****

**Institute of Technology, Carlow**

**B.Sc. (Honour) in Software Development**

**CW228**

**Research Manual**

**Thread Pool Benchmarking**

**Name: Bowen Nong**

**Login ID: C00163585**

**Supervisor: Joseph Kehoe**

**Date: November 20th 2012**

Contents

[1. Introduction 3](#_Toc341362441)

[2. Thread Pool 3](#_Toc341362442)

[2.1 C++ 11 3](#_Toc341362443)

[2.1.1 Overview Of How Is Works: 4](#_Toc341362444)

[2.1.2 Is It Easy To Use? 6](#_Toc341362445)

[2.1.3 How Flexible Is It 6](#_Toc341362446)

[2.1.4 How Does It Schedule 6](#_Toc341362447)

[2.1.5 Simple Codes 6](#_Toc341362448)

[2.2 QThread 7](#_Toc341362449)

[2.2.1 Overview Of How Is Works: 7](#_Toc341362450)

[2.2.2 Is It Easy To Use? 7](#_Toc341362451)

[2.2.3 How Flexible Is It 7](#_Toc341362452)

[2.2.4 How Does It Schedule 7](#_Toc341362453)

[2.2.5 Simple Codes 7](#_Toc341362454)

[2.3 C# TPL 8](#_Toc341362455)

[2.3.1 Overview Of How Is Works: 8](#_Toc341362456)

[2.3.2 Is It Easy To Use? 8](#_Toc341362457)

[2.3.3 How Flexible Is It 8](#_Toc341362458)

[2.3.4 How Does It Schedule 8](#_Toc341362459)

[2.3.5 Simple Codes 8](#_Toc341362460)

[2.4 Open MP 9](#_Toc341362461)

[2.4.1 Overview Of How Is Works: 9](#_Toc341362462)

[2.4.2 Is It Easy To Use? 9](#_Toc341362463)

[2.4.3 How Flexible Is It 9](#_Toc341362464)

[2.4.4 How Does It Schedule 9](#_Toc341362465)

[2.4.5 Simple Codes 10](#_Toc341362466)

[2.5 TBB 10](#_Toc341362467)

[2.5.1 Overview Of How Is Works: 10](#_Toc341362468)

[2.5.2 Is It Easy To Use? 10](#_Toc341362469)

[2.5.3 How Flexible Is It 10](#_Toc341362470)

[2.5.4 How Does It Schedule 10](#_Toc341362471)

[2.5.5 Simple Codes 11](#_Toc341362472)

[3. The Benchmark Suites: 11](#_Toc341362473)

[3.1 The PARSEC Benchmark Suite: 11](#_Toc341362474)

[3.2 The SPLASH-2 Benchmark Suite: 12](#_Toc341362475)

[3.3 The other Benchmark Suite: 14](#_Toc341362476)

[4. The Scheduling Algorithms: 14](#_Toc341362477)

[4.1 First Come First Served 14](#_Toc341362478)

[4.2 Last Come First Served 15](#_Toc341362479)

[4.3 Round Robin 15](#_Toc341362480)

[5. Games Benchmark 16](#_Toc341362481)

[6. References 23](#_Toc341362482)

Research Manual

# Introduction

The threads were used to increase the speed of running programs. They allow the computer to work seen like doing job on the same time. Actuary, the CPU just allocate the work time for each thread. When the threads are created or killed by system, the CPU will give the sources and time. If each task needs to create a thread for working, it will waste lots of sources and time. However, on the other way, we created a group of threads for jobs. Then we do not kill the thread. When a thread finishes a task, it will “sleep” in the queue. According to the scheduling algorithms of threads, the thread goes out of queue for the new task. It will re-use the threads and save the time and sources. So the “box” which we use to create, kills and manages the group of threads is thread pool.

Thread pool is managed by the system, so that programmers do not need to spend time taking or cart thread management. And they can concentrate on application tasks. The threads are stored by priority.

What are the best scheduling algorithms for thread pool? The different thread pool algorithms will be benchmarked. Then the answers will be unveiled according to this project.

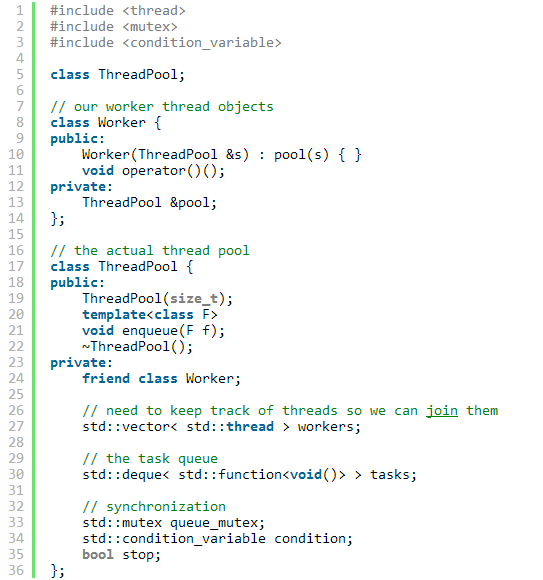
# Thread Pool

## C++ 11

[C++11]

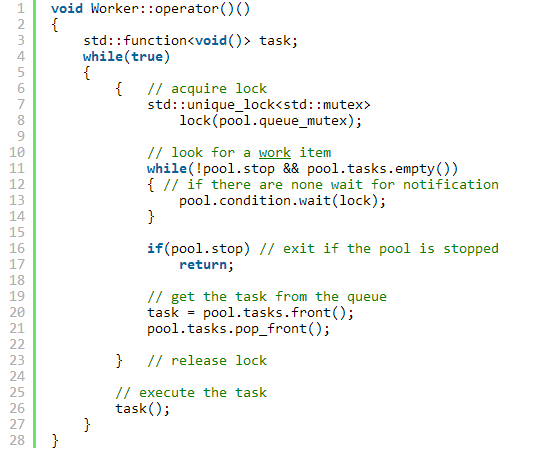
There is Boost Thread. Those aren't thread pool libraries, but we can build a thread pool on top of them. Depends on what problem we try to solve. There is an example thread pool which is created by someone.

### Overview Of How Is Works:



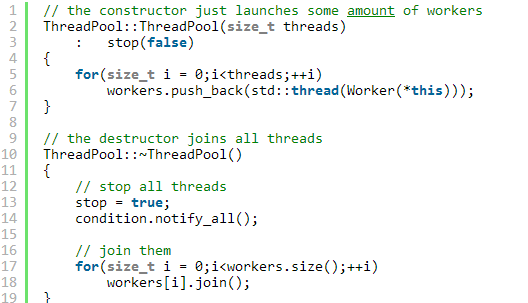
Example 1[C++11]

In the Example 1, the file identified the ThreadPool class and the Worker class. The ThreadPool had two functions for user, which are the constructor and the enqueue. For running the threads, the ThreadPool class allowed the Worker to visit its private variables and functions.



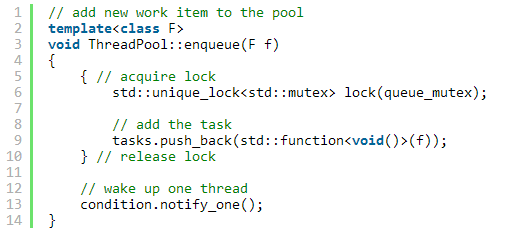
Example 2[C++11]

In the Example 2, the codes show the function operator how to take the threads working for tasks. It first locks the pool, then takes the threads for tasks and operates the task. If the pool is stopped, the operator also stopped.



Example 3[C++11]

In the Example 3, the constructor and destructor mostly remain the same. The destructor now uses notify\_allto make sure any suspended threads see that the stop flag is set.



Example 4[C++11]

In the Example 4, the enqueue function just locks the queue, adds a task to it and wakes up one thread in case any thread was suspended.

### Is It Easy To Use?

The answer is on. Because the user need to create two objects and both four functions which are coming from their objects, when the program use this thread pool for processing.

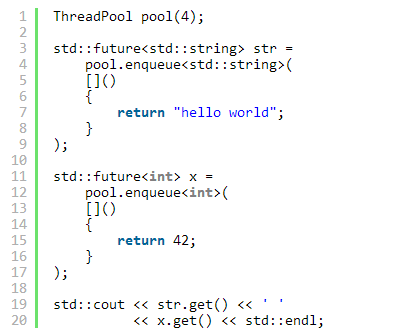
### How Flexible Is It

It allocated the work and store. So user can handle the task flexible.

### How Does It Schedule

It is easy to see that the scheduling algorithm is first-come, first-server. Both the tasks and threads stored in the queue.

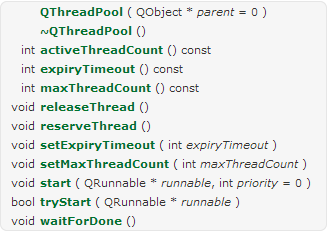
### Simple Codes

[C++11]

## QThread

[QThread]

### Overview Of How Is Works:



Example 5[QThread]

The example 5 shows the public functions of QThreadPool. To use one of the QThreadPool threads, subclass [QRunnable](https://developer.blackberry.com/cascades/reference/qrunnable.html) and implement the run() virtual function. Then create an object of that class and pass it to [QThreadPool::start](https://developer.blackberry.com/cascades/reference/qthreadpool.html" \l "start)().QThreadPool deletes the [QRunnable](https://developer.blackberry.com/cascades/reference/qrunnable.html) automatically by default. Use [QRunnable::setAutoDelete](https://developer.blackberry.com/cascades/reference/qrunnable.html" \l "setAutoDelete)() to change the auto-deletion flag. [QThread]

### Is It Easy To Use?

The answer is yes. Because it offers most of functions for programmer which have difference performance. And the clear name will easy to understand how to use it.

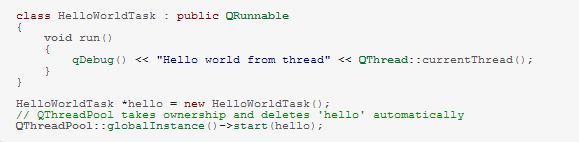
### How Flexible Is It

The QThreadPool allows programmers identify different properties of thread, so that the system can schedule them that the programmers wish.

### How Does It Schedule

Thread scheduling is never random - it is precisely determined by the scheduling algorithms used and the state of all the threads at the time the scheduler runs. In many designs, it can appear pseudo-random.

### 2.2.5 Simple Codes

Example 6[QThread]

The Example 6 shows the simple “Hello World” code of QThreadpool.

## C# TPL

[TPL]

### Overview Of How Is Works:

It first looks at the head of its local queue, then in the global queue, and then in the local queues of other threads. The term task parallelism refers to one or more independent tasks running concurrently. A task represents an asynchronous operation, and in some ways it resembles the creation of a new thread or ThreadPool workitem, but at a higher level of abstraction. A task is represented by the [System.Threading.Tasks.Task](http://msdn.microsoft.com/en-us/library/system.threading.tasks.task.aspx) class.

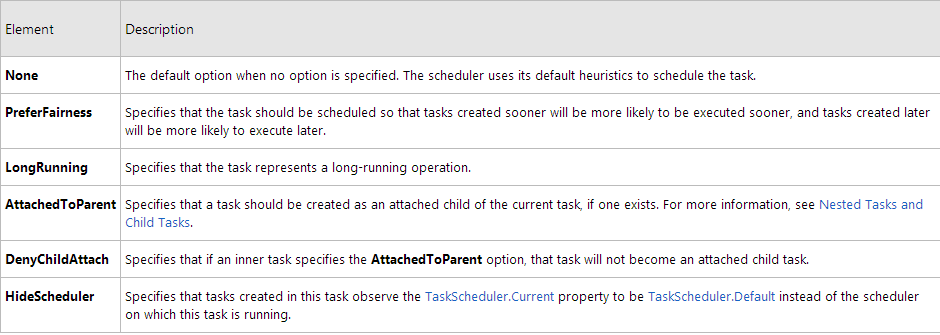
### Is It Easy To Use?

The answer is yes. Because most APIs that create tasks provide overloads that accept a parameter.

### How Flexible Is It

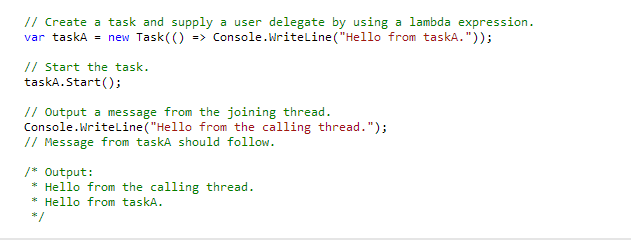
By specifying one of these options, we tell the task scheduler how to schedule the task on the thread pool.

### How Does It Schedule



Example 7: It shows the scheduling which are used by the TPL. [TPL]

### 2.3.5 Simple Codes



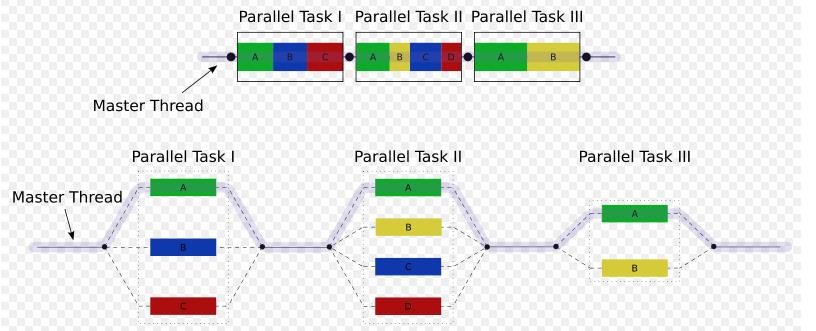
Example 8: it shows the simple “Hello World” code of TPL.

## Open MP

[OpenMP]

### Overview Of How Is Works:

When the OpenMP application begins to work, it should create the main thread. As the program executes, the application can encounter parallel regions in the pool. The thread group will create by the main thread. At the end of a parallel region, the thread teams are parked and the master thread continues execution.

 Example 9: It is the OpenMP model of thread implementation.[OpenMP]

### Is It Easy To Use?

In my point, the answer is yes. The programmers did not need to consider how to run the thread. It takes programmers attention to algorithms. The programmers just use “pragma” apply the mind of which their want system to do.

### How Flexible Is It

The OpenMP give the abstract description of parallel algorithms. The programmers write the standard in source codes, and add the synchronous standard when it necessary. If the compiler did not support the OpenMP, the standards can be ignored.

### How Does It Schedule

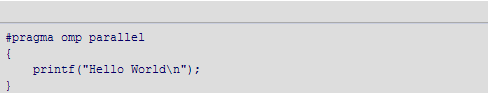
The OpenMP has three kinds of scheduling, which are dynamic, guided and static.

**Static:** all the threads are allocated iterations before they execute the loop iterations. The iterations are divided among threads equally by default. However, specifying an integer for the parameter chunk will allocate chunk number of contiguous iterations to a particular thread.

**Dynamic:** some of the iterations are allocated to a smaller number of threads. Once a particular thread finishes its allocated iteration, it returns to get another one from the iterations that are left. The parameter chunk defines the number of contiguous iterations that are allocated to a thread at a time.

**Guided:** A large chunk of contiguous iterations are allocated to each thread dynamically (as above). The chunk size decreases exponentially with each successive allocation to a minimum size specified in the parameter chunk.

### 2.4.5 Simple Codes



On a two-processor system you may expect output like this:



Example 10: it shows the simple “Hello World” code of OpenMP. [OpenMP]

## TBB

[TBB]

### Overview Of How Is Works:

The TBB is based on the concept of tasks. We define our own tasks which are derived from tbb::task, declared in tbb/task.h. The users must override the pure virtual method task\* task::execute() in their code.

### Is It Easy To Use?

Yes, it is. The TBB lets we easily write parallel C++ programs that take full advantage of multicore performance.

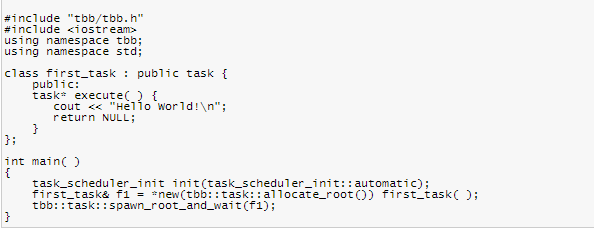
### How Flexible Is It

TBB can run on Windows, Linux, and UNIX, Intel, Microsoft and GNU tools, which covers the vast majority of field of system.

### How Does It Schedule

The TBB task scheduler's fundamental strategy is "breadth-first theft and depth-first work." The breadth-first theft rule raises parallelism sufficiently to keep threads busy. The depth-first work rule keeps each thread operating efficiently once it has sufficient work to do.

### 2.5.5 Simple Codes



Example 11: it shows the simple “Hello World” code of TBB.

# The Benchmark Suites:

## The PARSEC Benchmark Suite:

[PARSEC]

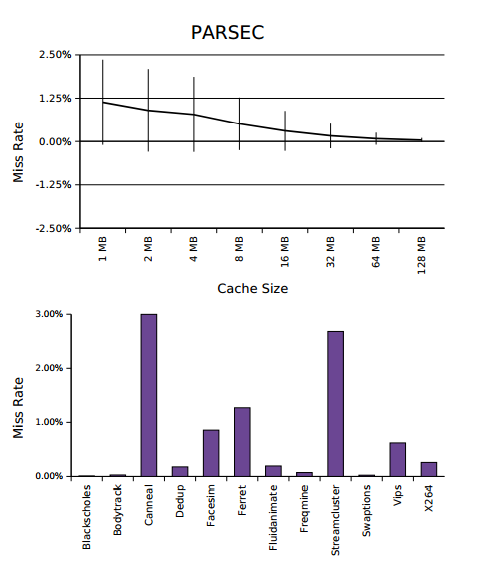
**What is the benchmark?**

The PARSEC, Princeton Application Repository for Shared-Memory Computers, is a benchmark suite include of multithreaded programs. The suite is used to emerge workloads and is designed to be representative of next-generation shared-memory programs for chip-multiprocessors.

The suite is available as open source free of charge for programmers. It includes emerging applications in recognition, mining and synthesis as well as systems applications which mimic large-scale multi-threaded commercial programs. [PARSEC]

**What type of code are they?**

* **bodytrack:**  The bodytrack computer vision application is an Intel RMP workload. It is tracks a 3D pose of a markerless human body with multiple cameras through an image sequence.
* **Blackscholes:** It calculates the prices for a portfolio of European options analytically with the Black-Scholes partial differential equation.
* **facesim:** It takes a model of a human face and a time sequence of muscle activations. Then computing a visually realistic animation of the modeled face by simulating the underlying physics.
* **ferret:** This application is based on the Ferret toolkit. It is used for content-based similarity search of feature-rich data. Such as audio, images, video, 3D shapes etc.
* **freqmine:** It is an array-based version of the Frequent Pattern-growth method for Frequent Item set Mining.
* **vips:** It is originally developed through several projects funded by European Union grants. And based on the VASARI Image Processing System.
* **x264:** It is an H.264/AVC(Advanced Video Coding) video encoder. It is randed 2nd best codes for its high encoding quality.
* **canneal:** It uses cache-aware simulated annealing to minimize the routing cost of a chip design.
* **Streamcluster:** For a stream of input points, it finds a predetermined number of medians so that each point is assigned to its nearest center.



Example 12: It is an example benchmark of PARSEC. [PARSEC]

## The SPLASH-2 Benchmark Suite:

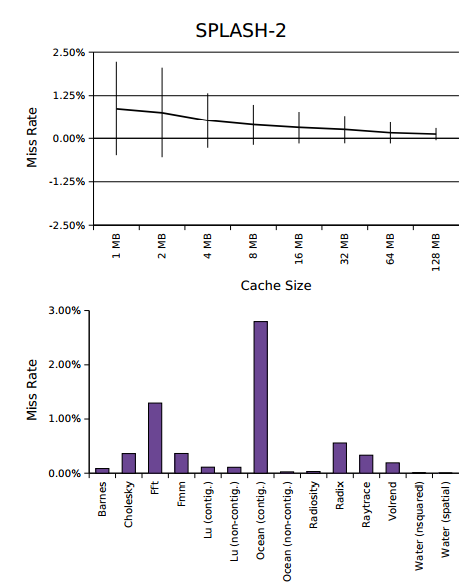
[SPLASH-2]

**What is the benchmark?**

SPLASH-2 is a test of CPU performance benchmark set. The SPLASH-2 suite of parallel applications has recently been released to facilitate the study of centralized and distributed shared-address-space multiprocessors. This work has started as a port of the original benchmarks suite to the [Cyclops-64](http://www.capsl.udel.edu/~venetis/Cyclops-64.html) many-core-on-a-chip architecture and, previously, to the [NANOS execution environment](http://recerca.ac.upc.edu/nanos). Over time, we made all necessary changes to make the suite compile and execute correctly on these platforms and finally decided to release the modified version. [SPLASH-2]

**What type of code are they?**

* **Barnes:** The Barnes application simulates the interaction of a system of bodies in three dimensions over a number of time-steps, using the Barnes-Hut hierarchical N-body method.
* **Cholesky:** The blocked sparse Cholesky factorization kernel factors a sparse matrix into the product of a lower triangular matrix and its transpose.
* **FFT:** The FFT kernel is a complex 1-D version of the radix, which is optimized to minimize interprocessor communication.
* **FMM:** Like Barnes, the FMM application also simulates a system of bodies over a number of timesteps.
* **LU:** The LU kernel factors a dense matrix into the product of a lower triangular and an upper triangular matrix
* **Ocean:** The Ocean application studies large-scale ocean movements based on eddy and boundary currents, and is an improved version of The Ocean program in SPLASH.
* **Radiosity:** This application computes the equilibrium distribution of light in a scene using the iterative hierarchical diffuse radiosity method.
* **Radix:** The integer radix sort kernel is based on the method described.
* **Raytrace:** This application renders a three-dimensional scene using ray tracing.
* **Volrend:** This application renders a three-dimensional volume using a ray casting technique.
* **Water-Nsquared:** This application is an improved version of the Water program in SPLASH.
* **Water-Spatial:** This application solves the same problem as Water Nsquared, but uses a more efficient algorithm.



Example 13: it is an example benchmark of SPLASH-2.

## The other Benchmark Suite:

For benchmark different workload, there are lots of apocopate benchmark suites. They are usually designed to study a specific program area and are thus limited to a single application domain. There are some examples of types of benchmark suites which are ALPBench[ALPBench], BioParallel[SPLASH-2], MediaBench[MediaBench], MineBench[MineBench] and PhysicsBench[SPLASH-2].

# The Scheduling Algorithms:

## First Come First Served

The First-Come, First Served is a scheduling algorithm for CPU scheduling. It makes the tasks input a queue, and the CPU will handle the task which in the head of queue. [Scheduling]

* Since context switches only occur upon tasks termination, and no reorganization of the tasks queue is required, scheduling overhead is minimal.
* Throughput can be low, since long tasks can hog the CPU
* Turnaround time, waiting time and response time can be high for the same reasons above
* No prioritization occurs, thus this system has trouble meeting tasks deadlines.
* The lack of prioritization means that as long as every task eventually completes, there is no starvation. In an environment where some tasks might not complete, there can be starvation.
* It is based on Queuing

## Last Come First Served

The Last-Come, First Served is a scheduling algorithm for CPU scheduling. It makes the tasks input a stack, and the CPU will handle the task which in the tail of stack. [Scheduling]

* Since context switches only occur upon tasks termination, and no reorganization of the tasks queue is required, scheduling overhead is minimal.
* Throughput can be low, since long tasks can hog the CPU
* Turnaround time, waiting time and response time can be high for the same reasons above
* No prioritization occurs, thus this system has trouble meeting tasks deadlines.
* The lack of prioritization means that as long as every task eventually completes, there is no starvation. In an environment where some tasks might not complete, there can be starvation.
* It is based on Stacking

## Round Robin

Each tasks gets a small unit of CPU time (*time quantum*), usually 10-100 milliseconds. After this time has elapsed, the tasks is preempted and added to the end of the ready queue. If there are *n* tasks in the ready queue and the time quantum is *q*, then each task gets 1/*n* of the CPU time in chunks of at most *q* time units at once. No tasks waits more than (*n*-1)*q* time units. [Scheduling]

* RR scheduling involves extensive overhead, especially with a small time unit.
* Balanced throughput between FCFS and SJF, shorter jobs are completed faster than in FCFS and longer processes are completed faster than in SJF.
* Poor average response time, waiting time is dependent on number of processes, and not average process length.
* Because of high waiting times, deadlines are rarely met in a pure RR system.
* Starvation can never occur, since no priority is given. Order of time unit allocation is based upon process arrival time, similar to FCFS.

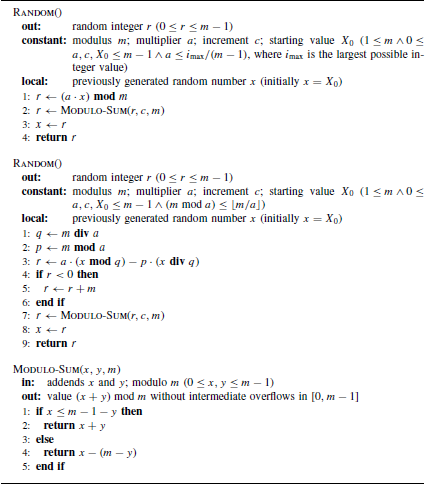
# Games Benchmark

**What do game benchmarks measure?**

A game benchmark typically renders in real time a recording from a game. This produces a repeatable measurement of how fast the graphics of that game runs on the tested system. Since it uses a recording of the animation, a game benchmark seldom measures in-game workloads, like physics, artificial intelligence and user input.

What are algorithms used to develop benchmark program in games? We can follow those:

**Random Numbers:** The random numbers used to generate game outcomes. The outcomes are generated in a highly secure device and cannot be used to determine the correct choice of player selection or influence the game outcome.

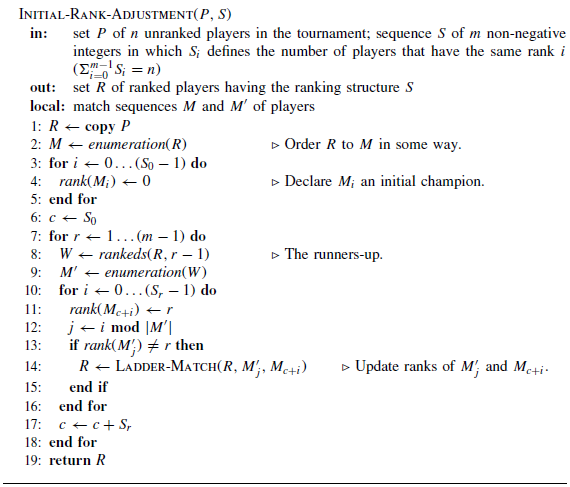


Example 14: They are two algorithm examples of Random Numbers, and describe implementation variants of the linear congenital method defined. [GameAlgorithm]

Description:

The first one can be used when a(m − 1) does not exceed the largest integer that can be represented by the machine word. For example, if m is a one-word integer, the product a(m − 1) must be evaluated within a two-word integer. The second variant can be applied when (m mod a) ≤ m/a. The idea is to express the modulus in the form m = aq + p to guarantee that the intermediate evaluations always stay within the interval (−m, m).

**Tournaments:** There are different tournaments and leagues for the programming games where the characters can compete with each other.

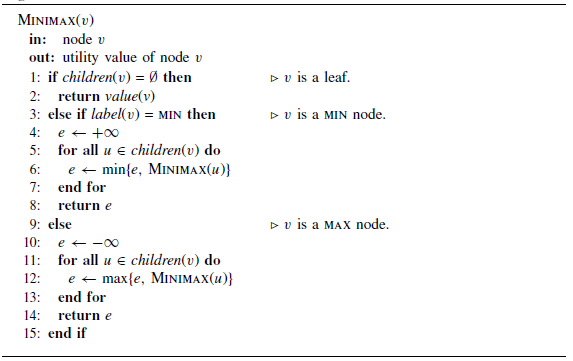


Example 15: It is used to construct initial ranking in rank adjustment tournaments. [GameAlgorithm]

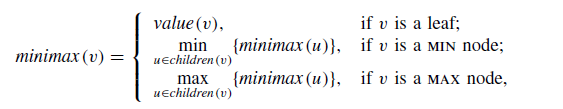
Description:

The ranking structure *S* has the size *m* = |*S*|, which defines the number of different ranks, 0*,* 1*, . . .,m* − 1. Value *Si* indicates how many players have the same rank *i* in the tournament. In other words, in a proper ranking *Si* = |*rankeds(i)*|.

**Game Tree:**  A game tree is a directed graph whose nodes are positions in a game and whose edges are moves.



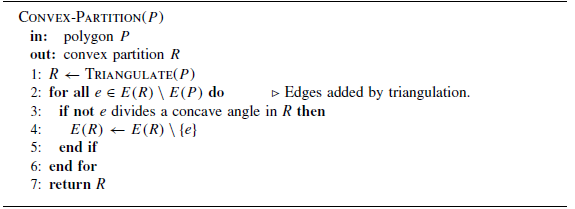
Example 16: It is an Algorithm example of Minimax. [GameAlgorithm]



Example 16.1

The minimax value for a node *v* can be defined with a simple recursion where *children(v)* gives the set of successors of node *v*. Algorithm 4.1 implements this recurrence by determining backed-up values for the internal nodes after the leaves have been evaluated. The Example 16 algorithm runs the equation in Example 16.1.

**Path Finding:** Path Finding in the context of video games concerns the way in which a moving entity finds a path.

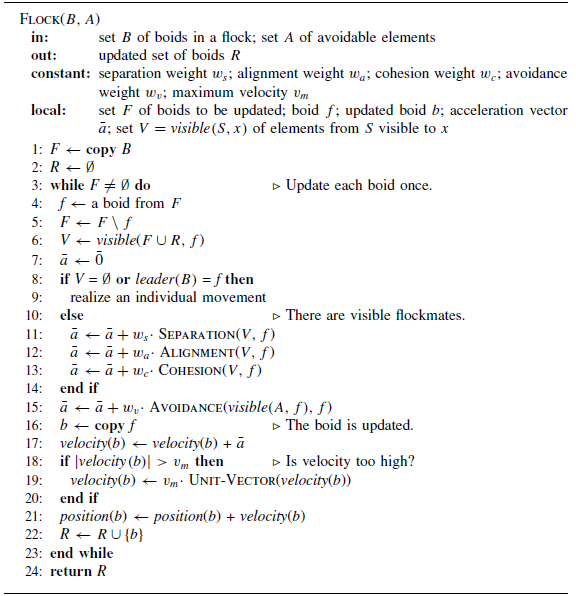


Example 17: It is an Algorithm example of Path Finding. It is Hertel–Mehlhorn method for convex partition. [GameAlgorithm]

Description:

Hertel– Mehlhorn heuristic finds a convex partition in the time *O(n* + *r* log *r)*, and the resulting partition has at most four times the number of polygons of the optimum solution (Hertel and Mehlhorn 1985). The method first triangulates the original polygon.

**Decision-making:** The Decision-making is the set of problems which are strategic, tactical and operational.



Example 18: It is an Algorithm example of decision-making. It is Flocking algorithm. [GameAlgorithm]

Description:

There are four behavior rules in this algorithm. They are following:

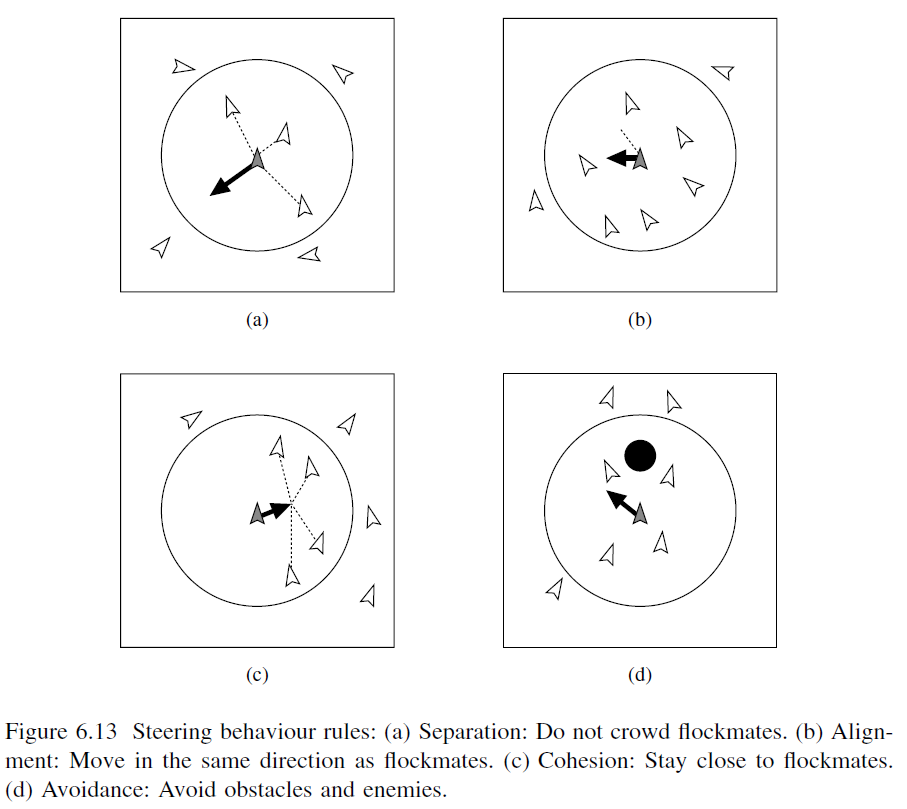
(i) *Separation*: Steer to avoid crowding local flockmates. A boid should maintain a certain

distance from the nearby boids to avoid collisions with them.

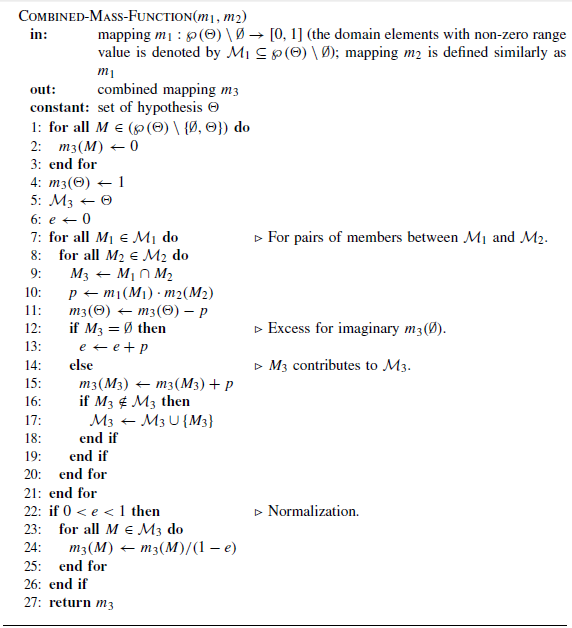
(ii) *Alignment*: Steer towards the average heading of local flockmates. A boid should move in the same direction as the nearby boids and match its velocity accordingly.

(iii) *Cohesion*: Steer to move towards the average position of local flockmates. A boid should stay close to the nearby flockmates.

(iv) *Avoidance*: Steer to avoid running into local obstacles or enemies. A boid should escape dangers when they occur.



**Modelling Uncertainty:** The Decision-making is based on the model of world. So the game is often subject to uncertainties.



Example 19: It is an Algorithm example of Modelling Uncertainty. It is combining two mass function algorithms. [GameAlgorithm]

Description:

Mapping *m*1 which is *℘(\_)* \ ∅→[0*,* 1] (the domain elements with non-zero range value is denoted by *M*1 ⊆ *℘(\_)* \ ∅). Then Mapping *m*2 is defined similarly as *m*1. Out put the combined mapping m3.

# References

[C++11] <http://progsch.net/wordpress/?p=81>  
[Assess 24-10-2012]  
  
[QThread]<http://harmattan-dev.nokia.com/docs/library/html/qt4/qthreadpool.html>  
[Assess 24-10-2012]  
  
[TPL] <http://msdn.microsoft.com/en-us/library/dd537609.aspx#Y100>  
[Assess 24-10-2012]  
  
[OpenMP] <http://msdn.microsoft.com/en-us/magazine/cc163717.aspx>  
[Assess 24-10-2012]  
  
[TBB] <http://threadingbuildingblocks.org/>  
[Assess 24-10-2012]

[Scheduling] http://en.wikipedia.org/wiki/Scheduling\_(computing )  
[Assess 07-11-2012]

[PARSEC] http://parsec.cs.princeton.edu/  
[Assess 11-11-2012]

[SPLASH-2]Christian Bienia, Sanjeev Kumar, Kai Li, “PARSEC vs. SPLASH-2: A Quantitative Comparison of Two Multithreaded Benchmark Suites on Chip-Multiprocessors”, Department of Computer Science, Princeton University, 2008.

[GameAlgorithm] Jouni Smed, Harri Hakonen, “Algorithms and Networking for Computer Games”, *University of Turku, Finland, 2006*

[ALPBench]Man-Lap Li,Ruchira Sasanka,Sarita V. Adve, "The ALPBench Benchmark Suite for Multimedia Applications",Department of Computer Science,University of Illinois at Urbana-Champaign,2005

[MediaBench]Greg Stitt, Frank Vahid, "Binary-Level Hardware/Software Partitioning of MediaBench, NetBench, and EEMBC Benchmarks",Department of Computer Science and Engineering,University of California, Riverside,2003

[MineBench] J Pisharath, Y Liu, W Liao, A Choudhary, "NU-MineBench 2.0",Center for Ultra-Scale Computing and Information Security,Northwestern University,2005