

Summarized API for playing Money Heist

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

Positions and directions.

```
// Enum to encode directions.  
enum Dir {  
    Down, Right, Up, Left  
};  
  
// Simple struct to handle positions.  
struct Pos {  
    int i, j;  
};  
  
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);
```

```

// Returns whether (i, j) is a position inside the board.
bool pos_ok (int i , int j );

// Returns whether p is a position inside the board.
bool pos_ok (Pos p );

State of the game.

// Returns whether pl is a valid player identifier.
bool player_ok (int pl) const;

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

// Defines kinds of cells.
enum CellType {
    Corridor,
    Wall
};

// Describes a cell on the board, and its contents.
struct Cell {
    CellType      type; // The kind of cell (Corridor or Wall).
    int          id; // The id of a unit if present, or -1 otherwise.
    bool         health_kit ; // Whether it contains a health kit.
    bool         poison_kit ; // Whether it contains a poison kit.
    int          box; // The id of a unit box present, or -1 if none
                      // If ≠ -1, this cell is the door of a safety box
    bool         gold; // Whether it contains gold
                      // (i.e. the gold inside a safety box)
    int          money; // 0 if no money, > 0 contains amount of money.
};

// Defines the type of the unit.
enum UnitType {
    Soldier ,
    Professor
};

```

```

// Describes a unit on the board and its properties.
struct Unit {
    UnitType type;           // The type of unit.
    int id;                // The unique id of this unit during the game.
                            // Ids are consecutive numbers between 0 and
                            //  $4 + 4 * INITIAL\_SOLDIERS\_PER\_CLAN - 1$ 
    int player;            // The player that owns this unit
    Pos pos;                // The position on the board.
    int health;             //  $\geq 0$  for professor: remaining health (-1 if soldier)
    int poison;             //  $\geq 0$  for soldier: amount of poison it has
                            // (-1 if professor)
    int rounds_for_reborn; // only for professors (-1 always for soldiers)
                            // 0 if professor is alive
                            // if it is  $N > 0$ , and we are at round  $R$ , this
                            // professor is now dead and will be reborn at
                            // the end of round  $R + (N - 1)$ 

    // Returns whether a unit is alive. Note that soldiers are always alive
    bool is_alive ();
};

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

// Returns a copy of the cell at (i, j).
Cell cell (int i, int j) const;

// Returns a copy of the cell at p.
Cell cell (Pos p) const;

// Returns the a copy of the unit with identifier id.
Unit unit (int id) const;

// Returns the ids of the soldiers of a player
vector<int> soldiers (int pl) const;

// Returns the id of the professor of a player
int professor (int pl) const;

// Returns the current available poison of a player
int available_poison (int pl) const;

// Returns the current available health points of a player
int available_health (int pl) const;

```

```

// Returns the current score of a player.
int score (int pl) const;

// Returns the hints of safety box with identifier id_box
// (note that 0 <= id_box < num_safety_boxes() )
vector<int> safety_box_hints (int id_box) const;

// Returns the percentage of cpu time used so far, in the
// range [0.0 — 1.0] or a value lesser than 0 if the player is dead.
// NOTE: only returns a sensible value in server executions.
// In local executions the returned value is meaningless.
double status (int pl) const;

```

Command actions.

```

// Commands unit with identifier 'id' to move following direction 'dir'.
void move(int id, Dir dir);

// Commands unit with identifier 'id' (a professor) to provide password
// 'sol' to open the safety box whose door is at direction 'dir'.
void open(int id, const vector<int>& sol, Dir d);

// Commands unit with identifier 'id' (a soldier) to fully charge its poison.
void charge(int id);

// Commands unit with identifier 'id' (a professor) to fully heal.
void heal(int id);

```

Initial settings.

```

// Returns the number of players in the game.
int num_players () const;

// Returns the number of rounds a match lasts.
int num_rounds () const;

// Returns the number of rows of the board.
int board_rows () const;

// Returns the number of columns of the board.
int board_cols () const;

```

```

// Returns the initial number of soldiers per clan.
int num_ini_soldiers_per_clan () const;

// Returns the initial number of poison units per soldier.
int soldier_ini_poisons () const;

// Returns the maximum amount of poison units per soldier.
int soldier_max_poisons () const;

// Returns the points obtained por killing a soldier.
int points_for_killing_soldier () const;

// Returns the maximum points of health that a professor can have.
int professor_max_health () const;

// Returns the initial points of health of a professor
// (after reborn or at the start of the game).
int professor_ini_health () const;

// Returns the number of points of health that are decremented when
// a professor is thrown poison at.
int professor_health_decrease () const;

// Returns the points obtained por killing a professor.
int points_for_killing_professor () const;

// Returns the number of rounds needed for a professor to be reborn
// after it is dead.
int professor_rounds_rebirth () const;

// Returns the number of money items in the board.
int num_money () const;

// Returns the minimum number of points the a money item can give.
int min_money_value () const;

// Returns the maximum number of points the a money item can give.
int max_money_value () const;

// Returns the number of health kits in the board.
int num_health_kits () const;

// Returns the number of health points gained if a kit value is collected.
int health_kit_value () const;

```

```

// Returns the number of poison items in the board.
int num_poison_kits () const;

// Returns the number of poison units gained if a poison kit is collected.
int poison_kit_value () const;

// Returns the number of safety boxes in the board.
int num_safety_boxes () const;

// Returns the points given if the gold inside a safety box is collected.
int gold_value () const;

// Returns the size of the hints of any safety box.
int size_hints_per_box () const;

```

Random.

```

// Returns a random integer in [l..u]. u - l + 1 must be between 1 and 106.
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 106.
vector<int> random_permutation (int n);

```