Summarized API for playing Dominator

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,
  None,
  DirSize
};

// Defines if a cell is empty or has a wall.
enum CellType {
  Empty, Wall,
  CellTypeSize
};

// Defines the type of a unit.
enum UnitType {
  Farmer, Knight, Witch,
  UnitTypeSize
};

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

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

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)) ...

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

```
// Compares using lexicographical order (first by i, then by j).
// 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)) ...

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

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

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

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

// Describes a cell in the board, and its contents.
struct Cell {
  CellType type; // The kind of cell.
  int owner;     // The player that last conquered this cell, or -1.
  int id;        // The id of a unit if present, or -1.
  bool haunted;  // Tells if the cell is threatened by a witch.
};

Cell::Cell(CellType type, int owner, int id, bool haunted);
// Exampe: Cell c(Empty, 2, 23, false);
// A cell with no wall, owned by player 2, with unit 23 on it, and not haunted.

// Describes a unit on the board and its properties.
struct Unit {
  UnitType type; // The kind of unit.
  int id;        // The unique id for this unit during the game.
  int player;    // The player that owns this unit.
  int health;    // The current health of the unit, if it is not a witch.
  Pos pos;       // The position inside the board.
  bool active;   // For witches.
};

Unit::Unit(UnitType type, int id, int player, int health, Pos pos, bool a);
// Example: Unit u(Farmer, 23, 2, 30, Pos(3, 6), false);

// 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).
Cell cell(int i, int j);
// Example: Cell c3 = cell(3, 6);
```

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

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

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

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

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

// Returns the current amount of land owned by a player.
int land (int player);

// Returns the total score of a player.
int total_score (int player);

// Returns the percentage of cpu time used in 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);
```