

Summarized API for playing Harry Potter

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. Alive units cannot move diagonally
enum Dir {
    Down, DR, Right, RU, Up, UL, Left, LD
};

// 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               owner; // The player that owns it, otherwise -1.  
    int               id; // The id of a unit if present, or -1 otherwise.  
    bool              food; // Whether it contains a book  
};
```

```
// Defines the type of the unit.  
enum UnitType {  
    Wizard ,  
    Ghost  
};
```

```

// 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.
    int player;            // The player that owns this unit
    Pos pos;              // The position on the board.
    int rounds_pending;    // For a wizard: num of rounds before
                          // conversion is completed (0 if not converting)
                          // For a ghost: num of rounds before it can
                          // perform a spell again
    int next_player;       // For a wizard in conversion: player it will be
                          // transformed to
                          // For a ghost: last round when it was attacked
                          // (-1 if none)
};

// Next 3 to be called only on wizards (more user friendly than accessing
the fields)

// 1.- Returns whether a wizard is in conversion process
bool is_in_conversion_process ( );

// 2.- Returns, for a wizard in conversion process, the number of rounds
before completing the process
int rounds_for_converting ( );

// 3.- Returns, for a wizard in conversion process, the player it will convert
to
int player_to_be_converted_to ( );

// Next 2 only for ghosts (more user friendly than accessing the fields)

// 1.- Returns the last round when a ghost was attacked. Returns -1 if
none.
int last_attack_received ( );

// 2.- Returns, for a ghost, the number of rounds it needs to wait until it
can perform an spell
int resting_rounds ( );

```

```

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

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

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

// Returns the position of Voldemort
Pos pos_voldemort ( ) const;

// Returns the current magic_strength of a player ( strength_points/units )
int magic_strength (int pl) const;

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

// Returns the ingredients of the current spell
vector<int> spell_ingredients () const;

```

Command actions.

```

// Commands unit with identifier id to move following direction dir.
void move (int id, Dir dir);
// Example: move(23,Down);

// Commands unit with identifier id to provide solution 'sol' to the current
// spell. Identifier id should be a ghost.
void spell (int id, const vector<int>& sol);
// Example: spell(1,0,1,0,2,0,2,1,2,1);

```

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 wizards per clan
int num_ini_wizards_per_clan () const;

// Returns the initial number of book items on the board
int num_ini_books () const;

// Returns the initial magic strength of each clan
int clan_ini_magic_strength () const;

// Returns the points obtained after converting a wizard
int points_for_converting_wizard () const;

// Returns the points obtained for each owned cell at the end of a round
int points_per_owned_cell () const;

// Returns the units of magic strength obtained by reading a book
int book_magic_strength () const;

// Returns the number of rounds needed to complete the converting
// process of a wizard
int rounds_for_converting_wizard () const;

// If a ghost is attacked, she will have to wait an additional rounds_attack_resting_ghost()
// number of of rounds before it can perform an spell again
int rounds_attack_resting_ghost ( ) const;

// If a ghost is attacked, she cannot be attacked again for the next rounds_no_attack_ghost()
// rounds
int rounds_no_attack_ghost ( ) 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);
```