ELORank

P. Baillehache

November 14, 2018

Contents

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

The ELO rank is calculated incrementally by updating the current ELO rank of each entity according to their result in an evaluation process independant 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 looser and, E'_w and E'_l are respectively the new ELO of the winner and the new ELO of the looser.

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

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 evalutation
```

```
  float _sumSoftElo;
} 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);

// 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) {
#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
  // 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) {
#if 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;
  // Return the new ELOEntity
  return that;
}

// Remove the entity 'data' from 'that'
void ELORankRemove(ELORank* const that, void* 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 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");
    PBErrCatch(ELORankErr);
  }
  if (GSetNbElem(res) < 2) {
    ELORankErr->_type = PBErrTypeInvalidArg;
    sprintf(ELORankErr->_msg,
      "Number of elements in result set invalid (%ld>=2)",
      GSetNbElem(res));
    PBErrCatch(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 = PBErrTypeNullPointer;
      sprintf(ELORankErr->_msg,
        "Entity in the result set can't be found in the ELORank.");
      PBErrCatch(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 = PBErrTypeNullPointer;
          sprintf(ELORankErr->_msg,
            "Entity in the result set can't be found in the ELORank.");
          PBErrCatch(ELORankErr);
        }
#endif
        // If elemA has won
        if (elemA->_sortVal > elemB->_sortVal) {
          float winnerELO = elemAElo->_sortVal;
          float looserELO = elemBElo->_sortVal;
          float a =
            1.0 / (1.0 + pow(10.0, (looserELO - winnerELO) / 400.0));
          VecSetAdd(deltaElo, iElem, that->_k * (1.0 - a));
        // Else, if elemA has lost
        } else {
          float winnerELO = elemBElo->_sortVal;
```

```
          float looserELO = elemAElo->_sortVal;
          float b =
            1.0 / (1.0 + pow(10.0, (winnerELO - looserELO) / 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 = PBErrTypeNullPointer;
      sprintf(ELORankErr->_msg,
        "Entity in the result set can't be found in the ELORank.");
      PBErrCatch(ELORankErr);
    }
#endif
    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 = 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 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 = PBErrTypeNullPointer;
    sprintf(ELORankErr->_msg,
      "Entity requested can't be found in the ELORank.");
    PBErrCatch(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) {
#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
  // 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;
#if 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) {
#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
  // 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 = PBErrTypeNullPointer;
    sprintf(ELORankErr->_msg,
      "Entity requested can't be found in the ELORank.");
    PBErrCatch(ELORankErr);
#endif
  }
  // Return the element
  return elo;
}

// Set the current ELO of the entity 'data' to 'elo'
void ELORankSetELO(const ELORank* const that, const void* const data,
  const float elo) {
#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;
  }
  if (elem != NULL) {
    // Set the elo
    elem->_sortVal = elo;
#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));
}
```

```
// Reset the current ELO of the entity 'data'
void ELORankResetELO(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;
  }
  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 = PBErrTypeNullPointer;
    sprintf(ELORankErr->_msg, "'that' is null");
    PBErrCatch(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 = PBErrTypeNullPointer;
    sprintf(ELORankErr->_msg, "'that' is null");
    PBErrCatch(ELORankErr);
  }
#endif
  return GSetNbElem(&(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 = PBErrTypeUnitTestFailed;
    sprintf(ELORankErr->_msg, "ELORankCreate failed");
    PBErrCatch(ELORankErr);
  }
  ELORankFree(&elo);
  if (elo != NULL) {
    ELORankErr->_type = PBErrTypeUnitTestFailed;
    sprintf(ELORankErr->_msg, "ELORankFree failed");
```

```
    PBErrCatch(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 = PBErrTypeUnitTestFailed;
    sprintf(ELORankErr->_msg, "ELORankSetK failed");
    PBErrCatch(ELORankErr);
  }
  if (ISEQUALF(ELORankGetK(elo), k) == false) {
    ELORankErr->_type = PBErrTypeUnitTestFailed;
    sprintf(ELORankErr->_msg, "ELORankGetK failed");
    PBErrCatch(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

```
UnitTestCreateFree OK
UnitTestSetGetK OK
UnitTestAddRemoveGetNb OK
UnitTestUpdateGetRankGetElo OK
UnitTestAll OK
```

7 ELORank.txt

```
1 8.000000 -8.000000 0.000000 8.000000 -8.000000 0.000000
2 15.723836 -15.723836 -0.000000 11.861917 -11.861917 -0.000000
3 15.181863 -7.181863 -8.000000 12.968566 -10.301899 -2.666667
4 22.658251 -6.934165 -15.724085 15.390987 -9.459966 -5.931021
5 29.878273 -14.695467 -15.182804 18.288445 -10.507066 -7.781378
6 36.852001 -14.190701 -22.661297 21.382370 -11.121005 -10.261365
7 43.589993 -21.704742 -21.885246 24.554888 -12.632967 -11.921919
8 50.101974 -20.963814 -29.138155 27.748274 -13.674323 -14.073948
9 48.398232 -12.250955 -36.147270 30.042714 -13.516171 -16.526540
10 46.752392 -3.834395 -42.917988 31.713681 -12.547993 -19.165685
11 53.164032 -3.704153 -49.459869 33.663713 -11.744007 -21.919701
12 59.366798 -11.578976 -47.787811 35.805636 -11.730255 -24.075378
13 65.365189 -11.188716 -54.176464 38.079449 -11.688599 -26.390846
14 71.173401 -10.813753 -60.359642 40.443303 -11.626110 -28.817189
15 76.800728 -10.453563 -66.347153 42.867131 -11.547940 -31.319185
16 82.256058 -10.107615 -72.148430 45.328938 -11.457920 -33.871014
17 79.547867 -1.775375 -77.772476 47.341815 -10.888359 -36.453452
18 84.931526 -9.717095 -75.214417 49.430132 -10.823289 -38.606839
19 90.144630 -9.398850 -80.745766 51.573001 -10.748318 -40.824678
20 95.206184 -17.093233 -78.112938 53.754663 -11.065564 -42.689090
```

```
21 100.111649 -24.538937 -75.572701 55.962141 -11.707153 -44.254976
22 104.868889 -23.745762 -81.123116 58.185175 -12.254363 -45.930800
23 109.494507 -22.983797 -86.510704 60.416016 -12.720860 -47.695143
24 105.994957 -30.251780 -75.743172 62.315140 -13.451314 -48.863810
25 110.581802 -29.279697 -81.302094 64.245806 -14.084449 -50.161343
26 115.044350 -36.345665 -78.698669 66.199594 -14.940650 -51.258930
27 119.379044 -35.190456 -84.188576 68.169203 -15.690643 -52.478547
28 123.600494 -34.080109 -89.520370 70.148891 -16.347410 -53.801470
29 119.714195 -25.012819 -94.701355 71.858036 -16.646217 -55.211813
30 123.941559 -32.228443 -91.713097 73.594157 -17.165625 -56.428524
31 128.049164 -31.220741 -96.828400 75.350767 -17.619017 -57.731745
32 132.054733 -30.252058 -101.802650 77.122765 -18.013800 -59.108959
33 127.963089 -21.320766 -106.642296 78.663382 -18.114010 -60.549364
34 131.989212 -20.663591 -111.325592 80.231790 -18.188997 -62.042782
35 135.918457 -20.031824 -115.886597 81.822838 -18.241649 -63.581180
36 123.755226 -19.424370 -104.330818 82.987623 -18.274502 -64.713114
37 127.890121 -18.821854 -109.068230 84.201205 -18.289295 -65.911898
38 131.923431 -26.242733 -105.680656 85.457051 -18.498596 -66.958445
39 127.842506 -17.436901 -110.405563 86.543858 -18.471373 -68.072472
40 123.880096 -8.900932 -114.979118 87.477264 -18.232112 -69.245135
41 128.036240 -8.627240 -119.408958 88.466505 -17.997847 -70.468643
42 132.091034 -16.364151 -115.726845 89.505185 -17.958949 -71.546218
43 136.027908 -15.865696 -120.162178 90.587107 -17.910270 -72.676820
44 131.872452 -7.386406 -124.486008 91.525413 -17.671091 -73.854304
45 135.838455 -7.163136 -128.675278 92.510145 -17.437581 -75.072548
46 139.711716 -14.948422 -124.763252 93.536271 -17.383469 -76.152779
47 135.472931 -6.500619 -128.972260 94.428545 -17.151919 -77.276596
48 139.356705 -6.305744 -133.050919 95.364553 -16.925957 -78.438563
49 135.151398 1.881705 -137.033066 96.176529 -16.542127 -79.634367
50 131.069504 -6.174252 -124.895218 96.874385 -16.334769 -80.539585
51 127.058136 2.012465 -129.070572 97.466222 -15.975020 -81.491173
52 131.167801 1.951608 -133.119385 98.114333 -15.630277 -82.484018
53 135.179230 1.893085 -137.072296 98.813670 -15.299648 -83.513985
54 131.096741 9.836794 -140.933502 99.411504 -14.834158 -84.577311
55 135.137115 9.544323 -144.681412 100.061062 -14.390914 -85.670117
56 139.082886 9.262930 -148.345795 100.757882 -13.968523 -86.789324
57 142.938095 8.992129 -151.930206 101.497884 -13.565705 -87.932146
58 138.706543 16.731459 -155.437988 102.139413 -13.043340 -89.096040
59 142.600052 8.246344 -150.846375 102.825187 -12.682498 -90.142653
60 130.373032 16.006891 -146.379913 103.284318 -12.204342 -91.079940
61 134.453491 7.533657 -141.987137 103.795290 -11.880768 -91.914487
62 138.407562 -0.689304 -137.718246 104.353555 -11.700261 -92.653257
63 142.242188 -8.668893 -133.573273 104.954962 -11.652144 -93.302781
64 145.964005 -8.412176 -137.551804 105.595726 -11.601520 -93.994171
65 149.603500 -8.165155 -141.438309 106.272769 -11.548653 -94.724084
66 153.164139 -7.927407 -145.236694 106.983243 -11.493786 -95.489428
67 156.649200 -7.698526 -148.950638 107.724529 -11.437140 -96.287357
68 160.061768 -7.478126 -152.583603 108.494198 -11.378920 -97.115242
69 163.404770 -7.265838 -156.138885 109.290004 -11.319309 -97.970654
70 166.680939 -7.061309 -159.619583 110.109877 -11.258480 -98.851353
71 169.892883 -6.864205 -163.028641 110.951894 -11.196589 -99.755261
72 173.043076 -6.674206 -166.368820 111.814270 -11.133778 -100.680447
73 176.133820 -6.491009 -169.642776 112.695366 -11.070179 -101.625134
74 179.167343 -6.314322 -172.852982 113.593631 -11.005911 -102.587673
75 182.145721 -6.143868 -176.001816 114.507656 -10.941083 -103.566530
76 185.070938 -5.979387 -179.091507 115.436125 -10.875797 -104.560277
77 187.944855 -5.820625 -182.124191 116.377803 -10.810146 -105.567598
78 182.769272 -13.667344 -169.101898 117.228979 -10.846777 -106.382136
79 177.653793 -5.298442 -172.355316 117.993844 -10.776545 -107.217242
80 180.664001 -5.155023 -175.508942 118.777222 -10.706276 -108.070886
81 183.619919 -5.016639 -178.603241 119.577751 -10.636034 -108.941659
82 186.523483 -4.883080 -181.640366 120.394162 -10.565876 -109.828232
```

```
83 189.376526 -4.754143 -184.622345 121.225280 -10.495855 -110.729363
84 192.180801 -4.629635 -187.551117 122.069987 -10.426019 -111.643903
85 194.937943 -4.509374 -190.428528 122.927252 -10.356412 -112.570784
86 189.649536 3.606816 -193.256317 123.703091 -10.194049 -113.508982
87 184.486328 11.513550 -195.999832 124.401749 -9.944536 -114.457155
88 187.449646 11.214846 -198.664444 125.118197 -9.704088 -115.414051
89 190.360336 10.926239 -201.286530 125.851255 -9.472287 -116.378906
90 193.220200 10.647317 -203.867477 126.599794 -9.248736 -117.350998
91 196.030945 2.377687 -198.408585 127.362777 -9.120973 -118.241737
92 198.755539 2.317230 -201.072723 128.138789 -8.996645 -119.142069
93 193.435593 10.258780 -203.694321 128.840904 -8.789597 -120.051233
94 196.240829 9.999002 -206.239792 129.557929 -8.589718 -120.968137
95 190.998749 17.747816 -208.746536 130.204677 -8.312481 -121.892116
96 193.882751 17.300257 -211.182983 130.867991 -8.045682 -122.822225
97 196.716660 16.867434 -213.584076 131.546845 -7.788846 -123.757913
98 199.502136 8.448761 -207.950882 132.240264 -7.623156 -124.617028
99 202.199768 8.238097 -210.437851 132.946930 -7.462942 -125.483902
100 204.853592 8.034303 -212.887878 132.349824 -7.234086 -125.115645
```