

# ELORank

P. Baillehache

November 16, 2018

## Contents

<b>1</b>	<b>Definitions</b>	<b>2</b>
1.1	ELO rank . . . . .	2
1.2	Soft ELO rank . . . . .	2
1.3	Milestone . . . . .	2
<b>2</b>	<b>Interface</b>	<b>3</b>
<b>3</b>	<b>Code</b>	<b>5</b>
3.1	elorkank.c . . . . .	5
3.2	elorkank-inline.c . . . . .	13
<b>4</b>	<b>Makefile</b>	<b>14</b>
<b>5</b>	<b>Unit tests</b>	<b>15</b>
<b>6</b>	<b>Unit tests output</b>	<b>18</b>
<b>7</b>	<b>ELORank.txt</b>	<b>18</b>

## Introduction

ELORank is a C library providing structures and functions implementing the ELO ranking system in a version supporting several players per run with eventual ties.

It uses the PBErr, PBMath and GSet library.

# 1 Definitions

## 1.1 ELO rank

The ELO rank is calculated incrementally by updating the current ELO rank of each entity according to their result in an evaluation process independent from the ELO ranking system. Given a result of this evaluation process, each pair of winner/looser in this result is updated as follow:

$$\begin{cases} E'_w = E_w + K * \left( 1.0 - \frac{1.0}{1.0 + 10.0^{\frac{E_l - E_w}{400.0}}} \right) \\ E'_l = E_l - K * \left( \frac{1.0}{1.0 + 10.0^{\frac{E_w - E_l}{400.0}}} \right) \end{cases} \quad (1)$$

where  $K = 8.0$  and,  $E_w$  and  $E_l$  are respectively the current ELO of the winner and the current ELO of the loser and,  $E'_w$  and  $E'_l$  are respectively the new ELO of the winner and the new ELO of the loser.

Tie between two entities results in no changes in their respective ELO rank.

## 1.2 Soft ELO rank

The soft ELO rank is the sliding average of the ELO rank over 100 evaluations:

$$softELO(N) = \begin{cases} ELO(0)|_{N=0} \\ ((N-1)softELO(N-1) + ELO(N))/N|_{N < 100} \\ (0.99 * (N-1)softELO(N-1) + ELO(N))/100|_{N \geq 100} \end{cases} \quad (2)$$

where  $N \geq 0$  is the number of evaluation.

## 1.3 Milestone

A milestone is an entity whose ELO rank is kept unchanged. Any other calculation is performed as usual, but the value of the ELO of this entity isn't updated. A milestone is useful when evaluating a pool of variable entities, for example during the training of a genetic algorithm where the non-elite entities are replaced at each step of the genetic algorithm. By setting the milestone to some cleverly selected entities, one can avoid the "relative" effect of the ELO algorithm and keep a ranking consistent even with respect

of entities removed from the ranking. Refer to the Oware example in the MiniFrame repository for an illustration of the use of the milestone property.

## 2 Interface

```
// ===== ELORANK.H =====

#ifndef ELORANK_H
#define ELORANK_H

// ===== Include =====

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdbool.h>
#include "pberr.h"
#include "gset.h"
#include "pbmath.h"

// ===== Define =====

#define ELORANK_K 8.0
#define ELORANK_STARTELO 0.0

// ===== Data structure =====

typedef struct ELOEntity {
    // Pointer toward user struct
    void* _data;
    // Number of evaluation
    long _nbRun;
    // Sum of evaluation
    float _sumSoftElo;
    // Flag to memorize if the entity is a milestone
    // (whose elo is blocked)
    bool _isMilestone;
} ELOEntity;

typedef struct ELORank {
    // ELO coefficient
    float _k;
    // Set of ELO entities
    GSet _set;
} ELORank;

// ===== Functions declaration =====

// Create a new ELORank
ELORank* ELORankCreate(void);
/*#if BUILDMODE == 0
ELORank ELORankCreateStatic(void);
#endif*/

// Free memory used by an ELORank
void ELORankFree(ELORank** that);
```

```

// Free memory used by an ELOEntity
void ELOEntityFree(ELOEntity** that);

// Set the K coefficient of 'that' to 'k'
#if BUILDMODE != 0
inline
#endif
void ELORankSetK(ELORank* const that, const float k);

// Get the K coefficient of 'that'
#if BUILDMODE != 0
inline
#endif
float ELORankGetK(const ELORank* const that);

// Add the entity 'data' to 'that'
void ELORankAdd(ELORank* const that, void* const data);

// Remove the entity 'data' from 'that'
void ELORankRemove(ELORank* const that, void* data);

// Get the number of entity in 'that'
#if BUILDMODE != 0
inline
#endif
int ELORankGetNb(const ELORank* const that);

// Update the ranks in 'that' with results 'res' given as a GSet of
// pointers toward entities (_data in GSetElem equals _data in
// ELOEntity) in winning order
// The _sortVal of the GSet represents the score (and so position)
// of the entities for this update (thus, equal _sortVal means tie)
// The set of results must contain at least 2 elements
// Elements in the result set must be in the ELORank
void ELORankUpdate(ELORank* const that, const GSet* const res);

// Get the current rank of the entity 'data' (starts at 0)
int ELORankGetRank(const ELORank* const that, const void* const data);

// Get the current ELO of the entity 'data'
float ELORankGetELO(const ELORank* const that, const void* const data);

// Get the current soft ELO (average of elo over nb of evaluation)
// of the entity 'data'
float ELORankGetSoftELO(const ELORank* const that,
    const void* const data);

// Set the current ELO of the entity 'data' to 'elo'
void ELORankSetELO(const ELORank* const that, const void* const data,
    const float elo);

// Set the milestone flag of the entity 'data' to 'flag'
void ELORankSetIsMilestone(const ELORank* const that,
    const void* const data, const bool flag);

// Reset the milestone flag of all the entities to false
void ELORankResetAllMilestone(const ELORank* const that);

// Reset the current ELO of the entity 'data'
void ELORankResetELO(const ELORank* const that, const void* const data);

// Get the 'rank'-th entity according to current ELO of 'that'

```

```

// (starts at 0)
const ELOEntity* ELORankGetRanked(const ELORank* const that, const int rank);

// ===== Inliner =====

#if BUILDMODE != 0
#include "elorank-inline.c"
#endif

#endif

```

## 3 Code

### 3.1 elorank.c

```

// ===== ELORANK.C =====

// ===== Include =====

#include "elorank.h"
#if BUILDMODE == 0
#include "elorank-inline.c"
#endif

// ===== Functions declaration =====

// Create a new ELOEntity
static ELOEntity* ELOEntityCreate(void* const data);

// Return the GSetElem in 'that'->_set for the entity 'data'
static GSetElem* ELORankGetElem(const ELORank* const that, const void* const data);

// ===== Functions implementation =====

// Create a new ELORank
ELORank* ELORankCreate(void) {
    // Allocate memory
    ELORank* that = PBErrMalloc(ELORankErr, sizeof(ELORank));
    // Set the default coefficient
    that->_k = ELORANK_K;
    // Create the set of entities
    that->_set = GSetCreateStatic();
    // Return the new ELORank
    return that;
}

// Free memory used by an ELORank
void ELORankFree(ELORank** that) {
    // Check the argument
    if (that == NULL || *that == NULL) return;
    // Empty the set of entities
    GSet* set = &((*that)->_set);
    while (GSetNbElem(set) > 0) {
        ELOEntity *ent = GSetPop(set);
        ELOEntityFree(&ent);
    }
    // Free memory
}

```

```

    free(*that);
    // Set the pointer to null
    *that = NULL;
}

// Free memory used by an ELOEntity
void ELOEntityFree(ELOEntity** that) {
    // Check the argument
    if (that == NULL || *that == NULL) return;
    // Free memory
    free(*that);
    // Set the pointer to null
    *that = NULL;
}

// Add the entity 'data' to 'that'
void ELORankAdd(ELORank* const that, void* const data) {
#ifdef BUILDMODE == 0
    // Check arguments
    if (that == NULL) {
        ELORankErr->_type = PBErrTypeNullPointer;
        sprintf(ELORankErr->_msg, "'that' is null");
        PBErrCatch(ELORankErr);
    }
    if (data == NULL) {
        ELORankErr->_type = PBErrTypeNullPointer;
        sprintf(ELORankErr->_msg, "'data' is null");
        PBErrCatch(ELORankErr);
    }
#endif
    // Create a new ELOEntity
    ELOEntity *ent = ELOEntityCreate(data);
    // Add the new entity to the set with a default score
    GSetAddSort(&(that->_set), ent, ELORANK_STARTELO);
}

// Create a new ELOEntity
static ELOEntity* ELOEntityCreate(void* const data) {
#ifdef BUILDMODE == 0
    // Check argument
    if (data == NULL) {
        ELORankErr->_type = PBErrTypeNullPointer;
        sprintf(ELORankErr->_msg, "'data' is null");
        PBErrCatch(ELORankErr);
    }
#endif
    // Allocate memory
    ELOEntity *that = PBErrMalloc(ELORankErr, sizeof(ELOEntity));
    // Set properties
    that->_data = data;
    that->_nbRun = 0;
    that->_sumSoftElo = 0.0;
    that->_isMilestone = false;
    // Return the new ELOEntity
    return that;
}

// Remove the entity 'data' from 'that'
void ELORankRemove(ELORank* const that, void* data) {
#ifdef BUILDMODE == 0
    // Check arguments
    if (that == NULL) {

```

```

        ELORankErr->_type = PBErrTypeNullPointer;
        sprintf(ELORankErr->_msg, "'that' is null");
        PBErrCatch(ELORankErr);
    }
    if (data == NULL) {
        ELORankErr->_type = PBErrTypeNullPointer;
        sprintf(ELORankErr->_msg, "'data' is null");
        PBErrCatch(ELORankErr);
    }
#endif
    // Search the entity
    GSetElem* elem = ELORankGetElem(that, data);
    // If we have found the entity
    if (elem != NULL) {
        // Free the memory
        ELOEntityFree((ELOEntity**>(&(elem->_data)));
        // Remove the element
        GSetRemoveElem(&(that->_set), &elem);
    }
}

// Return the GSetElem in 'that'->_set for the entity 'data'
static GSetElem* ELORankGetElem(const ELORank* const that, const void* const data) {
#if BUILDMODE == 0
    // Check arguments
    if (that == NULL) {
        ELORankErr->_type = PBErrTypeNullPointer;
        sprintf(ELORankErr->_msg, "'that' is null");
        PBErrCatch(ELORankErr);
    }
    if (data == NULL) {
        ELORankErr->_type = PBErrTypeNullPointer;
        sprintf(ELORankErr->_msg, "'data' is null");
        PBErrCatch(ELORankErr);
    }
#endif
    // Search the element
    GSetElem* elem = that->_set._head;
    while (elem != NULL && ((ELOEntity*)(elem->_data))->_data != data)
        elem = elem->_next;
    // Return the element
    return elem;
}

// Update the ranks in 'that' with results 'res' given as a GSet of
// pointers toward entities (_data in GSetElem equals _data in
// ELOEntity) in winning order
// The _sortVal of the GSet represents the score (and so position)
// of the entities for this update (thus, equal _sortVal means tie)
// The set of results must contain at least 2 elements
// Elements in the result set must be in the ELORank
void ELORankUpdate(ELORank* const that, const GSet* const res) {
#if BUILDMODE == 0
    // Check arguments
    if (that == NULL) {
        ELORankErr->_type = PBErrTypeNullPointer;
        sprintf(ELORankErr->_msg, "'that' is null");
        PBErrCatch(ELORankErr);
    }
    if (res == NULL) {
        ELORankErr->_type = PBErrTypeNullPointer;
        sprintf(ELORankErr->_msg, "'res' is null");
    }
#endif
}

```

```

        PBErCatch(ELORankErr);
    }
    if (GSetNbElem(res) < 2) {
        ELORankErr->_type = PBErTypeInvalidArg;
        sprintf(ELORankErr->_msg,
            "Number of elements in result set invalid (%ld>=2)",
            GSetNbElem(res));
        PBErCatch(ELORankErr);
    }
#endif
    // Declare a variable to memorise the delta of elo of each entity
    VecFloat* deltaElo = VecFloatCreate(GSetNbElem(res));
    // Calculate the delta of elo for each pair of entity
    GSetElem* elemA = res->_head;
    int iElem = 0;
    while (elemA != NULL) {
        GSetElem* elemAElo = ELORankGetElem(that, elemA->_data);
    if BUILDMODE == 0
        if (elemAElo == NULL) {
            ELORankErr->_type = PBErTypeNullPointer;
            sprintf(ELORankErr->_msg,
                "Entity in the result set can't be found in the ELORank.");
            PBErCatch(ELORankErr);
        }
    endif
        GSetElem* elemB = res->_head;
        while (elemB != NULL) {
            // Ignore tie and match with itself
            if (ISEQUALF(elemA->_sortVal, elemB->_sortVal) == false) {
                GSetElem* elemBElo = ELORankGetElem(that, elemB->_data);
    if BUILDMODE == 0
                if (elemBElo == NULL) {
                    ELORankErr->_type = PBErTypeNullPointer;
                    sprintf(ELORankErr->_msg,
                        "Entity in the result set can't be found in the ELORank.");
                    PBErCatch(ELORankErr);
                }
            endif
                // If elemA has won
                if (elemA->_sortVal > elemB->_sortVal) {
                    float winnerELO = elemAElo->_sortVal;
                    float loserELO = elemBElo->_sortVal;
                    float a =
                        1.0 / (1.0 + pow(10.0, (loserELO - winnerELO) / 400.0));
                    VecSetAdd(deltaElo, iElem, that->_k * (1.0 - a));
                }
                // Else, if elemA has lost
            } else {
                float winnerELO = elemBElo->_sortVal;
                float loserELO = elemAElo->_sortVal;
                float b =
                    1.0 / (1.0 + pow(10.0, (winnerELO - loserELO) / 400.0));
                VecSetAdd(deltaElo, iElem, -1.0 * that->_k * b);
            }
        }
        elemB = elemB->_next;
    }
    elemA = elemA->_next;
    ++iElem;
}
// Apply the delta of elo and update the number of run
GSetElem* elem = res->_head;
iElem = 0;

```



```

    while (elem != NULL) {
        GSetElem* elemElo = ELORankGetElem(that, elem->_data);
#if BUILDMODE == 0
        if (elemElo == NULL) {
            ELORankErr->_type = PErrTypeNullPointer;
            sprintf(ELORankErr->_msg,
                "Entity in the result set can't be found in the ELORank.");
            PErrCatch(ELORankErr);
        }
#endif
        // If the entity is a milestone, its elo is blocked to its current
        // value
        if (!((ELOEntity*)(elemElo->_data))->_isMilestone))
            elemElo->_sortVal += VecGet(deltaElo, iElem);
        ++((ELOEntity*)(elemElo->_data))->_nbRun;
        if (((ELOEntity*)(elemElo->_data))->_nbRun >= 100) {
            ((ELOEntity*)(elemElo->_data))->_sumSoftElo *= 0.99;
        }
        ((ELOEntity*)(elemElo->_data))->_sumSoftElo += elemElo->_sortVal;
        ++iElem;
        elem = elem->_next;
    }
    // Free memory
    VecFree(&deltaElo);
    // Sort the ELORank
    GSetSort(&(that->_set));
}

// Get the current rank of the entity 'data' (starts at 0)
int ELORankGetRank(const ELORank* const that, const void* const data) {
#if BUILDMODE == 0
    // Check arguments
    if (that == NULL) {
        ELORankErr->_type = PErrTypeNullPointer;
        sprintf(ELORankErr->_msg, "'that' is null");
        PErrCatch(ELORankErr);
    }
    if (data == NULL) {
        ELORankErr->_type = PErrTypeNullPointer;
        sprintf(ELORankErr->_msg, "'data' is null");
        PErrCatch(ELORankErr);
    }
#endif
    // Declare a variable to memorize the rank
    int rank = 0;
    // Search the element
    GSetElem* elem = that->_set._tail;
    while (elem != NULL && ((ELOEntity*)(elem->_data))->_data != data) {
        elem = elem->_prev;
        ++rank;
    }
#if BUILDMODE == 0
    if (elem == NULL) {
        ELORankErr->_type = PErrTypeNullPointer;
        sprintf(ELORankErr->_msg,
            "Entity requested can't be found in the ELORank.");
        PErrCatch(ELORankErr);
    }
#endif
    // Return the element
    return rank;
}

```

```

// Get the current ELO of the entity 'data'
float ELORankGetELO(const ELORank* const that, const void* const data) {
#ifdef BUILDMODE == 0
    // Check arguments
    if (that == NULL) {
        ELORankErr->_type = PBErrTypeNullPointer;
        sprintf(ELORankErr->_msg, "'that' is null");
        PBErrCatch(ELORankErr);
    }
    if (data == NULL) {
        ELORankErr->_type = PBErrTypeNullPointer;
        sprintf(ELORankErr->_msg, "'data' is null");
        PBErrCatch(ELORankErr);
    }
#endif
    // Declare a variable to memorize the ELO
    float elo = ELORANK_STARTELO;
    // Search the element
    GSetElem* elem = that->_set._head;
    while (elem != NULL && ((ELOEntity*)(elem->_data))->_data != data) {
        elem = elem->_next;
    }
    if (elem != NULL) {
        elo = elem->_sortVal;
    }
#ifdef BUILDMODE == 0
    } else {
        ELORankErr->_type = PBErrTypeNullPointer;
        sprintf(ELORankErr->_msg,
            "Entity requested can't be found in the ELORank.");
        PBErrCatch(ELORankErr);
    }
#endif
    // Return the element
    return elo;
}

// Get the current soft ELO (average of elo over nb of evaluation)
// of the entity 'data'
float ELORankGetSoftELO(const ELORank* const that,
    const void* const data) {
#ifdef BUILDMODE == 0
    // Check arguments
    if (that == NULL) {
        ELORankErr->_type = PBErrTypeNullPointer;
        sprintf(ELORankErr->_msg, "'that' is null");
        PBErrCatch(ELORankErr);
    }
    if (data == NULL) {
        ELORankErr->_type = PBErrTypeNullPointer;
        sprintf(ELORankErr->_msg, "'data' is null");
        PBErrCatch(ELORankErr);
    }
#endif
    // Declare a variable to memorize the ELO
    float elo = ELORANK_STARTELO;
    // Search the element
    GSetElem* elem = that->_set._head;
    while (elem != NULL && ((ELOEntity*)(elem->_data))->_data != data) {
        elem = elem->_next;
    }
    if (elem != NULL) {

```

```

        if (((ELOEntity*)(elem->_data))->_nbRun > 0) {
            elo = ((ELOEntity*)(elem->_data))->_sumSoftElo /
                (float)MIN(100, (((ELOEntity*)(elem->_data))->_nbRun));
        }
    }
    #if BUILDMODE == 0
    } else {
        ELORankErr->_type = PBErTypeNullPointer;
        sprintf(ELORankErr->_msg,
            "Entity requested can't be found in the ELORank.");
        PBErCatch(ELORankErr);
    }
    #endif
    // Return the element
    return elo;
}

// Set the milestone flag of the entity 'data' to 'flag'
void ELORankSetIsMilestone(const ELORank* const that,
    const void* const data, const bool flag) {
    #if BUILDMODE == 0
    // Check arguments
    if (that == NULL) {
        ELORankErr->_type = PBErTypeNullPointer;
        sprintf(ELORankErr->_msg, "'that' is null");
        PBErCatch(ELORankErr);
    }
    if (data == NULL) {
        ELORankErr->_type = PBErTypeNullPointer;
        sprintf(ELORankErr->_msg, "'data' is null");
        PBErCatch(ELORankErr);
    }
    #endif
    // Search the element
    GSetElem* elem = that->_set._head;
    while (elem != NULL && ((ELOEntity*)(elem->_data))->_data != data) {
        elem = elem->_next;
    }
    if (elem != NULL) {
        // Set the flag
        ((ELOEntity*)(elem->_data))->_isMilestone = flag;
    }
    #if BUILDMODE == 0
    } else {
        ELORankErr->_type = PBErTypeNullPointer;
        sprintf(ELORankErr->_msg,
            "Entity requested can't be found in the ELORank.");
        PBErCatch(ELORankErr);
    }
    #endif
}

// Reset the milestone flag of all the entities to false
void ELORankResetAllMilestone(const ELORank* const that) {
    #if BUILDMODE == 0
    // Check arguments
    if (that == NULL) {
        ELORankErr->_type = PBErTypeNullPointer;
        sprintf(ELORankErr->_msg, "'that' is null");
        PBErCatch(ELORankErr);
    }
    #endif
    // Search the element
    GSetElem* elem = that->_set._head;

```

```

while (elem != NULL) {
    ((ELOEntity*)(elem->_data))->_isMilestone = false;
    elem = elem->_next;
}
}

// Set the current ELO of the entity 'data' to 'elo'
void ELORankSetELO(const ELORank* const that, const void* const data,
    const float elo) {
#ifdef BUILDMODE == 0
    // Check arguments
    if (that == NULL) {
        ELORankErr->_type = PErrTypeNullPointer;
        sprintf(ELORankErr->_msg, "'that' is null");
        PErrCatch(ELORankErr);
    }
    if (data == NULL) {
        ELORankErr->_type = PErrTypeNullPointer;
        sprintf(ELORankErr->_msg, "'data' is null");
        PErrCatch(ELORankErr);
    }
#endif
    // Search the element
    GSetElem* elem = that->_set._head;
    while (elem != NULL && ((ELOEntity*)(elem->_data))->_data != data) {
        elem = elem->_next;
    }
    if (elem != NULL) {
        // Set the elo
        elem->_sortVal = elo;
#ifdef BUILDMODE == 0
    } else {
        ELORankErr->_type = PErrTypeNullPointer;
        sprintf(ELORankErr->_msg,
            "Entity requested can't be found in the ELORank.");
        PErrCatch(ELORankErr);
    }
#endif
    // Sort the ELORank
    GSetSort((GSet*)&(that->_set));
}

// Reset the current ELO of the entity 'data'
void ELORankResetELO(const ELORank* const that, const void* const data) {
#ifdef BUILDMODE == 0
    // Check arguments
    if (that == NULL) {
        ELORankErr->_type = PErrTypeNullPointer;
        sprintf(ELORankErr->_msg, "'that' is null");
        PErrCatch(ELORankErr);
    }
    if (data == NULL) {
        ELORankErr->_type = PErrTypeNullPointer;
        sprintf(ELORankErr->_msg, "'data' is null");
        PErrCatch(ELORankErr);
    }
#endif
    // Search the element
    GSetElem* elem = that->_set._head;
    while (elem != NULL && ((ELOEntity*)(elem->_data))->_data != data) {
        elem = elem->_next;
    }
}

```

```

    if (elem != NULL) {
        // Reset the elo, nbRun and sumSoftElo
        elem->_sortVal = ELORANK_STARTELO;
        ((ELOEntity*)(elem->_data))->_sumSoftElo = 0.0;
        ((ELOEntity*)(elem->_data))->_nbRun = 0;
    }
    #if BUILDMODE == 0
    } else {
        ELORankErr->_type = PBErrTypeNullPointer;
        sprintf(ELORankErr->_msg,
            "Entity requested can't be found in the ELORank.");
        PBErrCatch(ELORankErr);
    }
    #endif
    // Sort the ELORank
    GSetSort((GSet*)&(that->_set));
}

// Get the 'rank'-th entity according to current ELO of 'that'
// (starts at 0)
const ELOEntity* ELORankGetRanked(const ELORank* const that, const int rank) {
    #if BUILDMODE == 0
    // Check arguments
    if (that == NULL) {
        ELORankErr->_type = PBErrTypeNullPointer;
        sprintf(ELORankErr->_msg, "'that' is null");
        PBErrCatch(ELORankErr);
    }
    if (rank < 0 || rank >= GSetNbElem(&(that->_set))) {
        ELORankErr->_type = PBErrTypeInvalidArg;
        sprintf(ELORankErr->_msg, "'rank' is invalid (0<=%d<=%ld)", rank,
            GSetNbElem(&(that->_set)));
        PBErrCatch(ELORankErr);
    }
    #endif
    GSetElem* elem = that->_set._tail;
    for (int i = rank; i--;)
        elem = elem->_prev;
    return (ELOEntity*)(elem->_data);
}

```

## 3.2 elorank-inline.c

```

// ===== ELORANK-INLINE.C =====

// ===== Functions implementation =====

// Set the K coefficient of 'that' to 'k'
#if BUILDMODE != 0
inline
#endif
void ELORankSetK(ELORank* const that, const float k) {
    #if BUILDMODE == 0
    // Check arguments
    if (that == NULL) {
        ELORankErr->_type = PBErrTypeNullPointer;
        sprintf(ELORankErr->_msg, "'that' is null");
        PBErrCatch(ELORankErr);
    }
    #endif
    that->_k = k;
}

```

```

}

// Get the K coefficient of 'that'
#if BUILDMODE != 0
inline
#endif
float ELORankGetK(const ELORank* const that) {
#if BUILDMODE == 0
    // Check argument
    if (that == NULL) {
        ELORankErr->_type = PErrTypeNullPointer;
        sprintf(ELORankErr->_msg, "'that' is null");
        PErrCatch(ELORankErr);
    }
#endif
    return that->_k;
}

// Get the number of entity in 'that'
#if BUILDMODE != 0
inline
#endif
int ELORankGetNb(const ELORank* const that) {
#if BUILDMODE == 0
    // Check argument
    if (that == NULL) {
        ELORankErr->_type = PErrTypeNullPointer;
        sprintf(ELORankErr->_msg, "'that' is null");
        PErrCatch(ELORankErr);
    }
#endif
    return GSetNbElem(&(amp;that->_set));
}

```

## 4 Makefile

```

# Build mode
# 0: development (max safety, no optimisation)
# 1: release (min safety, optimisation)
# 2: fast and furious (no safety, optimisation)
BUILD_MODE?=0

all: pbmake_wget main

# Automatic installation of the repository PBMake in the parent folder
pbmake_wget:
if [ ! -d ../PBMake ]; then wget https://github.com/BayashiPascal/PBMake/archive/master.zip; unzip master.zip; rm -f

# Makefile definitions
MAKEFILE_INC=../PBMake/Makefile.inc
include $(MAKEFILE_INC)

# Rules to make the executable
repo=elorank
$($(repo)_EXENAME): \
$($(repo)_EXENAME).o \
$($(repo)_EXE_DEP) \
$($(repo)_DEP)

```

```
$(COMPILER) 'echo "$($(repo)_EXE_DEP) $($repo)_EXENAME.o" | tr ' ' '\n' | sort -u' $(LINK_ARG) $($repo)_LINK_ARG)

$($(repo)_EXENAME).o: \
$($(repo)_DIR)/$($(repo)_EXENAME).c \
$($(repo)_INC_H_EXE) \
$($(repo)_EXE_DEP)
$(COMPILER) $(BUILD_ARG) $($repo)_BUILD_ARG 'echo "$($(repo)_INC_DIR)" | tr ' ' '\n' | sort -u' -c $($repo)_DIR)/
```

## 5 Unit tests

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include <sys/time.h>
#include "elorank.h"
#include "pberr.h"
#include "pbmath.h"

#define RANDOMSEED 2

typedef struct Player {
    int _id;
} Player;

void UnitTestCreateFree() {
    ELORank* elo = ELORankCreate();
    if (elo == NULL || elo->k != ELORANK_K) {
        ELORankErr->_type = PErrTypeUnitTestFailed;
        sprintf(ELORankErr->_msg, "ELORankCreate failed");
        PErrCatch(ELORankErr);
    }
    ELORankFree(&elo);
    if (elo != NULL) {
        ELORankErr->_type = PErrTypeUnitTestFailed;
        sprintf(ELORankErr->_msg, "ELORankFree failed");
        PErrCatch(ELORankErr);
    }
    printf("UnitTestCreateFree OK\n");
}

void UnitTestSetGetK() {
    ELORank* elo = ELORankCreate();
    float k = 1.0;
    ELORankSetK(elo, k);
    if (ISEQUALF(elo->k, k) == false) {
        ELORankErr->_type = PErrTypeUnitTestFailed;
        sprintf(ELORankErr->_msg, "ELORankSetK failed");
        PErrCatch(ELORankErr);
    }
    if (ISEQUALF(ELORankGetK(elo), k) == false) {
        ELORankErr->_type = PErrTypeUnitTestFailed;
        sprintf(ELORankErr->_msg, "ELORankGetK failed");
        PErrCatch(ELORankErr);
    }
    ELORankFree(&elo);
}
```

```

    printf("UnitTestSetGetK OK\n");
}

void UnitTestAddRemoveGetNb() {
    ELORank* elo = ELORankCreate();
    Player *playerA = PBErrMalloc(ELORankErr, sizeof(Player));
    Player *playerB = PBErrMalloc(ELORankErr, sizeof(Player));
    ELORankAdd(elo, playerA);
    if (ELORankGetNb(elo) != 1) {
        ELORankErr->_type = PBErrTypeUnitTestFailed;
        sprintf(ELORankErr->_msg, "ELORankAdd failed");
        PBErrCatch(ELORankErr);
    }
    if (((ELOEntity*)(elo->_set._head->_data))->_data != playerA) {
        ELORankErr->_type = PBErrTypeUnitTestFailed;
        sprintf(ELORankErr->_msg, "ELORankAdd failed, _data invalid");
        PBErrCatch(ELORankErr);
    }
    if (((ELOEntity*)(elo->_set._head->_data))->_nbRun != 0) {
        ELORankErr->_type = PBErrTypeUnitTestFailed;
        sprintf(ELORankErr->_msg, "ELORankAdd failed, _nbRun invalid");
        PBErrCatch(ELORankErr);
    }
    if (ISEQUALF(elo->_set._head->_sortVal, ELORANK_STARTELO) == false) {
        ELORankErr->_type = PBErrTypeUnitTestFailed;
        sprintf(ELORankErr->_msg, "ELORankAdd failed, _sortVal invalid");
        PBErrCatch(ELORankErr);
    }
    ELORankAdd(elo, playerB);
    if (ELORankGetNb(elo) != 2) {
        ELORankErr->_type = PBErrTypeUnitTestFailed;
        sprintf(ELORankErr->_msg, "ELORankAdd failed");
        PBErrCatch(ELORankErr);
    }
    if (((ELOEntity*)(elo->_set._head->_next->_data))->_data !=
        playerB) {
        ELORankErr->_type = PBErrTypeUnitTestFailed;
        sprintf(ELORankErr->_msg, "ELORankAdd failed, _data invalid");
        PBErrCatch(ELORankErr);
    }
    ELORankRemove(elo, playerA);
    if (ELORankGetNb(elo) != 1) {
        ELORankErr->_type = PBErrTypeUnitTestFailed;
        sprintf(ELORankErr->_msg, "ELORankRemove failed");
        PBErrCatch(ELORankErr);
    }
    if (((ELOEntity*)(elo->_set._head->_data))->_data != playerB) {
        ELORankErr->_type = PBErrTypeUnitTestFailed;
        sprintf(ELORankErr->_msg, "ELORankRemove failed, _data invalid");
        PBErrCatch(ELORankErr);
    }
    ELORankFree(&elo);
    free(playerA);
    free(playerB);
    printf("UnitTestAddRemoveGetNb OK\n");
}

void UnitTestUpdateGetRankGetElo() {
    srandom(RANDOMSEED);
    ELORank* elo = ELORankCreate();
    Player *players[3] = {NULL};
    GSet res = GSetCreateStatic();

```



```

Gauss gausses[3];
for (int i = 3; i--;) {
    players[i] = PBErrMalloc(ELORankErr, sizeof(Player));
    players[i]->_id = i;
    ELORankAdd(elo, players[i]);
    gausses[i] = GaussCreateStatic(3 - i, 1.0);
}
int nbRun = 100;
FILE* f = fopen("./elorank.txt", "w");
for (int iRun = nbRun; iRun--;) {
    GSetFlush(&res);
    for (int i = 3; i--;)
        GSetAddSort(&res, players[i], GaussRnd(gausses + i));
    ELORankUpdate(elo, &res);
    fprintf(f, "%d %f %f %f %f %f %f\n", (nbRun - iRun),
        ELORankGetELO(elo, players[0]),
        ELORankGetELO(elo, players[1]),
        ELORankGetELO(elo, players[2]),
        ELORankGetSoftELO(elo, players[0]),
        ELORankGetSoftELO(elo, players[1]),
        ELORankGetSoftELO(elo, players[2]));
}
fclose(f);
for (int i = 3; i--;) {
    if (ELORankGetRank(elo, players[i]) != i) {
        ELORankErr->_type = PBErrTypeUnitTestFailed;
        sprintf(ELORankErr->_msg, "ELORankUpdate failed");
        PBErrCatch(ELORankErr);
    }
}
const ELOEntity *winner = ELORankGetRanked(elo, 0);
if (winner->_data != players[0]) {
    ELORankErr->_type = PBErrTypeUnitTestFailed;
    sprintf(ELORankErr->_msg, "ELORankGeRanked failed");
    PBErrCatch(ELORankErr);
}
if (winner->_nbRun != nbRun) {
    ELORankErr->_type = PBErrTypeUnitTestFailed;
    sprintf(ELORankErr->_msg, "nbRun invalid");
    PBErrCatch(ELORankErr);
}
ELORankSetELO(elo, players[0], 10.0);
if (!ISEQUALF(ELORankGetELO(elo, players[0]), 10.0)) {
    ELORankErr->_type = PBErrTypeUnitTestFailed;
    sprintf(ELORankErr->_msg, "ELORankSetELO failed");
    PBErrCatch(ELORankErr);
}
ELORankFree(&elo);
GSetFlush(&res);
for (int i = 3; i--;)
    free(players[i]);
printf("UnitTestUpdateGetRankGetElo OK\n");
}

void UnitTestAll() {
    UnitTestCreateFree();
    UnitTestSetGetK();
    UnitTestAddRemoveGetNb();
    UnitTestUpdateGetRankGetElo();
    printf("UnitTestAll OK\n");
}

```

```

int main() {
    UnitTestAll();
    // Return success code
    return 0;
}

```

## 6 Unit tests output

```

UnitCreateFree OK
UnitSetGetK OK
UnitAddRemoveGetNb OK
UnitUpdateGetRankGetElo OK
UnitAll OK

```

## 7 ELORank.txt

```

1 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
2 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
3 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
4 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
5 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
6 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
7 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
8 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
9 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
10 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
11 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
12 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
13 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
14 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
15 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
16 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
17 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
18 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
19 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
20 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
21 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
22 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
23 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
24 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
25 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
26 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
27 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
28 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
29 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
30 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
31 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
32 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
33 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
34 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
35 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
36 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
37 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
38 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
39 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
40 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000

```

[illegible]

