Vector3D coh = cohesion(boids); // Cohesion
41
42     // Arbitrarily weight these forces
43     sep *= 2.0f;
44     ali *= 1.0f;
45     coh *= 1.0f;
46
47     auto acc = Vector3D(*this->acc);
48     auto loc = Vector3D(this->loc->getX(), this->loc->getY());
49     auto vel = Vector3D(*this->vel);
50
51     // Add the force vectors to acceleration

```

```

52     acc += sep;
53     acc += ali;
54     acc += coh;
55
56     // Method to update location
57     // Update velocity
58     vel += acc;
59
60     // Limit speed
61     vel.limit(maxspeed);
62
63     loc += vel;
64
65     // Reset acceleration to 0 each cycle
66     acc.setXYZ(0,0,0);
67
68     // Wraparound
69     auto width = window->getSize().x;
70     auto height = window->getSize().y;
71     if (loc.getX() < -r)
72         loc.setX(width+r);
73     if (loc.getY() < -r)
74         loc.setY(height+r);
75     if (loc.getX() > width+r)
76         loc.setX(-r);
77     if (loc.getY() > height+r)
78         loc.setY(-r);
79
80     this->acc->setXYZ(acc);
81     this->loc->setXYZ(loc);
82     this->vel->setXYZ(vel);
83     this->setPosition(this->loc->getX(), this->loc->getY());
84     this->setRadius(r);
85 }
86
87
88
89 // A method that calculates a steering vector towards a target
90 // Takes a second argument, if true, it slows down as it approaches the target
91
92 Vector3D SyncFlockingBoid::steer(Vector3D* target, bool slowdown) {
93     Vector3D steer; // The steering vector
94     Vector3D desired = *target - *this->loc; // A vector pointing from the location to the target
95     float d = desired.magnitude(); // Distance from the target is the magnitude of the vector
96
97     // If the distance is greater than 0, calc steering (otherwise return zero vector)
98     if (d > 0) {
99         // Normalize desired
100         desired.normalize();
101
102         // Two options for desired vector magnitude (1 -- based on distance, 2 -- maxspeed)
103         if ((slowdown) && (d < 100.0f))
104             desired *= maxspeed * (d / 100.0f); // This damping is somewhat arbitrary
105         else
106             desired *= maxspeed;
107
108         // Steering = Desired minus Velocity
109         steer = desired - *this->vel;
110         steer.limit(maxforce); // Limit to maximum steering force
111
112     } else {
113         steer = Vector3D(0,0);
114     }
115     return steer;
116 }
117
118 // Separation
119 // Method checks for nearby boids and steers away
120 Vector3D SyncFlockingBoid::separate (std::vector<SyncFlockingBoid*>* boids) {
121     float desiredseparation = 25.0f;
122     Vector3D sum = Vector3D(0,0,0);
123     int count = 0;
124
125     // For every boid in the system, check if it's too close
126     for (auto other: *boids) {
127         float d = this->loc->distance(*other->loc);
128
129         // If the distance is greater than 0 and less than an arbitrary amount (0 when you are yourself)
130         if ((d > 0) && (d < desiredseparation)) {
131             // Calculate vector pointing away from neighbor
132             Vector3D diff = *this->loc - *other->loc;
133             diff.normalize();
134             diff /= d; // Weight by distance
135             sum += diff;
136             count++; // Keep track of how many
137         }
138     }

```

```

139
140     // Average -- divide by how many
141     if (count > 0)
142         sum /= count;
143     return sum;
144 }
145
146
147
148 // Alignment
149 // For every nearby boid in the system, calculate the average velocity
150 Vector3D SyncFlockingBoid::align (std::vector<SyncFlockingBoid*>* boids) {
151     float neighbordist = 50.0f;
152     Vector3D sum = Vector3D(0,0,0);
153     int count = 0;
154     for (auto & other : *boids) {
155         float d = this->loc->distance(*other->loc);
156         if ((d > 0) && (d < neighbordist)) {
157             sum += *other->vel;
158             count++;
159         }
160     }
161     if (count > 0) {
162         sum /= count;
163         sum.limit(maxforce);
164     }
165     return sum;
166 }
167
168 // Cohesion
169 // For the average location (i.e. center) of all nearby boids, calculate steering vector towards that
    location
170 Vector3D SyncFlockingBoid::cohesion (std::vector<SyncFlockingBoid*>* boids) {
171     float neighbordist = 50.0f;
172     Vector3D sum = Vector3D(0,0,0);    // Start with empty vector to accumulate all locations
173     int count = 0;
174     for (auto & other: *boids) {
175         float d = this->loc->distance(*other->loc);
176         if ((d > 0) && (d < neighbordist)) {
177             sum += *other->loc; // Add location
178             count++;
179         }
180     }
181
182     if (count > 0) {
183         sum /= count;
184         return steer(&sum, false); // Steer towards the location
185     }
186     return sum;
187 }
188
189 void SyncFlockingBoid::run()
190 {
191     if(this->receive())
192     {
193         this->loc->synchronise();
194         this->acc->synchronise();
195         this->vel->synchronise();
196         this->process();
197     }
198 }

```

3.12 SyncVector3D

3.12.1 SyncVector3D.h

Path: \$PROJECT_DIR/profiler/include/SyncVector3D.h

```

1 //
2 // SyncVector3D.h
3 // profiler
4 //
5 // Created by Yu Chen on 3/22/15.
6 //
7 //
8
9 #ifndef profiler_SyncVector3D_h
10 #define profiler_SyncVector3D_h
11
12 #include "Vector3D.h"
13 #include <dsf/Synchronisable.h>

```

```

14
15 class SyncVector3D : public dsf::Synchronisable<Vector3D>, public Vector3D {
16 public:
17     explicit SyncVector3D(float x=0, float y=0, float z=0);
18     void setX(float x) override;
19     void setY(float y) override;
20     void setZ(float z) override;
21     void add(const Vector3D& v) override;
22     void sub(const Vector3D& v) override;
23     void mul(float n) override;
24     void div(float n) override;
25     void synchronise() override;
26 };
27
28 #endif

```

3.12.2 SyncVector3D.cpp

Path: \$PROJECT_DIR/profiler/src/SyncVector3D.cpp

```

1 //
2 // Vector3D.cpp
3 // profiler
4 //
5 // Created by Yu Chen on 3/22/15.
6 //
7 //
8
9 #include "../include/SyncVector3D.h"
10
11
12 SyncVector3D::SyncVector3D(float x, float y, float z) : Vector3D(x, y, z) {
13     this->next = new Vector3D(x, y, z);
14 }
15
16 void SyncVector3D::setX(float x) {
17     this->next->setX(x);
18 }
19
20 void SyncVector3D::setY(float y) {
21     this->next->setY(y);
22 }
23
24 void SyncVector3D::setZ(float z) {
25     this->next->setZ(z);
26 }
27
28 void SyncVector3D::add(const Vector3D& v) {
29     this->next->add(v);
30 }
31
32 void SyncVector3D::sub(const Vector3D& v) {
33     this->next->sub(v);
34 }
35
36 void SyncVector3D::mul(float n) {
37     this->next->mul(n);
38 }
39
40 void SyncVector3D::div(float n) {
41     this->next->div(n);
42 }
43
44 void SyncVector3D::synchronise() {
45     this->x = this->next->getX();
46     this->y = this->next->getY();
47     this->z = this->next->getZ();
48 }

```

3.13 Vector3D

3.13.1 Vector3D.h

Path: \$PROJECT_DIR/profiler/include/Vector3D.h

```

1 //
2 // Vector3D.h

```

```

3 // profiler
4 //
5 // Created by Yu Chen on 3/22/15.
6 //
7 //
8
9 #ifndef profiler_Vector3D_h
10 #define profiler_Vector3D_h
11
12 #include <cmath>
13
14 class Vector3D {
15 protected:
16     float x;
17     float y;
18     float z;
19 public:
20     explicit Vector3D(float x=0, float y=0, float z=0);
21     virtual ~Vector3D();
22     virtual void operator=(const Vector3D& obj);
23
24     float getX();
25     float getY();
26     float getZ();
27
28     virtual void setX(float x);
29     virtual void setY(float y);
30     virtual void setZ(float z);
31
32     void setXY(float x, float y);
33     void setXYZ(float x, float y, float z);
34     void setXYZ(const Vector3D& v);
35
36     float magnitude();
37     void normalize();
38     void limit(float max);
39     virtual float heading2D();
40     float distance (const Vector3D& v);
41
42     virtual void add(const Vector3D& v);
43     void operator+=(const Vector3D& v);
44     virtual void sub(const Vector3D& v);
45     void operator-=(const Vector3D& v);
46     virtual void mul(float n);
47     void operator*=(float n);
48     virtual void div(float n);
49     void operator/=(float n);
50
51
52     Vector3D operator+(const Vector3D& v) const;
53     Vector3D operator-(const Vector3D& v) const;
54     Vector3D operator/(float n) const;
55     Vector3D operator*(float n) const;
56
57 };
58
59 #endif

```

3.13.2 Vector3D.cpp

Path: \$PROJECT_DIR/profiler/src/Vector3D.cpp

```

1 //
2 // Vector3D.cpp
3 // profiler
4 //
5 // Created by Yu Chen on 3/22/15.
6 //
7 //
8
9 #include "../include/Vector3D.h"
10
11
12 Vector3D::Vector3D(float x, float y, float z) : x(x), y(y), z(z) {
13 }
14
15 Vector3D::~Vector3D() {
16 }
17
18 void Vector3D::operator=(const Vector3D& obj)
19 {
20     this->setXYZ(obj.x, obj.y, obj.z);
21 }

```



```
22
23 float Vector3D::getX() {
24     return this->x;
25 }
26 float Vector3D::getY() {
27     return this->y;
28 }
29 float Vector3D::getZ() {
30     return this->z;
31 }
32
33 void Vector3D::setX(float x) {
34     this->x = x;
35 }
36
37 void Vector3D::setY(float y) {
38     this->y = y;
39 }
40
41 void Vector3D::setZ(float z) {
42     this->z = z;
43 }
44
45 void Vector3D::setXY(float x, float y) {
46     this->setX(x);
47     this->setY(y);
48 }
49
50 void Vector3D::setXYZ(float x, float y, float z) {
51     this->setXY(x, y);
52     this->setZ(z);
53 }
54
55 void Vector3D::setXYZ(const Vector3D& v) {
56     this->setXYZ(v.x, v.y, v.z);
57 }
58
59 float Vector3D::magnitude() {
60     return std::sqrt(std::pow(this->getX(), 2)
61                     + std::pow(this->getY(), 2)
62                     + std::pow(this->getZ(), 2));
63 }
64
65 void Vector3D::normalize() {
66     float m = magnitude();
67     if (m > 0)
68         div(m);
69 }
70
71 void Vector3D::limit(float max) {
72     if (magnitude() > max) {
73         normalize();
74         mul(max);
75     }
76 }
77
78 float Vector3D::distance (const Vector3D& v) {
79     float dx = x - v.x;
80     float dy = y - v.y;
81     float dz = z - v.z;
82     return std::sqrt(dx*dx + dy*dy + dz*dz);
83 }
84
85 float Vector3D::heading2D() {
86     return -std::atan2(-this->getY(), this->getX());
87 }
88
89 void Vector3D::add(const Vector3D& v) {
90     x += v.x;
91     y += v.y;
92     z += v.z;
93 }
94
95 void Vector3D::operator+=(const Vector3D& v) {
96     add(v);
97 }
98
99 void Vector3D::sub(const Vector3D& v) {
100     x -= v.x;
101     y -= v.y;
102     z -= v.z;
103 }
104
105 void Vector3D::operator-=(const Vector3D& v) {
106     sub(v);
107 }
108
```

```

109 void Vector3D::mul(float n) {
110     x *= n;
111     y *= n;
112     z *= n;
113 }
114
115 void Vector3D::operator*=(float n) {
116     mul(n);
117 }
118
119 void Vector3D::div(float n) {
120     x /= n;
121     y /= n;
122     z /= n;
123 }
124
125 void Vector3D::operator/=(float n) {
126     div(n);
127 }
128
129 Vector3D Vector3D::operator+(const Vector3D& v) const {
130     return Vector3D(x + v.x, y + v.y, z + v.z);
131 }
132
133 Vector3D Vector3D::operator-(const Vector3D& v) const {
134     return Vector3D(x - v.x, y - v.y, z - v.z);
135 }
136
137 Vector3D Vector3D::operator/(float n) const {
138     return Vector3D(x/n, y/n, z/n);
139 }
140
141 Vector3D Vector3D::operator*(float n) const {
142     return Vector3D(x*n, y*n, z*n);
143 }

```

3.14 main

3.14.1 main.cpp

Path: \$PROJECT_DIR/profiler/src/main.cpp

```

1 #include <SFML/Graphics.hpp>
2 #include <dsf/DualStateFramework.h>
3 #include "../include/ResourcePath.hpp"
4 #include "../include/FPS.h"
5 #include "../include/MyDSF.h"
6 #include "../include/SyncCircle.h"
7 #include "../include/RandomCircleManager.h"
8 #include "../include/SyncBouncingCircle.h"
9 #include "../include/BouncingCircleManager.h"
10 #include "../include/SyncFlockingBoid.h"
11 #include "../include/FlockingBoidManager.h"
12
13 void profile(int maxNumberOfThreads, int numberOfObjects, int durationPerIterator, int method);
14 void configure();
15
16 // Constances
17 const sf::Vector2f WINDOW_SIZE(800, 600);
18 const int CHAR_SIZE = 20;
19 const sf::Vector2f LEFT_CORNER(100, 100);
20 const sf::Vector2f RIGHT_CORNER(700, 100);
21 const sf::Vector2f INTENT(500, 25);
22 const sf::Vector2f SHADOW_SIZE(WINDOW_SIZE.x - LEFT_CORNER.x * 2, INTENT.y);
23 const int MAX_NUMBER_OF_THREADS = 64;
24 const int MIN_NUMBER_OF_THREADS = 2;
25 const int MAX_NUMBER_OF_OBJECTS = 2000;
26 const int MIN_NUMBER_OF_OBJECTS = 100;
27 const int MAX_DURATION = 360;
28 const int MIN_DURATION = 5;
29 const int NUMBER_OF_PROFILING_METHODS = 3;
30 const std::string profilingMethods[NUMBER_OF_PROFILING_METHODS] = {
31     "Random", "Collision", "Flocking"
32 };
33 const int NUMBER_OF_SELECTIONS = 4;
34 // Default Configurations
35 int currentSelection = 1;
36 int maxNumberOfThreads = 4;
37 int numberOfObjects = 1000;
38 int durationPerIterator = 60;
39 int profilingMethodIndex = 0;

```

```

40
41 int main()
42 {
43     configure();
44     return 0;
45 }
46
47 void configure()
48 {
49     bool ready = false;
50     // Create the main window
51     sf::RenderWindow window(sf::VideoMode(WINDOW_SIZE.x, WINDOW_SIZE.y), "DSF Profiler");
52     sf::Font font;
53     sf::Text maxNumberOfThreadsText;
54     sf::Text numberOfObjectsText;
55     sf::Text durationPerIteratorText;
56     sf::Text profilingMethodText;
57     sf::Text maxNumberOfThreadsValue;
58     sf::Text numberOfObjectsValue;
59     sf::Text durationPerIteratorValue;
60     sf::Text profilingMethodValue;
61     sf::Text help;
62     font.loadFromFile(resourcePath() + "sansation.ttf");
63     maxNumberOfThreadsText.setFont(font);
64     numberOfObjectsText.setFont(font);
65     durationPerIteratorText.setFont(font);
66     profilingMethodText.setFont(font);
67     maxNumberOfThreadsValue.setFont(font);
68     numberOfObjectsValue.setFont(font);
69     durationPerIteratorValue.setFont(font);
70     profilingMethodValue.setFont(font);
71     help.setFont(font);
72     maxNumberOfThreadsText.setCharacterSize(CHAR_SIZE);
73     numberOfObjectsText.setCharacterSize(CHAR_SIZE);
74     durationPerIteratorText.setCharacterSize(CHAR_SIZE);
75     profilingMethodText.setCharacterSize(CHAR_SIZE);
76     maxNumberOfThreadsValue.setCharacterSize(CHAR_SIZE);
77     numberOfObjectsValue.setCharacterSize(CHAR_SIZE);
78     durationPerIteratorValue.setCharacterSize(CHAR_SIZE);
79     profilingMethodValue.setCharacterSize(CHAR_SIZE);
80     help.setCharacterSize(CHAR_SIZE);
81     maxNumberOfThreadsText.setPosition(LEFT_CORNER);
82     numberOfObjectsText.setPosition(LEFT_CORNER.x, LEFT_CORNER.y + INTENT.y);
83     durationPerIteratorText.setPosition(LEFT_CORNER.x, LEFT_CORNER.y + INTENT.y * 2);
84     profilingMethodText.setPosition(LEFT_CORNER.x, LEFT_CORNER.y + INTENT.y * 3);
85     maxNumberOfThreadsValue.setPosition(LEFT_CORNER.x + INTENT.x, LEFT_CORNER.y);
86     numberOfObjectsValue.setPosition(LEFT_CORNER.x + INTENT.x, LEFT_CORNER.y + INTENT.y);
87     durationPerIteratorValue.setPosition(LEFT_CORNER.x + INTENT.x, LEFT_CORNER.y + INTENT.y * 2);
88     profilingMethodValue.setPosition(LEFT_CORNER.x + INTENT.x, LEFT_CORNER.y + INTENT.y * 3);
89     help.setPosition(LEFT_CORNER.x, LEFT_CORNER.y + INTENT.y * 5);
90     maxNumberOfThreadsText.setString("Max Number of Threads");
91     numberOfObjectsText.setString("Number of Objects");
92     durationPerIteratorText.setString("Duration per Iterator");
93     profilingMethodText.setString("Profiling Method");
94     help.setString("Press KeyUp, KeyDown, KeyLeft, and KeyRight to edit settings. \nPress Enter to run the
    application.");
95
96     sf::RectangleShape shadow;
97     shadow.setFillColor(sf::Color::Blue);
98     shadow.setSize(SHADOW_SIZE);
99     shadow.setPosition(LEFT_CORNER.x, LEFT_CORNER.y + INTENT.y * (currentSelection - 1));
100     while (window.isOpen())
101     {
102         // Process events
103         sf::Event event;
104         while (window.pollEvent(event))
105         {
106             // Close window: exit
107             if (event.type == sf::Event::Closed)
108                 window.close();
109             // Keyboard Events
110             if (sf::Keyboard::isKeyPressed(sf::Keyboard::Return)) {
111                 ready = true;
112                 window.close();
113             }
114             if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up))
115             {
116                 if (currentSelection > 1)
117                 {
118                     currentSelection --;
119                     auto spos = shadow.getPosition();
120                     shadow.setPosition(spos.x, spos.y - INTENT.y);
121                 }
122             }
123             if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down))
124             {
125                 if (currentSelection < NUMBER_OF_SELECTIONS)

```

```

126         {
127             currentSelection ++;
128             auto spos = shadow.getPosition();
129             shadow.setPosition(spos.x, spos.y + INTENT.y);
130         }
131     }
132     if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left))
133     {
134         switch (currentSelection) {
135             case 1:
136                 if(maxNumberOfThreads == MIN_NUMBER_OF_THREADS)
137                     maxNumberOfThreads = MAX_NUMBER_OF_THREADS;
138                 else
139                     maxNumberOfThreads --;
140                 break;
141             case 2:
142                 if(numberOfObjects == MIN_NUMBER_OF_OBJECTS)
143                     numberOfObjects = MAX_NUMBER_OF_OBJECTS;
144                 else
145                     numberOfObjects -= MIN_NUMBER_OF_OBJECTS;
146                 break;
147             case 3:
148                 if(durationPerIterator == MIN_DURATION)
149                     durationPerIterator = MAX_DURATION;
150                 else
151                     durationPerIterator -= MIN_DURATION;
152                 break;
153             case 4:
154                 if(profilingMethodIndex == 0)
155                     profilingMethodIndex = NUMBER_OF_PROFILING_METHODS - 1;
156                 else
157                     profilingMethodIndex --;
158                 break;
159         }
160     }
161     if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right))
162     {
163         switch (currentSelection) {
164             case 1:
165                 if(maxNumberOfThreads == MAX_NUMBER_OF_THREADS)
166                     maxNumberOfThreads = MIN_NUMBER_OF_THREADS;
167                 else
168                     maxNumberOfThreads ++;
169                 break;
170             case 2:
171                 if(numberOfObjects == MAX_NUMBER_OF_OBJECTS)
172                     numberOfObjects = MIN_NUMBER_OF_OBJECTS;
173                 else
174                     numberOfObjects += MIN_NUMBER_OF_OBJECTS;
175                 break;
176             case 3:
177                 if(durationPerIterator == MAX_DURATION)
178                     durationPerIterator = MIN_DURATION;
179                 else
180                     durationPerIterator += MIN_DURATION;
181                 break;
182             case 4:
183                 if(profilingMethodIndex == NUMBER_OF_PROFILING_METHODS - 1)
184                     profilingMethodIndex = 0;
185                 else
186                     profilingMethodIndex ++;
187                 break;
188         }
189     }
190 }
191
192 maxNumberOfThreadsValue.setString(std::to_string(maxNumberOfThreads));
193 numberOfObjectsValue.setString(std::to_string(numberOfObjects));
194 durationPerIteratorValue.setString(std::to_string(durationPerIterator));
195 profilingMethodValue.setString(profilingMethods[profilingMethodIndex]);
196 // Clear screen
197 window.clear();
198 // Draw Items
199 window.draw(shadow);
200 window.draw(maxNumberOfThreadsText);
201 window.draw(numberOfObjectsText);
202 window.draw(durationPerIteratorText);
203 window.draw(profilingMethodText);
204 window.draw(maxNumberOfThreadsValue);
205 window.draw(numberOfObjectsValue);
206 window.draw(durationPerIteratorValue);
207 window.draw(profilingMethodValue);
208 window.draw(help);
209 // Update the window
210 window.display();
211 }
212 if (ready)

```

```

213         profile(maxNumberOfThreads, numberOfObjects, durationPerIterator, profilingMethodIndex + 1);
214     }
215
216 void profile(int maxNumberOfThreads, int numberOfObjects, int durationPerIterator, int method)
217 {
218     if(method == 1)
219     {
220         auto dsf = new MyDSF();
221         auto rcm = new RandomCircleManager(dsf);
222         std::vector<SyncCircle*> circles(numberOfObjects);
223         for(auto & circle : circles)
224         {
225             circle = new SyncCircle();
226             circle->setRadius(2);
227             circle->setFillColor(sf::Color::Cyan);
228             dsf->add(circle);
229             dsf->send(circle, dsf->sender, rcm->create, new dsf::TaskArgument(circle));
230         }
231         dsf->duration = durationPerIterator;
232         dsf->maxNumberOfCores = maxNumberOfThreads;
233         dsf->start();
234         delete dsf;
235         delete rcm;
236     }
237     else if(method == 2)
238     {
239         auto dsf = new MyDSF();
240         auto bcm = new BouncingCircleManager(dsf);
241         auto bouncingCircles = bcm->createRandomCircles(numberOfObjects, 2, 800, 600);
242         for(auto & bouncingCircle : *bouncingCircles)
243         {
244             bouncingCircle->setFillColor(sf::Color::Cyan);
245             dsf->add(bouncingCircle);
246             dsf->send(bouncingCircle,
247                     dsf->sender,
248                     bcm->create,
249                     new dsf::TaskArgument(std::make_tuple(bouncingCircle, bouncingCircles)));
250         }
251         dsf->duration = durationPerIterator;
252         dsf->maxNumberOfCores = maxNumberOfThreads;
253         dsf->start();
254         delete dsf;
255         delete bcm;
256     }
257     else if(method == 3)
258     {
259         auto dsf = new MyDSF();
260         auto flockingBoids = new std::vector<SyncFlockingBoid*>();
261         auto fbm = new FlockingBoidManager(dsf);
262         for (int i = 0; i < numberOfObjects; i++)
263             flockingBoids->push_back(new SyncFlockingBoid(new Vector3D(dsf->window->getSize().x/2, dsf->
264 window->getSize().y/2), 2.0f, 0.05f));
265         for(auto & flockingBoid : *flockingBoids)
266         {
267             flockingBoid->setFillColor(sf::Color::Cyan);
268             dsf->add(flockingBoid);
269             dsf->send(flockingBoid,
270                     dsf->sender,
271                     fbm->create,
272                     new dsf::TaskArgument(std::make_tuple(flockingBoid, flockingBoids)));
273         }
274         dsf->duration = durationPerIterator;
275         dsf->maxNumberOfCores = maxNumberOfThreads;
276         dsf->start();
277         delete dsf;
278         delete fbm;
279     }
280     configure();
281 }

```

