Action.hh

```
C/C++
//Commands citizen with identifier id to move following direction dir.
void move(int id, Dir dir);

//Commands builder with identifier id build a barricade in direction dir.
void build(int id, Dir dir);
```

Player.hh

```
C/C++
//Identifier of my player.
  int me () const;
```

Random.hh

```
C/C++
  //Returns a random integer in [l..u]. u - l + 1 must be between 1 and
10^6.
  int random (int l, int u);

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

Settings.hh

```
C/C++
//Returns a string with the game name and version.
static string version ();

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

// Returns the number of days a match lasts.
int num_days () const;

// Returns number rounds each day lasts. It is an even number. Half of these
rounds are daylight and the other half are night.
int num_rounds_per_day () 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 builders.
int num_ini_builders() const;

// Returns the initial number of warriors.
int num_ini_warriors() const;

// Returns the initial number of money items.
int num_ini_money() const;

// Returns the initial number of food items.
int num_ini_food() const;

// Returns the initial number of guns.
int num_ini_guns() const;

// Returns the initial number of bazookas.
int num_ini_bazookas() const;

// Returns the initial life of a builder. It is also her maximum life.
int builder_ini_life() const;

// Returns the initial life of a warrior. It is also her maximum life.
int warrior_ini_life() const;

// Returns the initial life of a citizen type. It is also her maximum life.
int citizen_ini_life(CitizenType ct) const;

// Returns the number of points that collecting one unit of money gives.
int money_points() const;

// Returns the number of points that killing a builder gives.
int kill_builder_points() const;

// Returns the number of points that killing a warrior gives.
int kill_warrior_points() const;

// Returns the increment of life that eating a unit of food gives.
int food_incr_life() const;

// Returns the number of life points lost when losing an attack.
int life_lost_in_attack() const;

// Returns strength of a builder in an attack.
int builder_strength_attack() const;
```

```
// Returns strength of a hammer in an attack.
int hammer_strength_attack() const;

// Returns strength of a gun in an attack.
int gun_strength_attack() const;

// Returns strength of a bazooka in an attack.
int bazooka_strength_attack() const;

// Returns attack strength of a weapon. It returns the attack strength of a
builder if w is NoWeapon.
int weapon_strength_attack(WeaponType w) const;

// Returns strength of a builder when demolishing a barricade.
int builder_strength_demolish() const;

// Returns strength of a hammer when demolishing a barricade.
int hammer_strength_demolish() const;

// Returns strength of a gun when demolishing a barricade.
int gun_strength_demolish() const;

// Returns strength of a bazooka when demolishing a barricade.
int bazooka_strength_demolish() const;

// Returns demolish strength of a weapon. It returns the demolish strength
of a builder if w is NoWeapon.
int weapon_strength_demolish(WeaponType w) const;

// Returns the number of rounds to wait for a builder to be regenerated.
int num_rounds_regen_builder() const;

// Returns the number of rounds to wait for a warrior to be regenerated.
int num_rounds_regen_warrior() const;

// Returns the number of rounds to wait for a citizen to be regenerated.
int num_rounds_regen_citizen(CitizenType ci) const;

// Returns the number of rounds to wait for food to be regenerated.
int num_rounds_regen_food() const;

// Returns the number of rounds to wait for money to be regenerated.
int num_rounds_regen_money() const;

// Returns the number of rounds to wait for a weapon to be regenerated.
int num_rounds_regen_weapon() const;
```

```
// Returns the resistance given to a barricade in a build action.
int barricade_resistance_step() const;

// Returns the maximum resistance a barricade can achieve.
int barricade_max_resistance() const;

// Returns the maximum number of barricades each player can have.
int max_num_barricades () const;

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

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

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

// Returns whether round r is at night.
bool is_round_night (int r) const;

// Returns whether round r is day.
bool is_round_day (int r) const;
```

State.hh:

```
C/C++
// Returns the current round.
int round () const;

// Returns whether the current round is night.
bool is_night () const;

// Returns whether the current round is day.
bool is_day () 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 citizen with identifier id.
Citizen citizen (int id) const;

// Returns the ids of the builders of a player.
```

```
vector<int> builders(int pl) const;

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

// Returns the positions of the barricades owned by a player.
vector<Pos> barricades(int pl) const;

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

weapon

Structs.hh

```
C/C++
// Enum for directions: Down, Right, Up, Left.
enum Dir { Down, Right, Up, Left };

// Struct for positions with various constructors and operators.
struct Pos { int i, j };

// Defines types of bonuses: Money, Food, NoBonus.
enum BonusType { Money, Food, NoBonus };

// Defines types of weapons: Hammer, Gun, Bazooka, NoWeapon.
enum WeaponType { Hammer, Gun, Bazooka, NoWeapon };

// Returns the strongestWeapon
inline strongestWeapon (WeaponType w1, WeaponType w2);

// Defines types of cells: Street, Building.
enum CellType { Street, Building };

// Struct to describe a cell on the board including its type, bonus, weapon,
resistance, barricade owner, and citizen ID.
struct Cell {
  CellType type;        // Type of cell (Street or Building).
  BonusType bonus;      // Type of bonus present.
  WeaponType weapon;    // Type of weapon present.
  int resistance;       // Resistance of barricade, -1 if none.
  int b_owner;          // Owner of the barricade, -1 if none.
  int id;               // ID of  present, -1 if none.
  Cell();               // Default constructor.
  Cell(CellType t, BonusType b, WeaponType w, int r, int o, int i); //
Constructor with all fields.
  bool is_empty() const; // Checks if the cell is empty.
};

// Defines types of citizens: Builder, Warrior.
enum CitizenType { Builder, Warrior };

// Struct to describe a citizen on the board including their type, ID,
player owner, position, weapon, and life points.
struct Citizen {
  CitizenType type;     // Type of citizen (Builder or Warrior).
  int id;               // Unique ID of the citizen.
  int player;           // Owner of the citizen.
  Pos pos;              // Position on the board.
  WeaponType weapon;    // Weapon carried, NoWeapon if none.
  int life;             // Life points, zero indicates dead.
  Citizen();            // Default constructor.
  Citizen(CitizenType t, int i, int pl, Pos p, WeaponType w, int l);
// Constructor with all defining fields.
};
```