Summarized API for playing Crematoria

This short document briefly presents the main types, classes and methods that you may need to program your player.

```
// Enum to encode directions.
enum Dir {
  Bottom, BR, Right, RT, Top, TL, Left, LB, Up, Down, None,
  DirSize
};

// Defines the type of a cell.
enum CellType {
  Outside, Cave, Rock, Elevator,
  CellTypeSize
};

// Defines the type of a unit.
enum UnitType {
  Pioneer, Furyan, Necromonger, Hellhound,
  UnitTypeSize
};

// Simple struct to handle positions.
struct Pos {
  int i, j, k;
};

Pos::Pos (int i, int j, int k);
// Example: Pos p(3, 6, 0);

// Print operator.
ostream& operator<< (ostream& os, const Pos& p);
// Example: cerr << p << endl;

bool operator== (const Pos& a, const Pos& b);
// Example: if (p == Pos(3, 2, 1)) ...

bool operator!= (const Pos& a, const Pos& b);
// Example: if (p != Pos(3, 2, 1)) ...
```

```
    // Compares using lexicographical order (first by i, then by j, then by k).
    // If needed, you can sort vectors of positions or build sets of positions.
    bool operator< (const Pos& a, const Pos& b);
    // Example: if (p < Pos(3, 2, 1)) ...

// The following four methods are circular w.r.t. the second dimension.

    Pos& operator+= (Dir d);
    // Example: p += Left; // If p was (6, 0, 1), it will be (6, 79, 1).

    Pos operator+ (Dir d);
    // Example: Pos p2 = p + Right;

    Pos& operator+= (Pos p);
    // Example: p += Pos(3, 2, 1); // If p was (0, 78, 0), it will be (3, 0, 1).

    Pos operator+ (Pos p);
    // Example: p2 = p + Pos(3, 2, 1);

    // Returns whether (i, j, k) is a position inside the board.
    bool pos_ok (int i, int j, int k);
    // Example: if (pos_ok(i + 1, j - 1, 0)) ...

    // Returns whether p is a position inside the board.
    bool pos_ok (Pos p);
    // Example: if (pos_ok(p1 + Bottom)) ...

    // Returns whether the position p is currently under the sun.
    bool daylight (Pos p);

    // Describes a cell in the board.
    struct Cell {
      CellType type; // The kind of cell.
      int owner;     // For caves. The player that owns it, otherwise -1.
      int id;        // The id of a unit if present, otherwise -1.
      bool gem;      // For outside cells, if it has a gem or not.
    };

    // Returns a copy of the cell at p.
    Cell cell (Pos p);
    // Example: Cell c2 = cell(p);

    // Returns a copy of the cell at (i, j, k).
    Cell cell (int i, int j, int k);
    // Example: Cell c3 = cell(3, 6, 1);
```

```
// Describes a unit on the board and its properties.
struct Unit {
  UnitType type; // The kind of unit.
  int id;        // The id for this unit (new Necromongers may repeat old
ids).
  int player;    // The player that owns this unit.
  int health;    // For a Hellhound, 0. For the rest, the current health.
  int turns;     // For a Necromonger, the remaining turns until landing
(0 if already landed).
  Pos pos;       // The position inside the board.
};

// Returns the information of the unit with identifier id.
Unit unit (int id);
// Example: Unit u2 = unit(23);

// Print operator.
ostream& operator<< (ostream& os, const Unit& u);
// Example: cerr << u << endl;

// Identifier of your player, between 0 and 3.
int me ();

// Returns the identifiers of all the Furyans of a player.
vector<int> furyans (int player);
// Example: vector<int> F = furyans(3);

// Returns the identifiers of all the pioneers of a player.
vector<int> pioneers (int player);
// Example: vector<int> P = pioneers(0);

// Returns the identifiers of all the alive Necromongers, even those currently
descending.
vector<int> necromongers ();
// Example: vector<int> N = necromongers();

// Returns the identifiers of all the Hellhounds.
vector<int> hellhounds ();
// Example: vector<int> H = hellhounds();

// Returns the current round.
int round ();

// Returns the current number of cells owned by a player.
int nb_cells (int player);

// Returns the number of gems already accumulated by a player.
int nb_gems (int player);
```

```
// Returns the percentage of cpu time used up to the last round by a player.
// It is in the range [0..1], or -1 if this player is dead.
// Note that this method only works when executed in the judge.
double status (int player);

// Returns a random integer in [l..u]. u - l + 1 must be between 1 and 10^6.
int random (int l, int u);
// Example: if (random(0, 4) < 2) whatever();
// This code executes whatever() with probability 2/5.

// Returns a random permutation of [0..n-1]. n must be between 0 and 10^6.
vector<int> random_permutation (int n);

// A movement is defined by a unit identifier and a direction.
void command (int id, Dir dir);
// Example: command(23, Bottom);
```